Programmation dynamique avec approximation de la fonction valeur

Rémi Munos

▶ **To cite this version:**

Rémi Munos. Programmation dynamique avec approximation de la fonction valeur. Processus décisionnels de Markov et intelligence artificielle, Hermes, pp.19-50, 2008. hal-00830192

HAL Id: hal-00830192
https://hal.science/hal-00830192

Submitted on 4 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Processus Décisionnels de Markov en Intelligence Artificielle

Groupe PDMIA

27 février 2008

Table des matières

PREMIÈRE PARTIE. PRINCIPES GÉNÉRAUX 15

Chapitre 1. Processus Décisionnels de Markov 17
Frédérick GARCIA

1.1. Introduction 17
1.2. Problèmes décisionnels de Markov 18
1.2.1. Processus décisionnels de Markov 18
1.2.2. Les politiques d'actions 20
1.2.3. Critères de performance 22
1.3. Fonctions de valeur 23
1.3.1. Le critère fini 23
1.3.2. Le critère γ-pondéré 24
1.3.3. Le critère total 24
1.3.4. Le critère moyen 25
1.4. Politiques markoviennes 25
1.4.1. Equivalence des politiques histoire-dépendantes et markoviennes 25
1.4.2. Politique markovienne et chaîne de Markov valuée 27
1.5. Caractérisation des politiques optimales 27
1.5.1. Le critère fini 27
1.5.2. Le critère γ-pondéré 29
1.5.3. Le critère total 34
1.5.4. Le critère moyen 36
1.6. Algorithmes de résolution des MDP 40
1.6.1. Le critère fini 40
1.6.2. Le critère γ-pondéré 40
1.6.3. Le critère total 45
1.6.4. Le critère moyen 46
1.7. Conclusion et perspectives 48

Chapitre 2. Apprentissage par renforcement 51

Olivier SIGAUD, Frédérick GARCIA

2.1. Introduction 51
2.1.1. Bref aperçu historique 52
2.2. Apprentissage par renforcement : vue d'ensemble 53
2.2.1. Approximation de la fonction de valeur 53
2.2.2. Méthodes directes et indirectes 54
2.2.3. Apprentissage temporel, non supervisé et par essais et erreurs . . 55
2.2.4. Le dilemme exploration/exploitation 56
2.2.5. Des méthodes incrémentales fondées sur une estimation 58
2.3. Méthodes de Monte Carlo 59
2.3.1. Préliminaires généraux sur les méthodes d'estimation 59
2.3.2. Les méthodes de Monte Carlo 60
2.4. Les méthodes de différence temporelle 62
2.4.1. L'algorithme TD(0) 63
2.4.2. L'algorithme SARSA 64
2.4.3. L'algorithme Q-learning 66
2.4.4. Les algorithmes TD(λ), Sarsa(λ) et Q(λ) 68
2.4.5. Les architectures acteur-critique 69
2.4.6. Différences temporelles avec traces d'éligibilité : TD(λ) 69
2.4.7. De TD(λ) à SARSA (λ) 73
2.4.8. L'algorithme R-learning 75
2.5. Méthodes indirectes : apprentissage d'un modèle 77
2.5.1. Les architectures DYNA 78
2.5.2. L'algorithme E^3 80
2.5.3. L'algorithme R_{max} 81
2.6. Conclusion 82

Chapitre 3. PDM partiellement observables 85
Alain DUTECH, Bruno SCHERRER

3.1. Définition formelle des POMDP 86
3.1.1. Définition d'un POMDP 86
3.1.2. Critères de performance 88
3.1.3. Etat d'information 88
3.1.4. Politique 92
3.1.5. Fonctions de valeur 93
3.2. Problèmes non-markoviens (information incomplète) 95
3.2.1. Politiques adaptées 95
3.2.2. Critère pondéré 96
3.2.3. Algorithmes adaptés et critère moyen adapté 100
3.3. Calculer une politique exacte sur les états d'information 101
3.3.1. Cas général 102
3.3.2. Etats de croyance et fonction de valeur linéaire par morceaux . . 103

3.4. Algorithmes exacts d'itération sur les valeurs 107
3.4.1. Etapes de l'opérateur de programmation dynamique 107
3.4.2. Obtenir une représentation parcimonieuse de V 110
3.4.3. L'algorithme WITNESS 117
3.4.4. Elagage itératif (Iterative pruning) 120
3.5. Algorithmes d'itération sur les politiques 123
3.6. Conclusion et Perspectives 125

Chapitre 4. Une introduction aux jeux stochastiques 127
Andriy BURKOV, Brahim CHAIB-DRAA

4.1. Introduction 127
4.2. Rappel sur la théorie des jeux 128
4.2.1. Quelques définitions de base 128
4.2.2. Jeux statiques en information complète 133
4.2.3. Jeux dynamiques en information complète 137
4.3. Jeux stochastiques 143
4.3.1. Définition et équilibre d'un jeu stochastique 143
4.3.2. Résolution des jeux stochastiques 145
4.3.3. Complexité et extensibilité des algorithmes d'apprentissage multiagent 155
4.3.4. Au-delà de la recherche d'équilibre 157
4.3.5. Discussion 161
4.4. Conclusion et perspectives 162

Chapitre 5. Critères non classiques 165
Matthieu BOUSSARD , Maroua BOUZID , Abdel-Illah MOUADDIB , Régis SABBADIN , Paul WENG

5.1. Introduction 165
5.2. Les approches multicritères 166
5.2.1. Décision multicritère 167
5.2.2. Processus décisionnel de Markov multicritères 168
5.3. Prise en compte de la robustesse dans la résolution des MDP 172
5.4. Processus Décisionnels de Markov Possibilistes 175
5.4.1. Contreparties possibilistes de l'utilité espérée 175
5.4.2. Programmation Dynamique Possibiliste 178
5.4.3. Extensions des MDP possibilistes 183
5.5. MDP algébriques 187
5.5.1. Rappels 188
5.5.2. Définition d'un MDP algébrique 190
5.5.3. Fonctions de valeur d'une politique 191
5.5.4. Conditions 192
5.5.5. Exemples de AMDP 193
5.6. Conclusion 198

DEUXIÈME PARTIE. EXEMPLES D'APPLICATION DES (PO)MDP 201

Chapitre 6. Apprentissage en ligne de la manipulation de micro-objets . . 203
Guillaume LAURENT

6.1. Introduction 203
6.2. Dispositif de manipulation 204
6.2.1. Objectif : le micro-positionnement par poussée 204
6.2.2. Dispositif de manipulation 205
6.2.3. Boucle de commande 206
6.2.4. Représentation du système de manipulation sous la forme d'un MDP 206
6.3. Choix de l'algorithme d'apprentissage par renforcement 207
6.3.1. Caractéristiques du MDP 207
6.3.2. Un algorithme adapté : *STM-Q* 208
6.4. Résultats expérimentaux 210
6.4.1. Mise en œuvre 210
6.4.2. Résultats obtenus 210
6.5. Conclusion 211

Chapitre 7. Conservation de la biodiversité 215
Iadine CHADÈS

7.1. Introduction 215
7.2. Protéger, surveiller ou abandonner : gestion optimale d'espèces secrètes et menacées 216
7.2.1. Surveillance et gestion du tigre de Sumatra 216
7.2.2. Modèle 217
7.2.3. Résultats 217
7.2.4. Extension à plusieurs populations 220
7.3. Les loutres et les abalones peuvent-ils co-exister ? 221
7.3.1. Les abalones et les loutres, deux espèces menacées 221
7.3.2. Modèles 223
7.3.3. Méthodes 226
7.3.4. Résultats 227
7.3.5. Discussion 229
7.4. Conclusion 231

TROISIÈME PARTIE. EXTENSIONS 233

Chapitre 8. DEC-MDP/POMDP 235
Aurélie BEYNIER, François CHARPILLET, Daniel SZER, Abdel-Illah MOUADDIB

8.1. Introduction générale 235

8.2. Observabilité 236
8.3. Processus décisionnels de Markov multi-agents 238
8.3.1. Formalisme 238
8.3.2. Contrôle centralisé 239
8.3.3. Contrôle décentralisé 239
8.4. Contrôle décentralisé et processus décisionnels de Markov 240
8.4.1. Les processus décisionnels de Markov décentralisés 240
8.4.2. Multiagent Team Decision Problem 241
8.4.3. Gestion de la communication dans les DEC-POMDP 244
8.5. Propriétés et classes particulières de DEC-POMDP 247
8.5.1. Transitions, observations et buts 248
8.5.2. DEC-MDP dirigés par les événements 250
8.5.3. Modélisation de DEC-MDP avec contraintes 251
8.6. La résolution des DEC-POMDP 254
8.6.1. Algorithmes de résolution optimaux 254
8.6.2. Algorithmes de résolution approchée 264
8.7. Quelques exemples d'application 270
8.7.1. Robotique mobile exploratoire 270
8.8. Conclusion et perspectives 272

Chapitre 9. Représentations factorisées 275
Thomas DEGRIS, Olivier SIGAUD

9.1. Introduction 275
9.2. Le formalisme des FMDP 276
9.2.1. Représentation de l'espace d'état 276
9.2.2. L'exemple *Coffee Robot* 276
9.2.3. Décomposition et indépendances relatives aux fonctions 278
9.2.4. Indépendances relatives aux contextes 283
9.3. Planification dans les FMDP 284
9.3.1. Itérations structurées sur les valeurs et sur les politiques 284
9.3.2. L'algorithme *Stochastic Planning Using Decision Diagrams* . . . 289
9.3.3. Programmation linéaire approchée dans un FMDP 292
9.4. Conclusion et perspectives 299

Chapitre 10. Approches de résolution en ligne 301
Laurent PÉRET , Frédérick GARCIA

10.1.Introduction 301
10.1.1.Exploiter le temps en ligne 301
10.1.2.Recherche en ligne par simulation 302
10.2.Algorithmes en ligne pour la résolution d'un MDP 303
10.2.1.Algorithmes hors ligne, algorithmes en ligne 303
10.2.2.Formalisation du problème 303
10.2.3.Algorithmes heuristiques de recherche en ligne pour les MDP . . 306

10.2.4.L'algorithme par simulation de Kearns, Mansour et Ng 310
10.2.5.L'algorithme Rollout de Tesauro et Galperin 312
10.3.Contrôler la recherche 314
10.3.1.Bornes sur l'erreur et pathologie de la recherche avant 315
10.3.2.Allocation itérative des simulations 317
10.3.3.Focused Reinforcement Learning 320
10.3.4.Controlled Rollout 325
10.4.Conclusion 327

Chapitre 11. Programmation dynamique avec approximation 329
Rémi MUNOS

11.1.Introduction 330
11.2.Itérations sur les valeurs avec approximation (IVA) 332
11.2.1.Implémentation à partir d'échantillons et apprentissage supervisé 333
11.2.2.Analyse de l'algorithme IVA 335
11.2.3.Illustration numérique 336
11.3.Itérations sur les politiques avec approximation (IPA) 338
11.3.1.Analyse de l'algorithme IPA en norme L^∞ 340
11.3.2.Évaluation approchée d'une politique 342
11.3.3.Approximation linéaire et méthode des moindres carrés 343
11.4.Minimisation directe du résidu de Bellman 350
11.5.Vers une analyse de la programmation dynamique en norme L^p 351
11.5.1.Intuition d'une analyse L^p en programmation dynamique 352
11.5.2.Bornes PAC pour des algorithmes d'A/R 354
11.6.Conclusion et perspectives 355

Chapitre 12. Méthodes de gradient 357
Olivier BUFFET

12.1.Rappels sur la notion de gradient 358
12.1.1.Gradient d'une fonction 358
12.1.2.Descente de gradient 359
12.2.Optimisation d'une politique paramétrée par méthode de gradient . . . 360
12.2.1.Application aux MDP : aperçu 360
12.2.2.Estimation du gradient de f dans un MDP, cas de l'horizon temporel fini 362
12.2.3.Extension au cas de l'horizon temporel infini : critère actualisé, critère moyen 365
12.2.4.Cas partiellement observable 369
12.3.Méthodes "Acteur-Critique" 370
12.3.1.Estimateur du gradient utilisant les Q-valeurs 371
12.3.2.Compatibilité avec l'approximation d'une fonction de valeur . . . 372
12.4.Compléments 375
12.5.Conclusion 378

QUATRIÈME PARTIE. EXEMPLES D'APPLICATION (SUITE) 379

Chapitre 13. Hélicoptère autonome 381
Patrick FABIANI, Florent TEICHTEIL-KÖNIGSBUCH

13.1.Introduction 381
13.2.Présentation du scénario 383
13.2.1.Problème de planification 384
13.2.2.États et actions 385
13.2.3.Incertitudes 386
13.2.4.Critère à optimiser 386
13.2.5.Modèle formel de décision 386
13.3.Architecture de décision embarquée 387
13.3.1.Vue globale 387
13.3.2.Planification multi-tâche sur requête du superviseur 388
13.4.Programmation dynamique stochastique, incrémentale et locale 389
13.4.1.Obtention d'une première politique non optimisée 390
13.4.2.Génération du sous-espace d'états atteignables 391
13.4.3.Optimisation locale de la politique 391
13.4.4.Replanifications locales 392
13.5.Tests en vol et retour d'expérience 392
13.6.Conclusion 394

Chapitre 14. Robotique mobile 397
Simon LE GLOANNEC, Abdel-Illah MOUADDIB

14.1.La mission du robot explorateur 397
14.2.Formalisme d'une mission constituée de tâches progressives 399
14.3.Modélisation MDP / PRU 400
14.3.1.Les états de l'agent 401
14.3.2.Les actions de l'agent 401
14.3.3.La fonction de transition 402
14.3.4.La fonction de récompense 403
14.4.Calcul de la politique de contrôle 403
14.5.Modéliser concrètement une mission 405
14.6.Extensions possibles 406
14.7.Conclusion 406

Chapitre 15. Planification d'opérations 407
Sylvie THIÉBAUX, Olivier BUFFET

15.1.Planification d'opérations 407
15.1.1.Intuition 407
15.1.2.Définitions formelles 410
15.2.MDP 415

15.2.1.Modélisation sous la forme d'un CoMDP 415
15.3.Algorithmes 417
15.3.1.Résolution exacte 417
15.3.2.Résolution heuristique 419
15.3.3.Autres approches à base de modèles 423
15.4.Apprentissage par renforcement : FPG 425
15.4.1.Employer des méthodes approchées 425
15.4.2.Politique paramétrée 426
15.4.3.Méthodes de gradient 428
15.4.4.Améliorations de FPG 429
15.5.Expérimentations 429
15.6.Conclusion et perspectives 430

Bibliographie 433

Index 459

Avant-propos

Le présent volume est le second tome d'un ouvrage constitué de deux tomes, portant sur les problèmes de décision séquentielle dans l'incertain et de l'apprentissage par renforcement, deux classes de problèmes d'intelligence artificielle que l'on peut formaliser dans le cadre des processus décisionnels de Markov. Il a été écrit pour les étudiants, ingénieurs et chercheurs susceptibles de s'intéresser à ces disciplines et ces modèles.

L'opportunité de la rédaction de ces deux tomes est née de la structuration de la communauté « PDMIA », qui regroupe les chercheurs francophones en informatique, mathématiques appliquées et robotique qui utilisent le cadre des processus décisionnels de Markov (PDM) dans le cadre de l'intelligence artificielle (IA). Depuis 2001, des rencontres informelles annuelles puis la conférence JFPDA ont permis de mettre en évidence l'existence d'une activité de recherche significative dans ce domaine de la part de chercheurs francophones, au point de justifier la rédaction d'un ouvrage de référence sur ces travaux en langue française.

Les deux tomes, fortement complémentaires, sont organisés de la manière suivante.

Dans le premier tome, nous proposons une présentation introductive des bases et des principaux cadres de ce domaine (processus décisionnels de Markov, apprentissage par renforcement, processus décisionnels de Markov partiellement observables, jeux de Markov et critères non-classiques) et quelques applications des principes utilisés dans ces différents cadres dans les domaines de la micro-manipulation et de la gestion de systèmes naturels.

Dans le second tome, en supposant acquises les notions présentées dans le premier tome, nous rassemblons une sélection des travaux actuels plus avancés consistant en des extensions de ces cadres généraux (cadre multi-agent décentralisé, représentations

factorisées, représentations approchées, méthodes de gradient) et un choix d'applications concrètes traitées par des équipes appartenant à notre communauté (planification d'opérations, hélicoptère autonome, robotique).

Il n'était pas possible de présenter dans le cadre de cet ouvrage la totalité des axes de recherche de ce domaine globalement très actif au niveau international. Nous donnons ici quelques références pour orienter le lecteur vers quelques aspects que nous n'avons pas couverts. Ainsi, nous avons choisi de ne pas parler d'apprentissage par renforcement en temps continu [**?**], d'apprentissage par renforcement relationnel [**?**], d'apprentissage par renforcement hiérarchique [**?**], de systèmes de classeurs [**?**] ou de représentations prédictives de l'état [LIT 02].

Par ailleurs, nous nous sommes efforcés au sein des différents chapitres de donner des références vers des travaux connexes à ceux présentés ici.

Les activités scientifiques de la communauté PDMIA sont présentées sur internet à l'adresse suivante : `http://www.loria.fr/projets/PDMIA/index.php`. Le lecteur désireux d'entrer en contact avec les auteurs de cet ouvrage y trouvera le moyen de contacter chacun d'entre eux.

PREMIÈRE PARTIE

Principes généraux

Chapitre 1

Processus Décisionnels de Markov

1.1. Introduction

Les problèmes de décision traités dans cet ouvrage sont communément appelés *problèmes de décision séquentielle dans l'incertain*. La première caractéristique de ce type de problèmes est qu'il s'inscrit dans la durée et que ce n'est pas en fait un, mais plusieurs problèmes de décisions en séquence qu'un *agent* (ou décideur ou encore acteur) doit résoudre, chaque décision courante influençant la résolution des problèmes qui suivent. Ce caractère séquentiel des décisions se retrouve typiquement dans les problèmes de planification en intelligence artificielle et relève en particulier des méthodes de plus court chemin dans un graphe. La seconde caractéristique de ces problèmes est liée à l'incertitude des conséquences mêmes de chacune des décisions possibles. Ainsi, l'agent ne sait pas à l'avance précisément quels seront les effets des décisions qu'il prend. En tant que telle, cette problématique relève des théories de la décision dans l'incertain qui proposent de nombreuse voies de formalisation et approches de résolution, en particulier la théorie classique de maximisation de l'utilité espérée.

Les problèmes de décision séquentielle dans l'incertain couplent donc les deux problématiques de décision séquentielle et de décision dans l'incertain. Les *problèmes décisionnels de Markov* (MDP[1]) en sont une formalisation mathématique, qui généralise les approches de plus court chemin dans un environnement stochastique. A la base de ce formalisme, les *processus décisionnels de Markov* (que l'on note aussi MDP) intègrent les concepts d'*état* qui résume la situation de l'agent à chaque instant, d'*action*

Chapitre rédigé par Frédérick GARCIA.

1. Pour *Markov Decision Problem*

(ou décision) qui influence la dynamique de l'état, de *revenu* (ou *récompense*) qui est associé à chacune des transitions d'état. Les MDP sont alors des chaînes de Markov visitant les états, contrôlées par les actions et valuées par les revenus. Résoudre un MDP, c'est contrôler l'agent pour qu'il se comporte de manière optimale, c'est-à-dire de façon à maximiser son revenu. Toutefois, les solutions d'un MDP ne sont pas des décisions ou séquences de décisions, mais plutôt des *politiques*, ou *stratégies*, ou encore *règles de décision*, qui spécifient l'action à entreprendre en chacune des étapes pour toutes les situations futures possibles de l'agent. Du fait de l'incertitude, une même politique peut donner lieu à des séquences d'états / actions très variées selon les aléas.

EXEMPLE.– Illustrons ces concepts de manière plus concrète en prenant l'exemple de l'entretien d'une voiture. La question qui se pose est de décider, en fonction de l'*état* de la voiture (présence de panne, usure, âge, *etc*.), quelle est la meilleure *stratégie* (ne rien faire, remplacer préventivement, réparer, changer de voiture, *etc*.) pour minimiser le coût de l'entretien sur le long terme. Si on fait l'hypothèse que l'on connaît les *conséquences* et le *coût* des différentes actions pour chaque état (par exemple on connaît la *probabilité* qu'un moteur lâche si on ne répare pas une fuite d'huile) alors on peut modéliser ce problème comme un MDP dont la solution nous donnera, en fonction de l'état de la voiture, l'action optimale. Ainsi, la suite des actions prises au fur et à mesure de l'évolution de l'état de la voiture permettra, en moyenne, de minimiser son coût d'entretien.

Le cadre des problèmes décisionnels de Markov et ses généralisations que nous développerons dans des chapitres ultérieurs forment les modèles les plus classiques pour les problèmes de décision séquentielle dans l'incertain. Nous en exposons les bases dans ce chapitre, dans le cas d'un agent qui dispose a priori d'une connaissance parfaite du processus et de son état à tout instant, dont la tâche consiste donc à planifier a priori une politique optimale qui maximise son revenu au cours du temps.

1.2. Problèmes décisionnels de Markov

1.2.1. *Processus décisionnels de Markov*

Les processus décisionnels de Markov sont définis comme des processus stochastiques contrôlés satisfaisant la propriété de Markov, assignant des récompenses aux transitions d'états [BER 87, PUT 94]. On les définit par un quintuplet : (S, A, T, p, r) où :

– S est l'espace d'états dans lequel évolue le processus ;

– A est l'espace des actions qui contrôlent la dynamique de l'état ;

– T est l'espace des temps, ou axe temporel ;

– $p()$ sont les probabilités de transition entre états ;

– $r()$ est la fonction de récompense sur les transitions entre états.

La figure 1.1 représente un MDP sous la forme d'un diagramme d'influence. A chaque instant t de T, l'action a_t est appliquée dans l'état courant s_t, influençant le processus dans sa transition vers l'état s_{t+1}. La récompense r_t est émise au cours de cette transition.

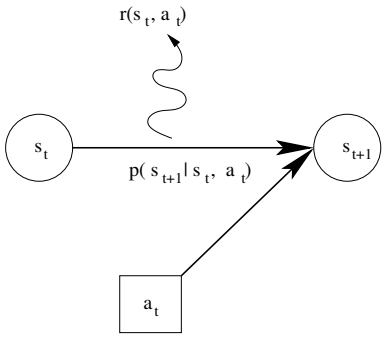


Figure 1.1. *Processus décisionnel de Markov.*

Le domaine T des étapes de décision est un ensemble discret, assimilé à un sous-ensemble de $\mathbb{N}$, qui peut être fini ou infini (on parle d'horizon fini ou d'horizon infini).

Les domaines S et A sont supposés finis, même si de nombreux résultats peuvent être étendus aux cas où S et A sont dénombrables ou continus (voir [BER 95] pour une introduction au cas continu). Dans le cas général, l'espace A peut être dépendant de l'état courant (A_s pour $s \in S$). De même, S et A peuvent être fonction de l'instant t (S_t et A_t). Nous nous limiterons ici au cas classique où S et A sont constants tout au long du processus.

Les probabilités de transition caractérisent la dynamique de l'état du système. Pour une action a fixée, $p(s'|s,a)$ représente la probabilité que le système passe dans l'état s' après avoir exécuté l'action a dans l'état s. On impose classiquement que $\forall s, a$, $\sum_{s'} p(s'|s,a) = 1$. Par ailleurs, on utilise classiquement une représentation matricielle de ces probabilités de transition, en notant P_a la matrice de dimension $|S| \times |S|$ dont les éléments sont $\forall s, s'\ P_{a\,s,s'} = p(s' \mid s,a)$. Les probabilités décrites par $p()$ se décrivent donc par $|A|$ matrices P_a, chacune des lignes de ces matrices ayant pour somme 1 : les P_a sont des matrices *stochastiques*.

Les distributions $p()$ vérifient la propriété fondamentale qui donne son nom aux processus décisionnels de Markov considérés ici. Si on note h_t l'historique à la date t du processus, $h_t = (s_0,\, a_0,\, \ldots,\, s_{t-1},\, a_{t-1},\, s_t)$, alors la probabilité d'atteindre un nouvel état s_{t+1} suite à l'exécution de l'action a_t n'est fonction que de a_t et de l'état courant s_t et ne dépend pas de l'historique h_t. Si on note de façon standard $P(x|y)$ la

probabilité conditionnelle de l'événement x sachant que y est vrai, on a :

$$\forall h_t, a_t, s_{t+1} \quad P(s_{t+1} \mid h_t, a_t) = P(s_{t+1} \mid s_t, a_t) = p(s_{t+1} \mid s_t, a_t)$$

Il faut noter que cela n'implique pas que le processus stochastique induit $(s_t)_{t \in T}$ soit lui-même markovien, tout dépend de la politique de choix des actions a_t.

Comme résultat d'avoir choisi l'action a dans l'état s à l'instant t, l'agent décideur reçoit une récompense, ou revenu, $r_t = r(s, a) \in \mathbb{R}$. Les valeurs de r_t positives peuvent être considérées comme des gains et les valeurs négatives comme des coûts. Cette récompense peut être instantanément perçue à la date t, ou accumulée de la date t à la date $t + 1$, l'important est qu'elle ne dépende que de l'état et de l'action choisie à l'instant courant. La représentation vectorielle de la fonction de récompense $r(s, a)$ consiste en $|A|$ vecteurs r_a de dimension $|S|$.

Une extension classique est de considérer des récompenses $r(s, a)$ aléatoires et l'on considère alors la valeur moyenne $r_t = \bar{r}(s, a)$. En particulier, r_t peut ainsi dépendre de l'état d'arrivée s' selon $r(s, a, s')$. On considère alors la valeur moyenne est $\bar{r}(s, a) = \sum_{s'} p(s'|s, a) r(s, a, s')$. Dans tous les cas, on suppose r_t bornée dans $\mathbb{R}$.

Par ailleurs, comme pour S et A, les fonctions de transition et de récompense peuvent elles-mêmes varier au cours du temps, auquel cas on les note p_t et r_t. Lorsque ces fonctions de varient pas, on parle de processus *stationnaires* : $\forall t \in T, \quad p_t() = p(), \; r_t() = r()$. Par la suite, nous supposerons vérifiée cette hypothèse de stationnarité dans l'étude des MDP à horizon infini.

1.2.2. *Les politiques d'actions*

Les processus décisionnels de Markov permettent de modéliser la dynamique de l'état d'un système soumis au contrôle d'un agent, au sein d'un environnement stochastique. On nomme alors politique (notée π), ou stratégie, la procédure suivie par l'agent pour choisir à chaque instant l'action à exécuter. Deux distinctions sont essentielles ici. Tout d'abord, une politique peut déterminer précisément l'action à effectuer, ou simplement définir une distribution de probabilité selon laquelle cette action doit être sélectionnée. Ensuite, une politique peut se baser sur l'historique h_t du processus, ou peut ne simplement considérer que l'état courant s_t. Nous obtenons ainsi quatre familles distinctes de stratégies, comme indiqué sur le tableau 1.1 :

Pour une politique déterministe, $\pi_t(s_t)$ ou $\pi_t(h_t)$ définit l'action a choisie à l'instant t. Pour une politique aléatoire, $\pi_t(a, s_t)$ ou $\pi_t(a, h_t)$ représente la probabilité de sélectionner a.

Ces quatre familles de politique définissent les quatre ensembles suivants :

politique π_t	déterministe	aléatoire
markovienne	$s_t \longrightarrow a_t$	$a_t,\ s_t \longrightarrow [0,1]$
histoire-dépendante	$h_t \longrightarrow a_t$	$h_t,\ s_t \longrightarrow [0,1]$

Tableau 1.1. *Différentes familles de politiques pour les* MDP.

– Π^{HA} pour les politiques histoire-dépendantes aléatoires ;
– Π^{HD} pour les politiques histoire-dépendantes déterministes ;
– Π^{MA} pour les politiques markoviennes aléatoires ;
– Π^{MD} pour les politiques markoviennes déterministes.

Ces différentes familles de politiques sont imbriquées entre elles, de la plus générale (histoire-dépendante aléatoire) à la plus spécifique (markovienne déterministe), comme le montre la figure 1.2.

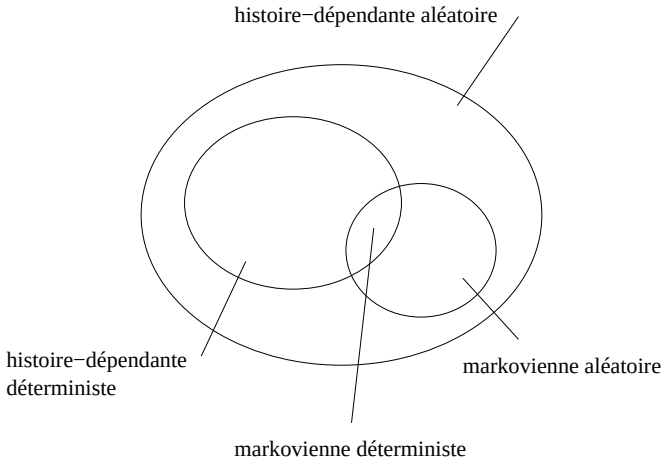


Figure 1.2. *Relations entre les différentes familles de politiques*

Indépendamment de cela et comme pour le processus décisionnel de Markov lui-même, la définition des politiques peut ou non dépendre explicitement du temps. Ainsi, une politique est *stationnaire* si $\forall t\, \pi_t = \pi$. Parmi ces politiques stationnaires, les politiques markoviennes déterministes sont centrales dans l'étude des MDP. Il s'agit du modèle le plus simple de stratégie décisionnelle, on nomme leur ensemble $\mathcal{D}$:

DÉFINITION 1.1.– *Politiques markoviennes déterministes stationnaires*
$\mathcal{D}$ est l'ensemble des fonctions π qui à tout état de S associent une action de A :

$$\pi : s \in S \longrightarrow \pi(s) \in A$$

Un autre ensemble important, noté $\mathcal{D}^A$ est constitué des politiques markoviennes aléatoires stationnaires. Les politiques de $\mathcal{D}$ et $\mathcal{D}^A$ sont très importantes car, comme nous le verrons, $\mathcal{D}$ et $\mathcal{D}^A$ contiennent les politiques optimales pour les principaux critères.

1.2.3. *Critères de performance*

Se poser un problème décisionnel de Markov, c'est rechercher parmi une famille de politiques celles qui optimisent un critère de performance donné pour le processus décisionnel markovien considéré. Ce critère a pour ambition de caractériser les politiques qui permettront de générer des séquences de récompenses les plus importantes possibles. En termes formels, cela revient toujours à évaluer une politique sur la base d'une mesure du *cumul espéré* des récompenses instantanées le long d'une trajectoire, comme on peut le voir sur les critères les plus étudiés au sein de la théorie des MDP, qui sont respectivement :

– le critère fini : $E[r_0 + r_1 + r_2 + \cdots + r_{N-1} \mid s_0]$

– le critère γ-pondéré : $E[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^t r_t + \cdots \mid s_0]$

– le critère total : $E[r_0 + r_1 + r_2 + \cdots + r_t + \cdots \mid s_0]$

– le critère moyen : $\lim_{n\to\infty} \frac{1}{n} E[r_0 + r_1 + r_2 + \cdots + r_{n-1} \mid s_0]$

Les deux caractéristiques communes à ces quatre critères sont en effet d'une part leur formule additive en r_t, qui est une manière simple de résumer l'ensemble des récompenses reçues le long d'une trajectoire et, d'autre part, l'espérance $E[.]$ qui est retenue pour résumer la distribution des récompenses pouvant être reçues le long des trajectoires, pour une même politique et un même état de départ. Ce choix d'un cumul espéré est bien sûr important, car il permet d'établir le *principe d'optimalité de Bellman* [BEL 57] (« les sous-politiques de la politique optimale sont des sous-politiques optimales »), à la base des nombreux algorithmes de programmation dynamique permettant de résoudre efficacement les MDP. On verra au chapitre 5 d'autres critères qui étendent les MDP, pour lesquels le principe d'optimalité n'est plus nécessairement respecté.

Dans la suite de ce chapitre, nous allons successivement caractériser les politiques optimales et présenter les algorithmes permettant d'obtenir ces politiques optimales pour chacun des précédents critères.

1.3. Fonctions de valeur

Les critères fini, γ-pondéré, total et moyen que nous venons de voir permettent de définir une *fonction de valeur* qui, pour une politique π fixée, associe à tout état initial $s \in S$ la valeur du critère considéré en suivant π à partir de s : $\forall \pi \quad V^\pi : S \longrightarrow \mathbb{R}$.

On note $\mathcal{V}$ l'espace des fonctions de S dans $\mathbb{R}$, identifiable à l'espace vectoriel $\mathbb{R}^{|S|}$. L'ensemble $\mathcal{V}$ est muni d'un ordre partiel naturel :

$$\forall U, V \in \mathcal{V} \quad U \leq V \Leftrightarrow \forall s \in S \quad U(s) \leq V(s).$$

L'objectif d'un problème décisionnel de Markov est alors de caractériser et de rechercher – si elles existent – les politiques optimales $\pi^* \in \Pi^{HA}$ telles que

$$\forall \pi \in \Pi^{HA} \quad \forall s \in S \quad V^\pi(s) \leq V^{\pi^*}(s)$$

soit encore

$$\pi^* \in \underset{\pi \in \Pi^{HA}}{\operatorname{argmax}} V^\pi.$$

On note alors $V^* = \max_{\pi \in \Pi^{HA}} V^\pi = V^{\pi^*}$. Dans le cadre des MDP, on recherche donc des politiques optimales meilleures que toute autre politique, quel que soit l'état de départ. Remarquons que l'existence d'une telle politique optimale n'est pas en soi évidente.

La spécificité des problèmes décisionnels de Markov est alors de pouvoir être traduits en terme d'équations d'optimalité portant sur les fonctions de valeur, dont la résolution est de complexité moindre que le parcours exhaustif de l'espace global des politiques de Π^{HA} (la taille du simple ensemble $\mathcal{D}$ est déjà de $|A|^{|S|}$).

1.3.1. *Le critère fini*

On suppose ici que l'agent doit contrôler le système en N étapes, avec N fini. Le critère fini conduit naturellement à définir la fonction de valeur qui associe à tout état s l'espérance de la somme des N prochaines récompenses en suivant la politique π à partir de s :

DÉFINITION 1.2.– *Fonction de valeur pour le critère fini*
Si $T = \{0, \ldots, N-1\}$, *on pose*

$$\forall s \in S \quad V_N^\pi(s) = E^\pi[\sum_{t=0}^{N-1} r_t \mid s_0 = s].$$

Dans cette définition, $E^\pi[.]$ dénote l'espérance mathématique sur l'ensemble des réalisations du MDP en suivant la politique π. E^π est associée à la distribution de probabilité P^π sur l'ensemble de ces réalisations.

Notons qu'il est parfois utile d'ajouter au critère une récompense terminale r_N fonction du seul état final s_N. Il suffit pour cela de considérer une étape artificielle supplémentaire où $\forall s, a \ r_N(s,a) = r_N(s)$. C'est le cas par exemple lorsqu'il s'agit de piloter un système vers un état but en N étapes et à moindre coût.

1.3.2. *Le critère γ-pondéré*

Le critère γ-pondéré, ou critère actualisé, est le critère à horizon infini le plus classique. La fonction de valeur du critère γ-pondéré est celle qui associe à tout état s la limite lorsque N tend vers l'infini de l'espérance en suivant la politique π à partir de s de la somme des N futures prochaines récompenses, pondérées par un *facteur d'actualisation*[2] γ :

DÉFINITION 1.3.– *Fonctions de valeur pour le critère γ-pondéré*
Pour $0 \leq \gamma < 1$, on pose

$$\begin{aligned} \forall s \in S \quad V_\gamma^\pi(s) &= E^\pi[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^t r_t + \cdots \mid s_0 = s] \\ &= E^\pi[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s] \end{aligned}$$

Le terme γ représente la valeur à la date t d'une unité de récompense reçue à la date $t+1$. Reçue à la date $t+\tau$, cette même unité vaudrait γ^τ. Cela implique que les instants de décision $t = 0, 1, 2, \ldots$ de T soient régulièrement répartis sur $\mathbb{N}$. Ce facteur γ a pour principal intérêt d'assurer la convergence de la série en horizon infini. D'un point de vue pratique, il est naturellement utilisé au sein des MDP modélisant des processus de décision économique en posant $\gamma = \frac{1}{1+\tau}$, où τ est le taux d'actualisation.

1.3.3. *Le critère total*

Il est toutefois possible de choisir $\gamma = 1$ dans certains cas particuliers à horizon infini. Lorsque cela a un sens, on pose ainsi :

2. *discount factor* en anglais

DÉFINITION 1.4.– *Fonction de valeur pour le critère total*

$$V^{\pi}(s) = E^{\pi}[\sum_{t=0}^{\infty} r_t \mid s_0 = s]$$

Ce critère est en pratique souvent utilisé pour des problèmes à horizon temporel aléatoire fini non borné : on sait que le processus de décision va s'arrêter à une étape terminale, mais on ne peut borner cet instant. Ce type de modèle est particulièrement utilisé dans des applications de type *optimal stopping* (à tout instant, la décision de l'agent porte simplement sur l'arrêt ou non du processus aléatoire), ou plus généralement de type jeux et paris.

1.3.4. *Le critère moyen*

Lorsque la fréquence des décisions est importante, avec un facteur d'actualisation proche de 1, ou lorsqu'il n'est pas possible de donner une valeur économique aux récompenses, on préfère considérer un critère qui représente la moyenne des récompenses le long d'une trajectoire et non plus leur somme pondérée. On associe ainsi à une politique l'espérance du gain moyen par étape. On définit alors le gain moyen $\rho^{\pi}(s)$ associé à une politique particulière π et à un état s :

DÉFINITION 1.5.– *Le gain moyen*

$$\rho^{\pi}(s) = \lim_{n \to \infty} E^{\pi}[\frac{1}{n} \sum_{t=0}^{n-1} r_t \mid s_0 = s]$$

Pour le critère moyen, une politique π^* est dite *gain-optimale* si $\rho^{\pi^*}(s) \geq \rho^{\pi}(s)$ pour toute politique π et tout état s.

Ce critère est particulièrement utilisé dans des applications de type gestion de file d'attente, de réseau de communication, de stock etc.

1.4. Politiques markoviennes

1.4.1. *Equivalence des politiques histoire-dépendantes et markoviennes*

Nous allons établir ici une propriété fondamentale des MDP pour ces différents critères, qui est d'accepter comme politiques optimales des politiques simplement markoviennes, sans qu'il soit nécessaire de considérer l'espace total Π^{HA} des politiques histoire-dépendantes.

PROPOSITION 1.1.– *Soit $\pi \in \Pi^{HA}$ une politique aléatoire histoire-dépendante. Pour chaque état initial $x \in S$, il existe alors une politique aléatoire markovienne $\pi' \in \Pi^{MA}$ telle que*

1) $V_N^{\pi'}(x) = V_N^{\pi}(x)$,

2) $V_\gamma^{\pi'}(x) = V_\gamma^{\pi}(x)$,

3) $V^{\pi'}(x) = V^{\pi}(x)$,

4) $\rho^{\pi'}(x) = \rho^{\pi}(x)$

PREUVE.– Soit $x \in S$ et π une politique aléatoire histoire-dépendante. Soit π' la politique aléatoire markovienne définie à partir de π et x selon :

$$\forall t = 0, 1, \ldots, \forall s \in S, \forall a \in A \quad \pi'(a_t = a, s_t = s) = P^{\pi}(a_t = a \mid s_t = s, s_0 = x)$$

On a ainsi $P^{\pi'}(a_t = a \mid s_t = s) = P^{\pi}(a_t = a \mid s_t = s, s_0 = x)$. On montre alors par récurrence sur t que $P^{\pi}(s_t = s, a_t = a \mid s_0 = x) = P^{\pi'}(s_t = s, a_t = a \mid s_0 = x)$.

L'égalité est directe pour $t = 0$. Pour $t > 0$, en supposant établie la propriété jusqu'à $t - 1$, on a

$$\begin{aligned} P^{\pi}(s_t = s \mid s_0 = x) &= \sum_{i \in S} \sum_{a \in A} P^{\pi}(s_{t-1} = i, a_{t-1} = a \mid s_0 = x) p(s \mid i, a) \\ &= \sum_{i \in S} \sum_{a \in A} P^{\pi'}(s_{t-1} = i, a_{t-1} = a \mid s_0 = x) p(s \mid i, a) \\ &= P^{\pi'}(s_t = s \mid s_0 = x) \end{aligned}$$

D'où $P^{\pi'}(s_t = s, a_t = a \mid s_0 = x) = P^{\pi'}(a_t = a \mid s_t = s) P^{\pi'}(s_t = s \mid s_0 = x) = P^{\pi}(a_t = a \mid s_t = s, s_0 = x) P^{\pi}(s_t = s \mid s_0 = x) = P^{\pi}(s_t = s, a_t = a \mid s_0 = x)$, ce qui établit la récurrence.

On conclut en remarquant alors que pour tout $x \in S$

$$\begin{aligned} V_N^{\pi}(x) &= \sum_{t=0}^{t=N-1} E^{\pi}[r(s_t, a_t) \mid s_0 = x] \\ V_\gamma^{\pi}(x) &= \sum_{t=0}^{t=\infty} \gamma^t E^{\pi}[r(s_t, a_t) \mid s_0 = x] \\ V^{\pi}(x) &= \sum_{t=0}^{t=\infty} E^{\pi}[r(s_t, a_t) \mid s_0 = x] \\ \rho^{\pi}(x) &= \lim_{n \to \infty} \frac{1}{n} \sum_{t=0}^{t=n-1} E^{\pi}[r(s_t, a_t) \mid s_0 = x], \end{aligned}$$

et

$$E^{\pi}[r(s_t, a_t) \mid s_0 = x] = \sum_{s \in S} \sum_{a \in A} r(s, a) P^{\pi}(s_t = s, a_t = a \mid s_0 = x).$$

□

Ce résultat permet d'affirmer que lorsque l'on connaît l'état initial (ou une distribution de probabilité sur l'état initial), toute politique histoire-dépendante aléatoire peut être remplacée par une politique markovienne aléatoire ayant la même fonction de valeur.

1.4.2. *Politique markovienne et chaîne de Markov valuée*

Pour toute politique markovienne $\pi \in \Pi^{MA}$, le processus décrit par l'état s_t vérifie, pour tout $s_0, s_1, \cdots, s_{t+1}$, $P^{\pi}(s_{t+1} \mid s_0, s_1, \cdots, s_t) = \sum_{a \in A} P^{\pi}(a_t = a \mid s_0, s_1, \cdots, s_t) P^{\pi}(s_{t+1} \mid s_0, s_1, \cdots, s_t, a_t = a) = \sum_{a \in A} \pi(a, s_t) P^{\pi}(s_{t+1} \mid s_t, a_t = a) = P^{\pi}(s_{t+1} \mid s_t)$.

Il s'agit donc d'un processus markovien, qui forme une chaîne de Markov dont la matrice de transition notée P_{π} est définie par

$$\forall s, s' \quad P_{\pi s,s'} = P^{\pi}(s_{t+1} = s' \mid s_t = s) = \sum_{a} \pi(a, s) p(s' \mid s, a).$$

Dans le cas où π est déterministe ($\pi \in \Pi^{MD}$), $P_{\pi s,s'}$ est simplement égal à $p(s' \mid s, \pi(s))$. La matrice P_{π} est construite simplement en retenant pour chaque état s la ligne correspondante dans la matrice P_a avec $a = \pi(s)$. De même, on note r_{π} le vecteur de composante $r(s, \pi(s))$ pour $\pi \in \Pi^{MD}$ et $\sum_a \pi(a, s) r(s, a)$ pour $\pi \in \Pi^{MA}$.

Le triplet (S, P_{π}, r_{π}) définit ce que l'on nomme un *processus de Markov valué*, ou *chaîne de Markov valuée*. Il s'agit simplement d'une chaîne de Markov avec des revenus associés aux transitions. Nous verrons qu'évaluer une politique π consiste alors à calculer certaines grandeurs asymptotiques (pour les critères infinis) caractéristiques de la chaîne de Markov valuée associée.

1.5. Caractérisation des politiques optimales

1.5.1. *Le critère fini*

1.5.1.1. *Equations d'optimalité*

Supposons que l'agent se trouve dans l'état s lors de la dernière étape de décision, confronté au choix de la meilleure action à exécuter. Il est clair que la meilleure décision à prendre est celle qui maximise la récompense instantanée à venir, qui viendra

s'ajouter à celles qu'il a déjà perçues. On a ainsi :

$$\pi^*_{N-1}(s) \in \operatorname*{argmax}_{a\in A} r_{N-1}(s,a)$$

et

$$V^*_1(s) = \max_{a\in A} r_{N-1}(s,a)$$

où π^*_{N-1} est la politique optimale à suivre à l'étape $N-1$ et V^*_1 la fonction de valeur optimale pour un horizon de longueur 1, obtenue en suivant cette politique optimale.

Supposons maintenant l'agent dans l'état s à l'étape $N-2$. Le choix d'une action a va lui rapporter de façon sûre la récompense $r_{N-2}(s,a)$ et l'amènera de manière aléatoire vers un nouvel état s' à l'étape $N-1$. Là, il sait qu'en suivant la politique optimale π^*_{N-1}, il pourra récupérer une récompense moyenne $V^*_1(s')$. Le choix d'une action a à l'étape $N-2$ conduit donc au mieux en moyenne à la somme de récompenses $r_{N-2}(s,a)+\sum_{s'} p_{N-2}(s'|s,a)V^*_1(s')$. Ainsi, le problème de l'agent à l'étape $N-2$ se ramène simplement à rechercher l'action qui maximise cette somme, soit :

$$\pi^*_{N-2}(s) \in \operatorname*{argmax}_{a\in A}\{r_{N-2}(s,a) + \sum_{s'} p_{N-2}(s'|s,a)V^*_1(s')\}$$

et

$$V^*_2(s) = \max_{a\in A}\{r_{N-2}(s,a) + \sum_{s'} p_{N-2}(s'|s,a)V^*_1(s')\}.$$

Ce raisonnement peut s'étendre jusqu'à la première étape de décision, où l'on a donc :

$$\pi^*_0(s) \in \operatorname*{argmax}_{a\in A}\{r_0(s,a) + \sum_{s'} p_0(s'|s,a)V^*_{N-1}(s')\}$$

et

$$V^*_N(s) = \max_{a\in A}\{r_0(s,a) + \sum_{s'} p_0(s'|s,a)V^*_{N-1}(s')\}.$$

Cela conduit ainsi à l'énoncé du théorème suivant :

THÉORÈME 1.1.– *Equations d'optimalité pour le critère fini*
Soit $N < \infty$. Les fonctions de valeurs optimales $V^* = (V^*_N, \ldots, V^*_1)$ sont les solutions uniques du système d'équations

$$\forall s \in S \quad V^*_{n+1}(s) = \max_{a\in A}\{r_{N-1-n}(s,a) + \sum_{s'} p_{N-1-n}(s'|s,a)V^*_n(s')\} \qquad (1.1)$$

avec $n = 0, \ldots, N-1$ et $V_0 = 0$. Les politiques optimales pour le critère fini $\pi^* = (\pi^*_0, \pi^*_1, \ldots, \pi^*_{N-1})$ sont alors déterminées par :

$$\forall s \in S \quad \pi^*_t(s) \in \operatorname*{argmax}_{a\in A}\{r_t(s,a) + \sum_{s'} p_t(s'|s,a)V^*_{N-1-t}(s')\}$$

pour $t = 0, \ldots, N-1$.

On voit donc ici dans le cadre du critère fini que les politiques optimales sont de type markovien déterministe, mais non stationnaire (le choix de la meilleure décision à prendre dépend de l'instant t).

1.5.1.2. *Evaluation d'une politique markovienne déterministe*

Soit une politique π markovienne déterministe. La même démarche permet alors de caractériser sa fonction de valeur V_N^π :

THÉORÈME 1.2.– *Caractérisation de* V_N
Soient $N < \infty$ *et* $\pi = (\pi_0, \pi_1, \ldots, \pi_{N-1})$ *une politique markovienne. Alors* $V_N^\pi = V_N$*, avec* $(V_N, V_{N-1}, \ldots, V_1)$*, solutions du système d'équations linéaires*

$$\forall s \in S \quad V_{n+1}(s) = r_{N-1-n}(s, \pi_{N-1-n}(s)) + \sum_{s'} p_{N-1-n}(s'|s, \pi_{N-1-n}(s)) V_n(s')$$

pour $n = 0, \ldots, N-1$ *et* $V_0 = 0$.

1.5.2. *Le critère γ-pondéré*

Ce critère est le plus classique en horizon infini et celui pour lequel il est assez simple de caractériser la fonction de valeur optimale et les politiques associées. On rappelle que l'on suppose ici, dans le cas de l'horizon infini, que le MDP considéré est stationnaire.

1.5.2.1. *Evaluation d'une politique markovienne stationnaire*

Pour une politique markovienne stationnaire $\pi \in \mathcal{D}^A$, on définit l'opérateur L_π de $\mathcal{V}$ dans $\mathcal{V}$, espace vectoriel muni de la norme max : $\forall V \in \mathcal{V}$, $\| V \| = \max_{s \in S} | V(s) |$.

DÉFINITION 1.6.– *Opérateur* L_π
Pour $\pi \in \mathcal{D}^A$*,*

$$\forall V \in \mathcal{V} \quad L_\pi V = r_\pi + \gamma P_\pi V$$

Un premier résultat permet alors de relier la fonction de valeur V_γ^π d'une politique stationnaire $\pi \in \mathcal{D}^A$ à cet opérateur L_π :

THÉORÈME 1.3.– *Caractérisation de* V_γ^π
Soient $\gamma < 1$ *et* $\pi \in \mathcal{D}^A$ *une politique stationnaire markovienne.*
Alors V_γ^π *est l'unique solution de l'équation* $V = L_\pi V$ *:*

$$\forall s \in S \quad V(s) = r_\pi(s) + \gamma \sum_{s' \in S} P_{\pi,s,s'} V(s') \tag{1.2}$$

et $V_\gamma^\pi = (I - \gamma P_\pi)^{-1} r_\pi$.

PREUVE.– Soit V solution de $V = L_\pi V$. On a donc $(I - \gamma P_\pi)V = r_\pi$. La matrice P_π étant stochastique, toutes les valeurs propres de la matrice γP_π sont de modules inférieurs ou égaux à $\gamma < 1$ et donc la matrice $I - \gamma P_\pi$ est inversible, avec

$$(I - \gamma P_\pi)^{-1} = \sum_{k=0}^{\infty} \gamma^k P_\pi^k$$

d'où

$$V = (I - \gamma P_\pi)^{-1} r_\pi = \sum_{k=0}^{\infty} \gamma^k P_\pi^k r_\pi.$$

$$\begin{aligned}
\text{Or}, \forall s \in S \quad V_\gamma^\pi(s) &= E^\pi[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^t r_t + \cdots \mid s_0 = s] \\
&= \sum_{t=0}^{\infty} \gamma^t E^\pi[r(s_t, a_t) \mid s_0 = s] \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{s' \in S} \sum_{a \in A} P^\pi(s_t = s', a_t = a \mid s_0 = s) r(s', a) \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{s' \in S} \sum_{a \in A} q(a, s') P^\pi(s_t = s' \mid s_0 = s) r(s', a) \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{s' \in S} P^\pi(s_t = s' \mid s_0 = s) r_\pi(s') \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{s' \in S} P_{\pi,s,s'}^t r_\pi(s') \\
&= \sum_{t=0}^{\infty} \gamma^t P_\pi^t r_\pi(s),
\end{aligned}$$

et donc $V = V_\gamma^\pi$. □

1.5.2.2. *Equations d'optimalité*

Rappelons que l'on cherche à résoudre le problème d'optimisation $\forall s \in S, V_\gamma^*(s) = \max_{\pi \in \Pi^{HA}} V_\gamma^\pi(s)$. Une politique π^* est dite optimale si $V_\gamma^{\pi^*} = V_\gamma^*$. De par la propriété 1.1, on a alors

$$\forall s \in S \quad V_\gamma^*(s) = \max_{\pi \in \Pi^{HA}} V_\gamma^\pi(s) = \max_{\pi \in \Pi^{MA}} V_\gamma^\pi(s).$$

Soit donc maintenant l'opérateur L de l'ensemble des fonctions de valeur $\mathcal{V}$ dans lui-même, nommé *opérateur de programmation dynamique* :

DÉFINITION 1.7.– *Opérateur L*

$$\forall V \in \mathcal{V} \quad \forall s \in S \quad LV(s) = \max_{a \in A} \left(r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V(s') \right)$$

soit en notation vectorielle

$$\forall V \in V \quad LV = \max_{\pi \in \mathcal{D}} (r_\pi + \gamma P_\pi V)$$

Le théorème principal concernant l'optimalité des fonctions de valeur pour le critère γ-pondéré est alors le suivant :

THÉORÈME 1.4.– *Equation de Bellman*
Soit $\gamma < 1$. Alors V_γ^* est l'unique solution de l'équation $V = LV$:

$$\forall s \in S \quad V(s) = \max_{a \in A} \left(r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V(s') \right). \tag{1.3}$$

PREUVE.–

Montrons que $\forall V$ et pour $0 \leq \gamma \leq 1$

$$LV = \max_{\pi \in \mathcal{D}} (r_\pi + \gamma P_\pi V) = \max_{\pi \in \mathcal{D}^A} (r_\pi + \gamma P_\pi V)$$

Pour cela, considérons une fonction de valeur V et $\delta \in \mathcal{D}^A$. Pour tout s, du fait du caractère positif des $\delta(a,s)$, on a

$$\sum_a \delta(a,s) \left(r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V(s') \right)$$

$$\begin{aligned}
&\leq \sum_a \delta(a,s) \max_{a'} \left(r(s,a') + \gamma \sum_{s' \in S} p(s' \mid s,a') V(s') \right) \\
&\leq \sum_a \delta(a,s) LV(s) \\
&\leq LV(s)
\end{aligned}$$

Ainsi, pour tout $\delta \in \mathcal{D}^A$

$$r_\delta + \gamma P_\delta V \leq \max_{\pi \in \mathcal{D}} (r_\pi + \gamma P_\pi V)$$

soit

$$\max_{\delta \in \mathcal{D}^A} (r_\delta + \gamma P_\delta V) \leq \max_{\pi \in \mathcal{D}} (r_\pi + \gamma P_\pi V)$$

L'inégalité inverse est immédiate car $\mathcal{D} \subset \mathcal{D}^A$.

Montrons alors que $\forall V, V \geq LV \Rightarrow V \geq V_\gamma^*$, et $V \leq LV \Rightarrow V \leq V_\gamma^*$.

Soit V telle que $V \geq LV$. On a donc

$$V \geq \max_{\pi \in \mathcal{D}} \{r_\pi + \gamma P_\pi V\} = \max_{\pi \in \mathcal{D}^A} \{r_\pi + \gamma P_\pi V\}$$

Soit $\pi = (\pi_0, \pi_1, \ldots) \in \Pi^{MA}$. Pour tout t, $\pi_t \in \mathcal{D}^A$, d'où

$$\begin{aligned} V &\geq r_{\pi_0} + \gamma P_{\pi_0} V \\ &\geq r_{\pi_0} + \gamma P_{\pi_0} (r_{\pi_1} + \gamma P_{\pi_1} V) \\ &\geq r_{\pi_0} + \gamma P_{\pi_0} r_{\pi_1} + \gamma^2 P_{\pi_0} P_{\pi_1} r_{\pi_2} + \cdots + \gamma^{n-1} P_{\pi_0} \cdots P_{\pi_{n-2}} r_{\pi_{n-1}} + \gamma^n P_\pi^n V. \end{aligned}$$

On a donc

$$V - V_\gamma^\pi \geq \gamma^n P_\pi^n V - \sum_{k=n}^{\infty} \gamma^k P_\pi^k r_{\pi_k}, \text{ car } V_\gamma^\pi = \sum_{k=0}^{\infty} \gamma^k P_\pi^k r_{\pi_k},$$

avec $P_\pi^k = P_{\pi_0} P_{\pi_1} \cdots P_{\pi_{k-1}}$. Les deux termes de droite peuvent être rendus aussi petits que désiré pour n suffisamment grand, car $\| \gamma^n P_\pi^n V \| \leq \gamma^n \| V \|$ et $\| \sum_{k=n}^{\infty} \gamma^k P_\pi^k r_{\pi_k} \| \leq \sum_{k=n}^{\infty} \gamma^k R \leq \frac{\gamma^n}{1-\gamma} R$ avec $R = max_{s,a} r(s,a)$. On en déduit

$$V - V_\gamma^\pi \geq 0$$

Cela tant vrai pour toute politique $\pi \in \Pi^{MA}$, on a donc

$$V \geq \max_{\pi \in \Pi^{MA}} V_\gamma^\pi = \max_{\pi \in \Pi^{HA}} V_\gamma^\pi = V_\gamma^*$$

Inversement, soit V telle que $V \leq LV$. On a donc $V \leq \max_{\pi \in \mathcal{D}} \{r_\pi + \gamma P_\pi V\}$ Supposons ce max atteint en π^*. On a donc

$$\begin{aligned} V &\leq r_{\pi^*} + \gamma P_{\pi^*} V \\ &\leq r_{\pi^*} + \gamma P_{\pi^*} (r_{\pi^*} + \gamma P_{\pi^*} V) \\ &\leq r_{\pi^*} + \gamma P_{\pi^*} r_{\pi^*} + \cdots + \gamma^{n-1} P_{\pi^*}^{n-1} r_{\pi^*} + \gamma^n P_{\pi^*}^n V \\ V - V_\gamma^{\pi^*} &\leq -\sum_{k=n}^{\infty} \gamma^k P_{\pi^*}^k r_{\pi^*} + \gamma^n P_{\pi^*}^n V \end{aligned}$$

Les termes de droite pouvant être rendus aussi proches de 0 que désiré, on a donc $V - V_\gamma^{\pi^*} \leq 0$, soit $V \leq V_\gamma^{\pi^*} \leq V_\gamma^*$.

On a ainsi montré que $V \geq LV \Rightarrow V \geq V_\gamma^*$, $V \leq LV \Rightarrow V \leq V_\gamma^*$, ce qui implique que $V = LV \Rightarrow V = V_\gamma^*$: toute solution de l'équation $LV = V$ est nécessairement égale à la fonction de valeur optimale V_γ^*. Montrons maintenant qu'une telle solution existe.

Rappelons pour cela le théorème du point fixe de Banach :

THÉORÈME 1.5.– *Théorème du point fixe de Banach*
Soient $\mathcal{U}$ un espace de Banach (i.e. espace vectoriel normé complet) et T une contraction sur $\mathcal{U}$ (i.e. $\forall u, v \parallel Tu - Tv \parallel \leq \lambda \parallel u - v \parallel$ pour $0 \leq \lambda < 1$). Alors

1) Il existe un unique $u^* \in \mathcal{U}$ tel que $Tu^* = u^*$;

2) Pour tout $u_0 \in \mathcal{U}$, la suite $(u_n)_{n \geq 0}$ définie par

$$u_{n+1} = Tu_n = T^{n+1}u_0$$

converge vers u^*.

L'espace $\mathcal{V}$ muni de la norme max est un espace vectoriel normé de dimension fini donc complet. Il suffit donc de montrer que l'opérateur L est une contraction pour cette norme.

PROPOSITION 1.2.– *Soit $\gamma < 1$. L'opérateur de programmation dynamique L défini par $LV = \max_{\pi \in \mathcal{D}} (r_\pi + \gamma P_\pi V)$ est une contraction sur $\mathcal{V}$.*

PREUVE.– Soient U et V dans $\mathcal{V}$ et $s \in S$.

Supposons $LV(s) \geq LU(s)$. Soit $a_s^* \in \operatorname{argmax}_{a \in A} \left(r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s, a) V(s') \right)$. On a alors

$$\begin{aligned}
0 \quad \leq \quad | LV(s) - LU(s) | \quad &= \quad LV(s) - LU(s) \\
&\leq \quad r(s, a_s^*) + \gamma \sum_{s' \in S} p(s' \mid s, a_s^*) V(s') \\
&\qquad -r(s, a_s^*) - \gamma \sum_{s' \in S} p(s' \mid s, a_s^*) U(s') \\
&\leq \quad \gamma \sum_{s' \in S} p(s' \mid s, a_s^*)(V(s') - U(s')) \\
&\leq \quad \gamma \sum_{s' \in S} p(s' \mid s, a_s^*) \parallel V - U \parallel \\
&\leq \quad \gamma \parallel V - U \parallel
\end{aligned}$$

D'où

$$\parallel LV - LU \parallel = \max_s | LV(s) - LU(s) | \leq \gamma \parallel V - U \parallel .$$

Cette propriété de contraction assure donc l'existence pour l'opérateur L d'un point fixe unique qui est donc égal à V_γ^*. □

On termine alors l'analyse du critère γ-pondéré avec le théorème suivant :

THÉORÈME 1.6.– *Caractérisation des politiques optimales*
Soit $\gamma < 1$. Alors

1) $\pi^* \in \Pi^{HA}$ est optimale $\Leftrightarrow V_\gamma^{\pi^*}$ est solution de $LV = V$ et $V_\gamma^{\pi^*} = V_\gamma^*$;

2) toute politique stationnaire π^* définie par

$$\pi^* \in \operatorname*{argmax}_{\pi \in \mathcal{D}} \{r_\pi + \gamma P_\pi V_\gamma^*\}$$

est une politique optimale.

PREUVE.– La première équivalence est évidente du fait du théorème précédent. Soit alors $\pi^* \in \operatorname{argmax}_{\pi \in \mathcal{D}} \{r_\pi + \gamma P_\pi V_\gamma^*\}$. On a alors

$$\begin{aligned} L_{\pi^*} V_\gamma^* &= r_{\pi^*} + \gamma P_{\pi^*} V_\gamma^* \\ &= \max_{\pi \in \mathcal{D}} \{r_\pi + \gamma P_\pi V_\gamma^*\} \\ &= LV_\gamma^* \\ &= V_\gamma^*. \end{aligned}$$

L'unicité de la solution de $V = L_{\pi^*} V$ démontrée par le théorème 1.3 permet de déduire que $V_\gamma^* = V_\gamma^{\pi^*}$ et donc que π^* est optimale. □

1.5.3. *Le critère total*

L'existence de la limite définissant le critère total ne peut être assurée que sous certaines hypothèses. Nous considérons ici deux classes de problèmes pour lesquelles cette limite existe nécessairement, et est finie pour au moins une politique : les modèles positifs et les modèles négatifs.

DÉFINITION 1.8.– *Soit $\pi \in \Pi^{HA}$. On définit les fonctions V_+^π et V_-^π par*

$$\begin{aligned} V_+^\pi(s) &= E^\pi[\sum_{t=0}^{\infty} \max(r_t, 0) \mid s_0 = s] \\ V_-^\pi(s) &= E^\pi[\sum_{t=0}^{\infty} \max(-r_t, 0) \mid s_0 = s] \end{aligned}$$

On appelle alors respectivement $\mathcal{V}^+$ et $\mathcal{V}^-$ l'ensemble des fonctions positives et négatives de $\mathcal{V}$

On suppose que pour toute politique π et état s, $V_+^\pi(s)$ ou $V_-^\pi(s)$ est fini, ce qui implique l'existence (finie ou infinie) de la limite V^π avec

$$\forall s \in S \quad V^\pi(s) = V_+^\pi(s) - V_-^\pi(s).$$

Les MDP bornés positivement, ou encore positifs, sont tels que
– pour chaque s il existe au moins une action $a \in A$ avec $r(s,a) \geq 0$ et
– $V_+^\pi(s) < \infty$ pour tout $s \in S$ et pour tout $\pi \in \Pi^{HA}$.

Les MDP négatifs sont tels que
– $r(s,a) \leq 0$ pour tout $s \in S$ et $a \in A$ et
– il existe $\pi \in \Pi^{HA}$ telle que $V^\pi(s) > -\infty$ pour tout $s \in S$.

L'existence d'une politique π pour laquelle $V_\pi(s)$ soit fini pour tout $s \in S$ est typiquement assurée par la présence d'un état absorbant s_∞ à récompense nulle :

$$\forall s_0 \quad P^\pi(\exists t^* \, s_{t^*} = s_\infty) = 1$$

avec

$$p(s_\infty \mid s_\infty, \pi(s_\infty)) = 1, \text{ et } r(s_\infty, \pi(s_\infty)) = 0$$

Pour un modèle positif, une politique optimale a un revenu total positif le plus éloigné de 0 possible. L'agent cherche à prolonger le plus possible les trajectoires pour accumuler des revenus positifs. Pour un modèle négatif, une politique optimale a un revenu négatif aussi proche de 0 possible. L'agent cherche à terminer aussi vite que possible en s_∞ pour minimiser les pertes. Les deux modèles ne sont donc pas exactement symétriques.

Nous énonçons ci-dessous quelques résultats importants concernant ces deux modèles. Pour cela, nous introduisons une nouvelle définition des opérateurs L_π et L.

DÉFINITION 1.9.– *Opérateurs L_π et L pour le critère total*
Soit π stationnaire $\in \mathcal{D}^A$,

$$\forall V \in \mathcal{V} \quad L_\pi V = r_\pi + P_\pi V$$

et

$$\forall V \in V \quad LV = \max_{\pi \in \mathcal{D}} (r_\pi + P_\pi V)$$

On montre alors pour les MDP positifs et négatifs les résultats suivants :

THÉORÈME 1.7 [PUT 94].– *Soit un* MDP *positif. Alors*

1) pour tout π *stationnaire* $\in \mathcal{D}$, V^π *est la solution minimale de* $V = L_\pi V$ *dans* $\mathcal{V}^+$.

2) V^* *est la solution minimale de l'équation* $V = LV$ *dans* $\mathcal{V}^+$.

3) une politique $\pi^* \in \Pi^{HA}$ *est optimale* $\Leftrightarrow V^{\pi^*} = LV^{\pi^*}$

4) si $\pi^* \in \operatorname{argmax}_{\pi \in \mathcal{D}} (r_\pi + P_\pi V^*)$ *et si* $\lim_{N \to \infty} P_{\pi^*}^N V^*(s) = 0$ *pour tout* $s \in S$, *alors* π^* *est optimale.*

THÉORÈME 1.8 [PUT 94].– *Soit un* MDP *négatif. Alors*

1) pour tout π *stationnaire* $\in \mathcal{D}$, V^π *est la solution maximale de* $V = L_\pi V$ *dans* $\mathcal{V}^-$.

2) V^* *est la solution maximale de l'équation* $V = LV$ *dans* $\mathcal{V}^-$.

3) toute politique $\pi^* \in \operatorname{argmax}_{\pi \in \mathcal{D}} (r_\pi + P_\pi V^*)$ *est optimale.*

On note que pour un MDP négatif, il peut exister une politique π vérifiant $V^\pi = LV^\pi$ qui ne soit pas optimale (voir l'exemple 7.3.1 dans [PUT 94]).

1.5.4. *Le critère moyen*

L'analyse théorique du critère moyen est plus complexe que pour les précédents critères. Elle fait intervenir le comportement limite du processus markovien valué sous-jacent. Nous nous limitons ici à présenter les résultats principaux, dans le cadre des MDP récurrents (pour toute politique markovienne déterministe, la chaîne de Markov correspondante est constituée d'une unique classe récurrente), unichaînes (chaque chaîne de Markov est constituée d'une unique classe récurrente plus éventuellement quelques états transitoires) ou multichaînes (il existe au moins une politique dont la chaîne de Markov correspondante soit constituée de deux classes récurrentes irréductibles ou plus). On suppose de plus ici que pour toute politique, la chaîne de Markov correspondante est apériodique.

1.5.4.1. *Evaluation d'une politique markovienne stationnaire*

Soit $\pi \in \mathcal{D}^A$ une politique stationnaire et (S, P_π, r_π) le processus de Markov valué qui lui est associé. Rappelons que le gain moyen ou critère moyen est défini par :

$$\forall s \in S \quad \rho^\pi(s) = \lim_{N \to \infty} E^\pi \left[\frac{1}{N} \sum_{t=0}^{N-1} r_\pi(s_t) \mid s_0 = s \right].$$

Sous forme matricielle, on a ainsi

$$\rho^\pi = \lim_{N\to\infty} \frac{1}{N} \sum_{t=0}^{N-1} P_\pi^t r_\pi.$$

Soit $P_\pi^* = \lim_{N\to\infty} \frac{1}{N} \sum_{t=0}^{N-1} P_\pi^t$ la *matrice limite* de P_π. On montre que P_π^* existe et est une matrice stochastique pour tout S fini. De plus, P_π^* vérifie

$$PP_\pi^* = P_\pi^* P = P_\pi^* P_\pi^* = P_\pi^*.$$

Le coefficient $P^*_{\pi,s,s'}$ peut être interprété comme la fraction de temps que le système passera dans l'état s' en étant parti de l'état s. Pour des MDP apériodiques, on a de plus $P_\pi^* = \lim_{N\to\infty} P_\pi^N$ et $P^*_{\pi,s,s'}$ peut être interprété comme la probabilité à l'équilibre d'être dans l'état s' en étant parti de s. Enfin, pour un MDP unichaîne, P_π^* est alors la matrice dont toutes les lignes sont identiques et égales à la *mesure invariante* μ_π de la chaîne contrôlée par la politique π. Ainsi $\forall s, s'\ P^*_{\pi,s,s'} = \mu_\pi(s')$.

De la définition précédente de ρ^π, on déduit que $\rho^\pi = P_\pi^* r_\pi$. Pour un MDP unichaîne on établit ainsi que $\rho(s) = \rho$ est constant pour tout s, avec

$$\rho = \sum_{s\in S} \mu_\pi(s) r_\pi(s)$$

Dans le cas général d'un MDP multichaîne, $\rho(s)$ est constant sur chaque classe de récurrence.

Cette première caractérisation de ρ_π fait intervenir P_π^* qu'il n'est pas facile de calculer. Il est toutefois possible d'obtenir autrement ρ_π, en introduisant une nouvelle fonction de valeur pour le critère moyen, dite *fonction de valeur relative* :

DÉFINITION 1.10.– *Fonction de valeur relative pour le critère moyen*

$$\forall s \in S \quad U^\pi(s) = E^\pi[\sum_{t=0}^{\infty} (r_t - \rho^\pi) \mid s_0 = s]$$

En termes vectoriels, on a ainsi

$$\begin{aligned} U^\pi &= \sum_{t=0}^{\infty} P_\pi^t (r_\pi - \rho^\pi) \\ &= \sum_{t=0}^{\infty} P_\pi^t (r_\pi - P_\pi^* r_\pi) \\ &= \sum_{t=0}^{\infty} (P_\pi^t - P_\pi^*) r_\pi \\ &= (I - P_\pi^* + \sum_{t=1}^{\infty} (P_\pi - P_\pi^*)^t) r_\pi \\ &= (-P_\pi^* + (I - P_\pi + P_\pi^*)^{-1}) r_\pi, \end{aligned}$$

car on montre que la matrice $(I - P_\pi + P_\pi^*)$ est inversible et pour $t > 0$: $(P_\pi - P_\pi^*)^t = P_\pi^t - P_\pi^*$. En multipliant à gauche par $(I - P_\pi + P_\pi^*)$, on en déduit

$$\begin{aligned} U^\pi &= (I - P_\pi + P_\pi^*)^{-1} (I - P_\pi^* (I - P_\pi + P_\pi^*)) r_\pi \\ &= (I - P_\pi + P_\pi^*)^{-1} (I - P_\pi^*) r_\pi. \end{aligned}$$

On note $H_{P_\pi} = (I - P_\pi + P_\pi^*)^{-1}(I - P_\pi^*)$ la *matrice de déviation* de P_π, qui est donc telle que

$$U^\pi = H_{P_\pi} r_\pi.$$

On peut vérifier que H_{P_π} est matrice pseudo-inverse de $(I - P_\pi)$, ce qui établit un lien clair entre cette définition de U^π et l'expression de V_γ^π établie au théorème 1.3.

On montre alors le résultat général suivant, valable pour tout processus de Markov valué qu'il soit récurrent, unichaîne ou multichaîne :

THÉORÈME 1.9 [PUT 94].– *Soit (S, P_π, r_π) un processus de Markov valué associé à une politique stationnaire $\pi \in \mathcal{D}^A$. Alors*

1) si ρ^π et U^π sont le gain moyen et la fonction de valeur relative de π
  a) $(I - P_\pi)\rho^\pi = 0$,
  b) $\rho^\pi + (I - P_\pi)U^\pi = r_\pi$.

2) réciproquement, si ρ et U vérifient les deux égalités précédentes, alors
  a) $\rho = P_\pi^* r_\pi = \rho^\pi$,

b) $U = H_{P_\pi} r_\pi + u$, *où* $(I - P_\pi)u = 0$.
c) si de plus $P_\pi^* U = 0$, *alors* $U = H_{P_\pi} r_\pi = U^\pi$.

On retiendra que la fonction de valeur relative U^π est l'unique solution de $(I - P_\pi)U = (I - P_\pi^*)r_\pi$ telle que $P_\pi^* U = 0$, obtenue en utilisant la pseudo-inverse H_{P_π} de $(I - P_\pi)$.

Dans le cas simplifié d'un processus unichaîne, la première équation se simplifie en $\rho_\pi(s) = \rho_\pi$ et la seconde peut s'écrire selon :

$$\forall s \in S \quad U(s) + \rho = r_\pi(s) + \sum_{s' \in S} P_{\pi,s,s'} U(s'). \tag{1.4}$$

Toute solution (ρ, U) de cette équation vérifie alors $\rho = \rho_\pi$ et $U = U_\pi + ke$, où k est un scalaire quelconque et e le vecteur dont toutes les composantes sont égales à 1. Si de plus $\sum_{s \in S} \mu_\pi(s) U(s) = 0$ alors $U = U_\pi$. Cette équation est bien sûr à rapprocher de celle établie pour le critère γ-pondéré.

1.5.4.2. *Equations d'optimalité*

Enonçons maintenant les conditions d'optimalité qu'il est possible d'établir pour le critère moyen. Rappelons que l'on recherche les politiques $\pi^* \in \Pi^{HA}$ telles que

$$\rho_{\pi^*} = \max_{\pi \in \Pi^{HA}} \rho_\pi = \rho^*$$

Le résultat principal énoncé dans le cadre général des MDP multichaînes est le suivant :

THÉORÈME 1.10 [PUT 94].– *Equations d'optimalité multichaîne*
Il existe une solution (ρ, U) *au système d'équations définies pour tout* $s \in S$ *:*

$$\rho(s) = \max_{a \in A} \sum_{s' \in S} p(s' \mid s, a) \rho(s')$$

$$U(s) + \rho(s) = \max_{a \in B_s} \left(r(s,a) + \sum_{s' \in S} p(s' \mid s, a) U(s') \right)$$

avec

$$B_s = \left\{ a \in A \mid \sum_{s' \in S} p(s' \mid s, a) \rho(s') = \rho(s) \right\}$$

On a alors $\rho = \rho^*$.

Dans le cas unichaîne, le gain ρ est constant et $B_s = A$. Les équations d'optimalité se réduisent alors à

$$\forall s \in S \quad U(s) + \rho = \max_{a \in A} \left(r(s,a) + \sum_{s' \in S} p(s' \mid s,a) U(s') \right) \tag{1.5}$$

Le lien entre solutions des équations d'optimalité et politiques optimales est alors établi avec le théorème suivant :

THÉORÈME 1.11 [PUT 94].– *Soit (ρ, U) une solution aux équations d'optimalité. Il existe alors une politique stationnaire markovienne déterministe gain-optimale $\pi^* \in \mathcal{D}$, déterminée par :*

$$\forall s \in S \quad \pi^*(s) \in \operatorname*{argmax}_{a \in B_s} \left(r(s,a) + \sum_{s' \in S} p(s' \mid s,a) U(s') \right)$$

Notons qu'il peut exister des politiques gain-optimales π^* telles que $(\rho^{\pi^*}, U^{\pi^*})$ ne vérifient pas les équations d'optimalité.

Remarquons enfin que les solutions des équations d'optimalité ne sont pas uniques, car si (ρ^*, U) est solution, il en est au moins de même pour $(\rho^*, U + ke)$ pour tout scalaire k. Surtout, il peut y avoir plusieurs fonctions de valeur relative solutions, définissant des politiques différentes, associées au même ρ^* optimal. Il est alors utile de rechercher parmi ces différentes solutions celles qui maximisent la fonction de valeur relative, on parle alors de *bias-optimality*.

1.6. Algorithmes de résolution des MDP

1.6.1. *Le critère fini*

Le cas de l'horizon fini est assez simple. Les équations d'optimalité permettent en effet de calculer récursivement à partir de la dernière étape les fonctions de valeur optimales $V_1^*, \ldots, V_N^*$ selon l'algorithme 1.1.

La complexité temporelle et spatiale de cet algorithme est en $O(N|S|^2|A|)$.

1.6.2. *Le critère γ-pondéré*

Trois grandes familles de méthodes existent pour résoudre de tels MDP : la programmation linéaire, l'itération sur les valeurs et l'itération sur les politiques. Toutes recherchent des politiques optimales dans $\mathcal{D}$.

Algorithme 1.1 : Programmation dynamique à horizon fini

$V_0 \leftarrow 0$
pour $n \leftarrow 0$ **jusqu'à** $N-1$ **faire**
  pour $s \in S$ **faire**
    $V^*_{n+1}(s) = \max_{a \in A}\{r_{N-1-n}(s,a) + \sum_{s'} p_{N-1-n}(s'|s,a)V^*_n(s')\}$
    $\pi_{N-1-n}(s) \in$
    $\operatorname{argmax}_{a \in A}\{r_{N-1-n}(s,a) + \sum_{s'} p_{N-1-n}(s'|s,a)V^*_n(s')\}$
retourner V^*, π^*

1.6.2.1. *Programmation linéaire*

Il est immédiat de vérifier que si $V \in \mathcal{V}$ minimise la fonction $\sum_{s \in S} V(s)$ sous la contrainte $V \geq LV$, alors $V = V^*_\gamma$. En effet, nous avons montré au cours de la preuve du théorème 1.4 que $V \geq LV$ impliquait $V \geq V^*_\gamma$ et donc que $\sum_{s \in S} V(s) \geq \sum_{s \in S} V^*_\gamma(s)$. Une manière de rechercher la fonction de valeur optimale V^*_γ est donc de résoudre le système linéaire associé, comme décrit dans l'algorithme 1.2 ci-dessous.

Algorithme 1.2 : Programmation linéaire pour le critère γ-pondéré

résoudre

$$\min_{V \in \mathcal{V}} \sum_{s \in S} V(s)$$

avec

$$V(s) \geq r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a)V(s'), \quad \forall s \in S,\, a \in A$$

pour $s \in S$ **faire**
  $\pi(s) \in \operatorname{argmax}_{a \in A}\{r(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s')\}$
retourner V, π

Cette approche a été proposée initialement par [D'E 63]. Si n et m sont les tailles respectives de S et A, avec $p()$ et $r()$ codées sur b bits, la complexité d'un tel algorithme de programmation linéaire sur les rationnels est polynomiale en $|S|$, $|A|$, b, avec des temps de résolution assez lents [LIT 95c]. Nous verrons toutefois au chapitre 9 que des méthodes de programmation linéaire peuvent s'avérer très efficaces dans le cadre des MDP admettant une représentation factorisée.

1.6.2.2. *Algorithme d'itération sur les valeurs*

L'approche la plus classique se base aussi sur la résolution directe de l'équation d'optimalité de Bellman $V = LV$, en utilisant pour cela une méthode itérative de type point fixe, d'où son nom anglais de *value iteration* [BEL 57, BER 87, PUT 94].

Comme le prouve le théorème 1.4, la solution de l'équation de Bellman est obtenue comme limite de la suite $V_{n+1} = LV_n$, quelle que soit l'initialisation de V_0. Il est alors établi qu'un nombre maximum d'itérations polynomial en $|S|$, $|A|$, b, $1/(1-\gamma)\log(1/(1-\gamma))$ est nécessaire pour atteindre π^*, chaque itération étant de complexité $O(|A||S|^2)$ [PAP 87]. Au delà de ce nombre d'itérations, la suite V_n est de plus en plus proche de V^* mais la politique correspondante $\pi_n = \pi^*$ ne change plus.

En pratique, plusieurs conditions d'arrêt de l'itération peuvent être envisagées. La plus classique consiste à stopper l'itération lorsque $\| V_{n+1} - V_n \| < \epsilon$, où ϵ est un seuil d'erreur fixé a priori. On aboutit à l'algorithme 1.3 suivant :

Algorithme 1.3 : Algorithme d'itération sur les valeurs - Critère γ-pondéré

initialiser $V_0 \in \mathcal{V}$
$n \leftarrow 0$
répéter
  pour $s \in S$ **faire**
    $V_{n+1}(s) = \max_{a \in A} \left\{ r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V_n(s') \right\}$
  $n \leftarrow n+1$
jusqu'à $\| V_{n+1} - V_n \| < \epsilon$
pour $s \in S$ **faire**
  $\pi(s) \in \text{argmax}_{a \in A} \{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_n(s') \}$
retourner V_n, π

On montre alors que $\| V_n - V_\gamma^* \| < \epsilon'$ avec $\epsilon' = \frac{2\gamma}{1-\gamma}\epsilon$ (voir chapitre 11).

Il est possible d'améliorer la vitesse de convergence de l'algorithme d'itération sur les valeurs en modifiant légèrement le calcul de V_{n+1}. L'idée consiste à utiliser $V_{n+1}(s)$ à la place de $V_n(s)$ lorsque cette valeur a déjà été calculée. On définit ainsi l'algorithme de Gauss-Seidel, en numérotant les états de S de 1 à $|S|$ (algorithme 1.4).

Cette idée peut encore être généralisée au cas où les états mis à jour à chaque itération sont sélectionnés aléatoirement parmi S. On définit ainsi la programmation dynamique asynchrone [BER 89].

Il est enfin possible d'optimiser encore l'algorithme en éliminant dès que possible des actions qui s'avèrent être définitivement non optimales. Cela permet ainsi de réduire la complexité de l'opération de maximisation sur A.

1.6.2.3. *Algorithme d'itération sur les politiques*

La dernière classe importante d'algorithmes de résolution est constituée des méthodes itérant sur les politiques elles-mêmes.

Algorithme 1.4 : Algorithme d'itération sur les valeurs - Gauss-Seidel

initialiser $V_0 \in \mathcal{V}$
$n \leftarrow 0$
répéter
  pour $i \leftarrow 1$ **jusqu'à** $|S|$ **faire**
    $V_{n+1}(s_i) = \max_{a \in A} \Big\{ r(s,a) + \gamma \sum_{1 \leq j < i} p(s_j \mid s,a) V_{n+1}(s_j) + \gamma \sum_{i \leq j \leq |S|} p(s_j \mid s,a) V_n(s_j) \Big\}$
  $n \leftarrow n+1$
jusqu'à $\| V_{n+1} - V_n \| < \epsilon$
pour $s \in S$ **faire**
  $\pi(s) \in \operatorname{argmax}_{a \in A} \{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_n(s') \}$
retourner V_n, π

Considérons une politique stationnaire $\pi \in \mathcal{D}$ et V_γ^π sa fonction de valeur. L'algorithme d'itération sur les politiques exploite la propriété suivante :

PROPRIÉTÉ 1.1.– *Amélioration sur 1 coup de la politique*
Soit $\pi \in \mathcal{D}$. Toute politique π^+ définie par

$$\pi^+ \in \operatorname*{argmax}_{\delta \in \mathcal{D}} \{ r_\delta + \gamma P_\delta V_\gamma^\pi \}$$

vérifie

$$V_\gamma^{\pi^+} \geq V_\gamma^\pi$$

avec $V_\gamma^{\pi^+} = V_\gamma^\pi \Leftrightarrow \pi = \pi^*$.

PREUVE.– On a

$$\begin{aligned} r_{\pi^+} + \gamma P_{\pi^+} V_\gamma^\pi &= \max_{\delta \in \mathcal{D}} \{ r_\delta + \gamma P_\delta V_\gamma^\pi \} \\ &\geq r_\pi + \gamma P_\pi V_\gamma^\pi \\ &\geq V_\gamma^\pi \end{aligned}$$

car $V_\gamma^\pi = r_\pi + \gamma P_\pi V_\gamma^\pi$. D'où

$$\begin{aligned} r_{\pi^+} + \gamma P_{\pi^+} V_\gamma^{\pi^+} + \gamma P_{\pi^+}(V_\gamma^\pi - V_\gamma^{\pi^+}) &\geq V_\gamma^\pi \\ V_\gamma^{\pi^+} - \gamma P_{\pi^+} V_\gamma^{\pi^+} &\geq V_\gamma^\pi - \gamma P_{\pi^+} V_\gamma^\pi \\ (I - \gamma P_{\pi^+}) V_\gamma^{\pi^+} &\geq (I - \gamma P_{\pi^+}) V_\gamma^\pi \\ V_\gamma^{\pi^+} &\geq V_\gamma^\pi \end{aligned}$$

car si $u \geq v$, $(I - \gamma P_{\pi^+})^{-1}u = u + \gamma P_{\pi^+}u + \gamma^2 P_{\pi^+}^2 u^2 \cdots \geq v + \gamma P_{\pi^+}v + \gamma^2 P_{\pi^+}^2 v^2 \cdots \geq (I - \gamma P_{\pi^+})^{-1}v$.

L'égalité n'est possible que si $\max_{\delta \in \mathcal{D}}\{r_\delta + \gamma P_\delta V_\gamma^\pi\} = V_\gamma^\pi$, soit $V_\gamma^\pi = V_\gamma^*$. $\square$

L'algorithme d'itération sur les politiques se décline donc ainsi (algorithme 1.5) : soit la politique π_n à l'itération n. Dans une première étape, on résout le système d'équations linéaires $V_n = L_{\pi_n}V_n$ puis, dans un second temps, on améliore la politique courante en posant $\pi_{n+1} \in \text{argmax}_{\delta \in \mathcal{D}}\{r_\delta + \gamma P_\delta V_n\}$. On stoppe l'algorithme lorsque $\pi_n = \pi_{n+1}$.

La suite V_n, croissante et bornée par V_γ^*, converge. Comme il y a un nombre fini de politiques, la suite π_n converge alors en un nombre fini d'itération. A la limite, $V_n = V_\gamma^*$ et π_n est optimale.

Algorithme 1.5 : Algorithme d'itération sur les politiques - Critère γ-pondéré

initialiser $\pi_0 \in \mathcal{D}$
$n \leftarrow 0$
répéter
  résoudre

$$V_n(s) = r(s, \pi_n(s)) + \gamma \sum_{s' \in S} p(s' \mid s, \pi_n(s))V_n(s'), \quad \forall s \in S$$

  pour $s \in S$ **faire**
    $\pi_{n+1}(s) \in \text{argmax}_{a \in A}\left\{r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s, a)V_n(s')\right\}$
  $n \leftarrow n + 1$
jusqu'à $\pi_n = \pi_{n+1}$
retourner V_n, π_{n+1}

La complexité de l'algorithme d'itération sur les politiques est en $O(|A||S|^2) + O(|S|^3)$ par itération, avec un nombre maximum d'itérations polynomial en $|S|$, $|A|$, b à γ constant [PAP 87].

Là aussi, il est possible d'améliorer l'efficacité de cet algorithme en simplifiant la phase d'évaluation de la politique courante π_n. Une approche classique consiste à résoudre l'équation $V_n = L_{\pi_n}V_n$ de manière itérative, comme pour l'itération sur les valeurs, mais à s'arrêter au bout d'un faible nombre d'itérations. L'utilisation de ce principe conduit à l'algorithme modifié d'itération sur les politiques (algorithme 1.6).

Cet algorithme combine les caractéristiques de l'itération sur les valeurs et de l'itération sur les politiques. Il converge pour tout δ vers une politique optimale pour

Algorithme 1.6 : Algorithme modifié d'itération sur les politiques - Critère γ-pondéré

initialiser $V_0 \in \mathcal{V}$ tel que $LV_0 \geq V_0$
$flag \leftarrow 0$
$n \leftarrow 0$
répéter
  pour $s \in S$ **faire**
    $\pi_{n+1}(s) \in \text{argmax}_{a \in A} \left\{ r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V_n(s') \right\}$
    ($\pi_{n+1}(s) = \pi_n(s)$ si possible)
    $V_n^0(s) = \max_{a \in A} \left\{ r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V_n(s') \right\}$
  $m \leftarrow 0$
  si $\| V_n^0 - V_n \| < \epsilon$ **alors** $flag \leftarrow 1$
  sinon
    répéter
      pour $s \in S$ **faire**
        $V_n^{m+1}(s) = r(s, \pi_{n+1}(s)) + \gamma \sum_{s' \in S} p(s' \mid s, \pi_{n+1}(s)) V_n^m(s')$
      $m \leftarrow m + 1$
    jusqu'à $\| V_n^{m+1} - V_n^m \| < \delta$
    $V_{n+1} \leftarrow V_n^m$
    $n \leftarrow n + 1$
jusqu'à $flag = 1$
retourner V_n, π_{n+1}

$\epsilon \rightarrow 0$, sous l'hypothèse $LV_0 \geq V_0$. Cette condition est par exemple vérifiée pour le choix suivant de V_0 :

$$V_0(s) = \frac{1}{1-\gamma} min_{s' \in S} min_{a \in A} r(s', a)$$

pour tout $s \in S$.

En pratique, les algorithmes d'itération sur les politiques et, en particulier, l'algorithme modifié d'itération sur les politiques, apparaissent plus efficaces que les algorithmes d'itération sur les valeurs et doivent leur être préférés.

1.6.3. *Le critère total*

1.6.3.1. MDP *positifs*

On montre pour les modèles bornés positivement que l'algorithme d'itération sur les valeurs converge de manière monotone vers V^* sous l'hypothèse que V_0 vérifie $0 \leq V_0 \leq V^*$.

En ce qui concerne l'algorithme d'itération sur les politiques adapté au cas des MDP positifs (algorithme 1.7), on impose une condition sur V_0 qui implique que V_n reste dans $\mathcal{V}^+$ pour tout n. Le calcul de V_n peut être mené en forçant à 0 la valeur $V_n(s)$ pour tous les états récurrents de la chaîne définie par P_{π_n}. On montre alors que cet algorithme converge en un nombre fini d'itérations vers V^* et π^*. De même, sous les hypothèses $LV_0 \geq V_0$ et $V_0 \leq V^*$, on montre que l'algorithme modifié d'itération sur les politiques converge vers une politique optimale. En pratique $V_0 = 0$ est une condition suffisante.

Algorithme 1.7 : Algorithme d'itération sur les politiques - Critère total - MDP positifs

initialiser $\pi_0 \in \mathcal{D}$ avec $r_{\pi_0} \geq 0$
$n \leftarrow 0$
répéter
  calculer la solution minimale de

$$V_n(s) = r(s, \pi_n(s)) + \sum_{s' \in S} p(s' \mid s, \pi_n(s)) V_n(s'), \quad \forall s \in S$$

  pour $s \in S$ **faire**
    $\pi_{n+1}(s) \in \operatorname{argmax}_{a \in A} \left\{ r(s,a) + \sum_{s' \in S} p(s' \mid s, a) V_n(s') \right\}$
    ($\pi_{n+1}(s) = \pi_n(s)$ si possible)
  $n \leftarrow n + 1$
jusqu'à $\pi_n = \pi_{n+1}$
retourner V_n, π_{n+1}

1.6.3.2. MDP *négatifs*

On montre que l'algorithme d'itération sur les valeurs converge de manière monotone vers V^* pour toute condition initiale $V^* \leq V_0 \leq 0$.

Par contre, ce n'est pas le cas pour les algorithmes d'itération sur les politiques, qui peuvent s'arrêter sur des politiques sous-optimales. Il en est de même pour l'algorithme modifié d'itération sur les politiques.

1.6.4. *Le critère moyen*

En ce qui concerne le critère moyen, de même que pour le critère γ-pondéré, on dispose de nombreux algorithmes de programmation dynamique pour calculer des politiques gain-optimales. On présente ici les deux principaux, dans le cas simplifié de MDP unichaînes (toutes les politiques sont unichaînes) pour lesquels le gain moyen ρ est constant. Le test d'arrêt est ici basé sur l'emploi de la semi-norme span sur $\mathcal{V}$: $\forall V \in \mathcal{V}$, $\operatorname{span}(V) = \max_{s \in S} V(s) - \min_{s \in S} V(s)$. Contrairement à $\| V \|$ qui

mesure l'écart de V à 0, la semi-norme $\mathrm{span}(V)$ mesure l'écart de V à un vecteur constant.

1.6.4.1. *Algorithme d'itération sur les valeurs relatives*

L'algorithme 1.8 est un algorithme d'itération sur les valeurs relatives $U(s) = V(s) - \rho$.

Algorithme 1.8 : Algorithme d'itération sur les valeurs relatives - Critère moyen

initialiser $U_0 \in \mathcal{V}$
choisir $s^* \in S$
$n \leftarrow 0$
répéter
  $\rho_{n+1} = \max_{a \in A} \left\{ r(s^*, a) + \sum_{s' \in S} p(s' \mid s^*, a) U_n(s') \right\}$
  pour $s \in S$ **faire**
    $U_{n+1}(s) = \max_{a \in A} \left\{ r(s, a) + \sum_{s' \in S} p(s' \mid s, a) U_n(s') \right\} - \rho_{n+1}$
  $n \leftarrow n + 1$
jusqu'à $\mathrm{span}(U_{n+1} - U_n) < \epsilon$
pour $s \in S$ **faire**
  $\pi(s) \in \mathrm{argmax}_{a \in A} \{ r(s, a) + \sum_{s'} p(s' | s, a) U_n(s') \}$
retourner ρ_n, U_n, π

Sous différentes hypothèses techniques, on peut montrer sa convergence pour $\epsilon \to 0$ vers une solution (ρ^*, V^*) des équations d'optimalité (1.5) et donc vers une politique optimale π^* (voir [PUT 94], théorème 8.5.3). Pour un MDP unichaîne, c'est le cas par exemple si $p(s \mid s, a) > 0$ pour tout s et a.

1.6.4.2. *Algorithme modifié d'itération sur les politiques*

L'algorithme 1.9 est un algorithme modifié d'itération sur les politiques, qui ne nécessite pas la résolution de l'équation (1.4) pour évaluer la fonction de valeur relative.

Pour δ élevé, l'algorithme est équivalent à l'itération sur les valeurs (non relative car on ne gère pas ici explicitement le revenu moyen ρ_n). Pour δ proche de 0, on retrouve une itération sur les politiques classique. Sous les mêmes conditions techniques précédentes, on montre que cet algorithme converge pour tout δ vers une politique optimale pour $\epsilon \to 0$. Plus précisément, lorsque l'algorithme s'arrête, on a

$$\min_{s \in S}(V_n^0(s) - V_n(s)) \leq \rho^{\pi_{n+1}} \leq \rho^* \leq \max_{s \in S}(V^0(s) - V_n(s)),$$

ce qui assure $\mid \rho^{\pi_{n+1}} - \rho^* \mid \leq \epsilon$.

Algorithme 1.9 : Algorithme modifié d'itération sur les politiques - Critère moyen

initialiser $V_0 \in \mathcal{V}$
$flag \leftarrow 0$
$n \leftarrow 0$
répéter
  pour $s \in S$ **faire**
    $\pi_{n+1}(s) \in \text{argmax}_{a \in A} \left\{ r(s,a) + \sum_{s' \in S} p(s' \mid s,a) V_n(s') \right\}$
    ($\pi_{n+1}(s) = \pi_n(s)$ si possible)
    $V_n^0(s) = \max_{a \in A} \left\{ r(s,a) + \sum_{s' \in S} p(s' \mid s,a) V_n(s') \right\}$
  $m \leftarrow 0$
  si $\text{span}(V_n^0 - V_n) < \epsilon$ **alors** $flag \leftarrow 1$
  sinon
    répéter
      pour $s \in S$ **faire**
        $V_n^{m+1}(s) = r(s, \pi_{n+1}(s)) + \sum_{s' \in S} p(s' \mid s, \pi_{n+1}(s)) V_n^m(s')$
      $m \leftarrow m + 1$
    jusqu'à $\text{span}(V_n^{m+1} - V_n^m) < \delta$
    $V_{n+1} \leftarrow V_n^m$
    $n \leftarrow n + 1$
jusqu'à $flag = 1$
retourner V_n, π_{n+1}

1.7. Conclusion et perspectives

Le cadre des processus décisionnels de Markov, avec les modèles de décision, critères d'optimalité et algorithmes d'optimisation que nous venons de présenter constitue un outil méthodologique de base en intelligence artificielle. Il est en particulier devenu incontournable pour concevoir et analyser les méthodes formelles développées aujourd'hui sur le thème de la décision séquentielle dans l'incertain.

Malgré sa généralité, le cadre théorique que nous avons exposé au cours de ce chapitre n'est toutefois pas exempt de limites en termes théoriques. Tout d'abord, ce cadre suppose de la part de l'agent une parfaite connaissance des fonctions de transition et de récompense qui définissent le problème auquel il est confronté. Nous verrons au chapitre 2 comment l'apprentissage par renforcement permet de relâcher cette hypothèse. Par ailleurs, on suppose que l'agent a directement accès à son état. Or, dans la plupart des situations où l'on représente un agent en interaction avec son environnement, l'agent ne dispose pas d'un tel accès à son état, mais plutôt à des perceptions différenciées qui le renseignent exhaustivement ou non sur sa situation vis-à-vis de son environnement. Nous verrons au chapitre 3 comment formaliser une observation partielle de l'état du monde. De même, une autre limite formelle concerne le caractère

mono-agent du cadre des MDPs. De nombreux problèmes requièrent la modélisation de plusieurs agents évoluant et agissant ensemble au sein du même environnement. Nous présenterons dans les chapitres 4 et 8 les travaux qui étendent les MDPs au cadre multi-agents. Enfin, une autre limitation théorique du cadre des MDP provient de l'expression du critère à optimiser, sous la forme de l'espérance d'une somme de récompenses à maximiser. Nous verrons alors dans le chapitre 5 comment il est possible d'étendre les représentations de l'incertitude et des préférences de l'agent à d'autres formalismes.

Les utilisations de plus en plus nombreuses du cadre des MDP et de ses extensions évoquées ci-dessus pour aborder des problèmes de décision dans des domaines finalisés variés, qui vont de la gestion indutrielle aux agro-écosystèmes en passant par la robotique et les applications militaires, ont amené à considérer de manière de plus en plus sérieuse la question de l'efficacité des algorithmes de résolution proposés. Plusieurs chapitres de cet ouvrage seront ainsi consacrés à des méthodes récentes permettant de dépasser les limitations traditionnelles des algorithmes de programmation dynamique, dont les principales sont l'approximation de la fonction de valeur (chapitre 11), les représentations factorisées (chapitre 9), l'optimisation de politiques paramétrées (chapitre 12) ou encore l'optimisation de décision en ligne (chapitre 10).

Chapitre 2

Apprentissage par renforcement

2.1. Introduction

Par rapport aux méthodes de planification présentées au chapitre 1, dans lesquelles l'agent connaît a priori la fonction de transition et la fonction de récompense du problème décisionnel de Markov auquel il est confronté, les méthodes d'*apprentissage par renforcement* permettent de traiter les situations dans lesquelles les fonctions de transition et de récompense ne sont pas connues a priori.

EXEMPLE.– Reprenons l'exemple de l'entretien d'une voiture abordé dans l'introduction du chapitre précédent (voir page 18). Si on doit maintenant faire l'entretien d'un modèle de voiture que nous n'avons jamais rencontré précédemment et qui nous est livrée sans manuel technique, nous n'avons pas assez de connaissance pour modéliser ce problème sous forme d'un MDP. En effet, il se peut que cette voiture soit, par exemple, plus robuste aux fuites d'huile et donc nous ne connaissons pas les probabilités de panne dans ce cas. De même, nous ne savons pas le prix des pièces de rechange et il est donc impossible de connaître le coût d'une action à l'avance. Une solution possible dans ce cas est d'utiliser les méthodes d'apprentissage par renforcement qui s'appuient sur une succession d'expériences : en expérimentant, nous allons petit à petit pouvoir estimer directement la valeur des actions, au sens de la fonction de valeur d'action décrite au chapitre précédent, pour chaque état de la voiture, sans forcément avoir à apprendre les différentes probabilités de panne. Et cette estimation de la valeur des actions permettra finalement de choisir l'action optimale en fonction de l'état de la voiture.

Chapitre rédigé par Olivier SIGAUD et Frédérick GARCIA.

Le présent chapitre est donc consacré au traitement de ce problème plus complexe que le précédent.

2.1.1. *Bref aperçu historique*

La présentation adoptée dans la suite de ce chapitre fait l'objet d'une reconstruction *a posteriori* qui ne rend pas compte de l'enchaînement historique des idées. Avec un souci de clarté des concepts, notre présentation s'appuiera sur l'ouvrage de Sutton et Barto [SUT 98], qui constitue une synthèse d'une qualité telle qu'il est difficile de s'en démarquer. Cependant, avant d'en venir au cœur de cette présentation, nous donnons un bref aperçu de la succession des étapes qui ont conduit à la formulation actuelle des modèles théoriques de l'apprentissage par renforcement, en nous focalisant sur le développement de modèles informatiques.

La plupart des méthodes algorithmiques de l'apprentissage par renforcement reposent sur des principes simples issus de l'étude de la cognition humaine ou animale, comme par exemple le fait de renforcer la *tendance* à exécuter une action si ses conséquences sont jugées positives, ou encore de faire dépendre ce renforcement de la durée qui sépare la récompense de l'action, ou de la fréquence à laquelle cette action a été testée. Cet arrière-plan psychologique a été présenté dans [SIG 04].

Les premiers travaux en informatique représentatifs du cadre de l'apprentissage par renforcement datent approximativement de 1960. En 1961, Michie [MIC 61] décrit un système capable d'apprendre à jouer au morpion par essais et erreurs. Puis Michie et Chambers [MIC 68] écrivent en 1968 un programme capable d'apprendre à maintenir un pendule inversé à l'équilibre. Parallèlement, Samuel [SAM 59] réalise un logiciel qui apprend à jouer aux dames en utilisant une notion de différence temporelle. Les deux composantes, exploration par essais et erreurs et gestion de séquences d'action par différences temporelles vont être au cœur des modèles ultérieurs. La synthèse entre les deux courants est réalisée par Klopf [KLO 72, KLO 75]. Dans la lignée de ces travaux, les principaux acteurs du développement de l'apprentissage par renforcement en informatique, Sutton et Barto, implémentent en 1981 [SUT 81] un perceptron linéaire dont l'équation de mise à jour dérive directement des théories développées en psychologie expérimentale par Rescorla et Wagner [RES 72]. Jozefowiez fait très clairement apparaître dans sa thèse [JOZ 01] que l'équation de Rescorla-Wagner, qui est d'une importance considérable pour les modèles de l'apprentissage animal, n'est rien d'autre que la version approximée par un perceptron linéaire de l'algorithme $TD(0)$ présenté à la section 2.4.1. Cette équivalence explique que le recours

aux algorithmes d'apprentissage par renforcement soit aussi appelé « programmation neurodynamique » [1].

Sur ces bases, Sutton et Barto proposent en 1983 avec Anderson [BAR 83] une méthode, AHC-LEARNING [2], qui est considérée comme le véritable point de départ des travaux qui font l'objet de ce chapitre. De façon intéressante, AHC-LEARNING repose sur une architecture acteur-critique dont nous verrons à la section 2.4.5, qu'elle est au cœur du dialogue qui s'instaure à présent entre la modélisation informatique et la modélisation neurophysiologique de l'apprentissage par renforcement chez l'animal.

La formalisation mathématique des algorithmes d'apprentissage par renforcement telle que nous la connaissons aujourd'hui s'est développée à partir de 1988 lorsque Sutton [SUT 88] puis Watkins [WAT 89] ont fait le lien entre leurs travaux et le cadre théorique de la commande optimale proposée par Bellman en 1957 avec la notion de MDP [BER 95].

2.2. Apprentissage par renforcement : vue d'ensemble

2.2.1. *Approximation de la fonction de valeur*

Le principe de l'apprentissage par renforcement repose en premier lieu sur une interaction itérée du système apprenant avec l'environnement, sous la forme de l'exécution à chaque instant n d'une action a_n depuis l'état courant s_n, qui conduit au nouvel état s'_n et qui fournit la récompense r_n. Sur la base de cette interaction, une politique est petit à petit améliorée. En pratique toutefois, la plupart des algorithmes d'apprentissage par renforcement ne travaillent pas directement sur la politique, mais passent par l'approximation itérative d'une *fonction de valeur*, issue de la théorie des MDP présentée au chapitre précédent.

La notion de fonction de valeur, qui associe à chaque état possible une estimation de la valeur pour l'agent de se situer en cet état en fonction de l'objectif visé[3], est fondamentale en apprentissage par renforcement. Elle permet de distinguer clairement l'apprentissage par renforcement de toutes les autres méthodes d'optimisation basées sur la simulation, comme les algorithmes génétiques [**?**], la programmation génétique [**?**], le recuit simulé [**?**], etc. qui permettent aussi de construire des politiques

1. Plus précisément, le terme *neuro-dynamic programming* définit l'ensemble des techniques couplant programmation dynamique ou apprentissage par renforcement et méthodes de généralisation [BER 96].

2. *Adaptive Heuristic Critic learning*

3. Il s'agit donc là d'une notion similaire à celle introduite en théorie des jeux sous le nom de fonction d'évaluation.

optimales, mais qui n'exploitent pas pour cela la structure temporelle des problèmes décisionnels considérés, comme le fait l'apprentissage par renforcement à l'échelle de l'expérience (état courant, action, récompense, état suivant).

La plupart des méthodes de l'apprentissage par renforcement que nous allons voir dans ce chapitre sont étroitement liées aux algorithmes de programmation dynamique présentés au chapitre précédent. En effet, le problème de définir et de calculer des politiques optimales a été formalisé dans le cadre des MDP depuis la fin des années 50, et l'apprentissage par renforcement peut être perçu comme une simple extension des algorithmes classiques de programmation dynamique au cas où la dynamique du processus à contrôler n'est pas connue a priori.

2.2.2. *Méthodes directes et indirectes*

Les méthodes d'apprentissage par renforcement sont dites indirectes ou directes selon que l'on maintient ou non un modèle explicite des fonctions de transition et de récompense du MDP que l'on cherche à contrôler. À ce titre, les méthodes de programmation dynamique vues au chapitre précédent peuvent être considérées comme des cas limites de méthodes indirectes, où le modèle maintenu est le modèle exact par hypothèse.

Lorsque le modèle de la dynamique n'est pas connu initialement, les méthodes indirectes doivent donc *identifier* en ligne ce modèle. Dans les cas discrets que nous considérons, cette identification se fait simplement par maximum de vraisemblance. D'autre part, il s'agit de rechercher sur la base du modèle courant une fonction de valeur ou une politique optimale. Il est alors possible d'exploiter les algorithmes classiques de programmation dynamique.

Les méthodes directes ne passent pas par l'identification d'un modèle de la dynamique du système : les paramètres cachés $p(s'|s,a)$ et $r(s,a)$ ne sont pas estimés et seule la fonction de valeur est mise à jour itérativement au cours du temps. Le principal avantage est ici en terme de place mémoire nécessaire et, historiquement, l'apprentissage par renforcement revendiquait d'être une méthode directe. Ainsi, les algorithmes les plus classiques de l'apprentissage par renforcement que nous présentons ci-dessous partagent tous, chacun à sa manière, le même principe général qui est d'approximer à partir d'expériences une fonction de valeur optimale V sans nécessiter la connaissance a priori d'un modèle du processus, et sans chercher à estimer ce modèle à travers les expériences accumulées. Plutôt qu'une fonction V qui associe une valeur à chaque état, nous verrons que l'on peut aussi chercher à approximer une fonction Q qui associe une valeur à chaque action réalisée dans chaque état.

2.2.3. *Apprentissage temporel, non supervisé et par essais et erreurs*

Quelques caractéristiques permettent de distinguer l'apprentissage par renforcement des autres formes d'apprentissage.

En informatique, on distingue d'une part l'apprentissage dit « supervisé », dans lequel un « instructeur » indique à l'apprenant quelle réponse il aurait dû fournir dans un contexte donné et, d'autre part, l'apprentissage dit « non supervisé », dans lequel l'apprenant doit identifier par lui-même la meilleure réponse possible.

Dans le cadre de l'apprentissage supervisé, la meilleure réponse possible est fournie à l'apprenant si bien qu'il n'a pas besoin de la rechercher. C'est le cas, par exemple, avec l'algorithme de rétro-propagation du gradient dans les réseaux de neurones à couches [**?**]. L'apprentissage par renforcement se distingue de l'apprentissage supervisé par le fait que, lorsqu'il reçoit un premier signal d'évaluation, l'apprenant ne sait toujours pas si la réponse qu'il a donnée est la meilleure possible ; il doit essayer d'autres réponses pour déterminer s'il peut recevoir une meilleure évaluation.

Parmi les méthodes d'apprentissage non supervisé, il faut distinguer les méthodes dans lesquelles l'apprentissage se fait sans évaluation, par exemple en mémorisant simplement des associations observées, et les méthodes dans lesquelles une évaluation est fournie à l'apprenant, comme c'est le cas pour l'apprentissage par renforcement. L'apprentissage par renforcement est donc une forme d'apprentissage non supervisé reposant sur une évaluation. La présence de cette évaluation implique un mode de fonctionnement par essais et erreurs.

Enfin, l'apprentissage par renforcement porte sur des séquences temporelles. Lorsque l'on cherche à classer des données décrivant des iris, même si le processus d'apprentissage est itératif (le professeur fournit des exemples l'un après l'autre), le temps ne change rien à l'affaire, au sens où l'ordre dans lequel sont présentés les exemples importe peu. Au contraire, en apprentissage par renforcement, tout choix d'une action exécutée dans un état a des conséquences à plus ou moins long terme et la donnée de la récompense immédiate n'est rien sans les autres données qui correspondent à la suite de l'interaction entre l'agent et l'environnement. On peut en effet être confronté à des *récompenses retardées* et les méthodes de l'apprentissage par renforcement offrent des outils permettant de gérer cette difficulté.

Dès lors, alors que l'apprentissage par simple mémorisation des associations observées peut se faire directement, l'apprentissage par renforcement induit par nature une activité d'exploration de la part de l'agent. Il faut que cet agent explore son environnement pour déterminer dans quelles circonstances il est puni ou récompensé et quelles sont les séquences d'action qui lui permettent d'atteindre les récompenses plutôt que les punitions. Cette nécessité d'explorer est à la source de la présence dans tous les travaux d'apprentissage par renforcement du dilemme posé par le compromis entre exploration et exploitation.

2.2.4. *Le dilemme exploration/exploitation*

Pour régler la politique de façon à maximiser sa récompense sur le long terme, la phase d'apprentissage se trouve confrontée à la nécessité de trouver un compromis entre l'exploitation, qui consiste à refaire les actions dont on connaît déjà la récompense à laquelle elles donnent lieu, et l'exploration, qui consiste à parcourir de nouveaux couples *(état, action)* à la recherche d'une récompense cumulée plus grande, mais au risque d'adopter parfois un comportement sous-optimal. En effet, tant que l'agent n'a pas exploré la totalité de son environnement, il n'est pas certain que la meilleure politique qu'il connaît est la politique optimale.

En conséquence, toutes les preuves de convergence des algorithmes d'apprentissage par renforcement exigent en théorie que toutes les transitions soient expérimentées [WAT 92]. En pratique, toutefois, on se contente d'une exploration partielle qui suffit en général à découvrir une politique satisfaisante.

L'exploration peut porter sur le choix de l'état s_n ou sur celui de l'action a_n. La plupart du temps, le choix le plus naturel concernant s_n est de poser à chaque itération $s_{n+1} = s'_n$, c'est-à-dire de laisser à la dynamique du système le soin de gérer l'exploration de l'espace d'états. D'une part, cela permet de se concentrer sur les zones importantes de l'espace d'états, accélérant ainsi la convergence (c'est aussi une des raisons de l'efficacité d'algorithmes comme RTDP, décrit chapitre 10). D'autre part, c'est souvent nécessaire du fait de la structure des systèmes sur lesquels est réalisé l'apprentissage. Par exemple, dans un contexte robotique, on ne maîtrise pas le choix de l'état suivant. Il est clair toutefois que lorsque s'_n est un état absorbant, c'est-à-dire un état dans lequel le système reste une fois qu'il y est entré, il est nécessaire de réinitialiser le processus en tirant par exemple au hasard un nouvel état s_{n+1} dans S.

La seconde heuristique concerne le choix de l'action a_n. Une action choisie uniformément dans A à chaque itération satisfait bien le critère de convergence, avec une *exploration* maximale, mais un tel choix est peu efficace pour deux raisons. Tout d'abord, la valeur de la meilleure action en chaque état étant aussi souvent mise à jour que la valeur de la plus mauvaise action, l'apprentissage se fait sûrement mais très lentement. D'autre part, les récompenses accumulées au cours de l'apprentissage sont nécessairement moyennes, ce qui peut être inacceptable lorsque cet apprentissage est en prise avec le système dynamique réel et non simulé.

Inversement, le choix à chaque itération de l'action a_n correspondant à la politique optimale courante (on parle d'action gloutonne [4] et de politique gloutonne [5]), n'est pas non plus satisfaisant, car il conduit généralement soit à une politique sous-optimale, soit à la divergence de l'algorithme.

---

4. *greedy-action*
5. *greedy-policy*

Ainsi, les algorithmes d'apprentissage par renforcement retiennent un compromis entre exploration et exploitation qui consiste à suivre la politique optimale courante la plupart du temps, tout en choisissant plus ou moins régulièrement une action aléatoire pour a_n. Comme le dit élégamment Jozefowiez [JOZ 01], compte tenu de la nécessité de cette exploration plus ou moins aléatoire, la résolution d'un problème d'apprentissage par renforcement reste un art plutôt qu'une science.

Plusieurs méthodes de choix de a_n ont été proposées, que l'on classe en deux catégories dites dirigées ou non-dirigées [THR 92].

Les méthodes non-dirigées utilisent peu d'information issues de l'apprentissage autre que la fonction de valeur elle-même. Citons par exemple [BER 96, KAE 98] :

– sur un intervalle de N_1 itérations, suivre la meilleure politique connue, puis sur N_2 itérations, tirer uniformément a_n dans A ;

– les méthodes ϵ-*greedy*, qui consistent à utiliser à chaque itération un tirage semi-uniforme, qui consiste à suivre la meilleure politique connue avec une probabilité $1-\epsilon$, ou à tirer uniformément a_n dans A avec une probabilité ϵ, et $\epsilon \in [0, 1]$;

– les méthodes *softmax*, qui consistent à tirer a_n dans A selon une distribution de Boltzmann, la probabilité associée à l'action a étant

$$p_T(a) = \frac{\exp(-\frac{Q_n(s_n,a)}{T})}{\sum_{a'} \exp(-\frac{Q_n(s_n,a')}{T})}$$
$$\text{avec } \lim_{n\to\infty} T = 0$$

où $Q_n(s, a)$ représente la fonction de valeur associée à la réalisation de l'action a dans l'état s.

Ces différentes fonctions d'exploration font intervenir des paramètres (N_1, N_2, τ et T) qui contrôlent le degré d'exploration dans le choix de a_n. Par exemple, si $N_1 = 0$, T très grand ou $\tau = 1$, l'algorithme d'apprentissage parcourt uniformément toutes les actions. Les cas utiles en pratique sont pour $N_1 > 0$, $T < +\infty$ et $\tau < 1$, qui assurent expérimentalement une convergence beaucoup plus rapide. La méthode de la roue de la fortune [6] est un cas particulier de méthode *softmax* dans laquelle le facteur T de température est constant au lieu de décroître.

Les méthodes dirigées utilisent pour leur part des heuristiques propres au problème de l'exploration, en se basant sur des informations acquises au cours de l'apprentissage. La plupart de ces méthodes reviennent à ajouter à la valeur $Q(s, a)$ d'une action dans un état un *bonus d'exploration* [MEU 96]. Ce bonus peut être local comme dans

6. *roulette wheel selection*

la méthode de l'estimation d'intervalle [7] [**?**], ou propagé d'état à état au cours de l'apprentissage [**?**]. Des définitions simples de ce bonus d'exploration conduisent à des résultats intéressants :

– la *recency-based method* : le bonus est égal à $\varepsilon\sqrt{\delta n_{sa}}$ où δn_{sa} représente le nombre d'itérations parcourues depuis la dernière exécution de l'action a dans l'état s, et où ε est une constante inférieure à 1 ;

– la méthode de l'*uncertainty estimation* : le bonus est égal à $\frac{c}{n_{sa}}$, où c est une constante et n_{sa} représente le nombre de fois où l'action a a déjà été choisie dans l'état s.

Ces différentes méthodes d'exploration peuvent être utilisées indifféremment quel que soit l'algorithme d'apprentissage par différence temporelle auquel on fait appel. En effet, pour tous les algorithmes de différence temporelle que nous allons présenter dans la suite, si l'hypothèse de Markov est vérifiée et si l'on parcourt tous les états un nombre infini de fois, alors les valeurs $V(s_t)$ ou les qualités des actions $Q(s_t, a_t)$ convergent vers les valeurs optimales.

Notons qu'au sein d'un algorithme d'apprentissage par renforcement, il est nécessaire de distinguer la politique d'exploration simulée à chaque itération, qui est une politique aléatoire et la meilleure politique courante, qui est markovienne déterministe et qui tend vers une politique optimale π^*.

Enfin, nous verrons à la section 2.5.2 qu'il existe dans le cadre de l'apprentissage par renforcement indirect des méthodes d'exploration récentes dotés d'une preuve de convergence en fonction polynomiale de la taille du problème.

2.2.5. *Des méthodes incrémentales fondées sur une estimation*

Il existe trois classes d'algorithme d'optimisation du comportement :

• Les algorithmes de programmation dynamique s'appliquent dans le cas où l'agent dispose d'un modèle de son environnement, c'est-à-dire lorsque les fonctions de transition p et de récompense r sont connues a priori. Elles peuvent aussi s'appliquer dans le cas où l'on cherche à apprendre le modèle, donc dans le cadre des méthodes indirectes. Nous y reviendrons à la section 2.5.1.

• Les méthodes de programmation dynamique présentent l'avantage d'être incrémentales. On peut réaliser des itérations successives qui convergent peu à peu vers la fonction de valeur optimale, ce qui permet d'agir sans attendre de réaliser toutes

7. interval estimation

les itérations. Par contre, elles exigent une connaissance parfaite de la fonction de transition et de la fonction de récompense du MDP associé. Les méthodes de Monte Carlo présentent les avantages et inconvénients opposés. Elles ne présupposent aucune connaissance a priori du problème de décision markovien à résoudre, mais elles ne sont pas incrémentales.

• Les méthodes de différence temporelle reposent sur une estimation incrémentale du modèle de l'environnement. Comme les méthodes de Monte Carlo, elles réalisent cette estimation sur la base de l'expérience de l'agent et se passent ainsi d'un modèle du monde. Néanmoins, elles combinent cette estimation avec des mécanismes de propagation locale d'estimation des valeurs tirées de la programmation dynamique, ce qui leur permet de conserver un caractère incrémental.

Les méthodes de différence temporelle, qui constituent le cœur de l'apprentissage par renforcement proprement dit, se caractérisent donc par cette combinaison du recours à l'estimation avec des propriétés d'incrémentalité.

Les méthodes de programmation dynamique ont déjà été présentées au chapitre précédent. Nous consacrons la section suivante aux méthodes de Monte Carlo, avant de nous tourner à la section 2.4 vers les méthodes de différence temporelle.

2.3. Méthodes de Monte Carlo

2.3.1. *Préliminaires généraux sur les méthodes d'estimation*

Nous avons vu au chapitre précédent qu'il existait des méthodes de résolution de l'équation d'optimalité de Bellman qui réclamaient de calculer à chaque itération la fonction de valeur associée à la politique courante (algorithmes de *policy iteration*). Ces algorithmes de programmation dynamique nécessitant la connaissance des probabilités de transition et des fonctions de récompense, des méthodes ont été développées permettant d'estimer au mieux la fonction de valeur V^π d'une politique π fixée, sur la base des seules transitions simulées en suivant cette politique. Ces algorithmes d'apprentissage d'une fonction de valeur peuvent alors être utilisés au sein de méthodes directes en apprentissage par renforcement.

Pour des raisons de clarté, nous nous plaçons à présent dans le cas d'un MDP et d'une politique π telle que la chaîne de Markov associée $p(s_{t+1}|s_t) = p(s_{t+1}|s_t, \pi(s_t))$ conduise pour tout état initial vers un état terminal T absorbant de récompense nulle. On considère donc le critère total et on cherche à estimer $V(s) = E(\sum_0^\infty r_t \mid s_0 = s)$ à partir des seules observations (s_t, s_{t+1}, r_t).

Une façon simple de réaliser cette estimation consiste à simuler des trajectoires à partir de chacun des états s jusqu'à l'état terminal T. Si l'on note $R_k(s)$ la somme cumulée obtenue le long de la trajectoire k en suivant la politique π, alors une estimation

de la fonction de valeur V en s après $k+1$ trajectoires est donnée par :

$$\forall s \in S, V_{k+1}(s) = \frac{R_1(s) + R_2(s) + ... + R_k(s) + R_{k+1}(s)}{k+1} \tag{2.1}$$

Pour ne pas avoir à stocker chacune des R_k reçues, on montre très simplement qu'un tel calcul peut se reformuler de manière incrémentale :

$$\forall s \in S, V_{k+1}(s) = V_k(s) + \frac{1}{k+1}[R_{k+1}(s) - V_k(s)] \tag{2.2}$$

Pour calculer $V_{k+1}(s)$, il suffit donc de stocker $V_k(s)$ et k. Plutôt que de stocker le nombre k d'expériences réalisées, on utilise généralement une formule d'estimation encore plus générique :

$$\forall s \in S, V_{k+1}(s) = V_k(s) + \alpha[R_{k+1}(s) - V_k(s)] \tag{2.3}$$

qui, pour α bien choisi, vérifie

$$\lim_{k \to \infty} V_k(s) = V^{\pi}(s) \tag{2.4}$$

Nous retrouverons cette méthode d'estimation incrémentale dans les méthodes de différence temporelle.

2.3.2. *Les méthodes de Monte Carlo*

Pour résoudre un problème de planification en utilisant la programmation dynamique dans le cas où l'on ne connaît pas *a priori* les fonctions de transition $p()$ et de récompense $r()$, l'approche indirecte de type *maximum de vraisemblance* qui consiste à estimer les paramètre $p()$ et $r()$ puis à calculer V en résolvant l'équation $V = L_{\pi}V$ (avec ici $\gamma = 1$) est généralement trop coûteuse, en temps et en espace.

On lui préfère classiquement l'approche dite de *Monte Carlo*, qui revient à simuler un grand nombre de trajectoires issues de chaque état s de S, et à estimer $V(s)$ en moyennant les coûts observés sur chacune de ces trajectoires. À chaque expérience réalisée, l'agent mémorise les transitions qu'il a effectuées et les récompenses qu'il a reçues. Il met alors à jour une estimation de la valeur des états parcourus en associant à chacun d'eux la part de la récompense reçue qui lui revient. Au fil de ces expériences, la valeur estimée associée à chaque état converge alors vers la valeur exacte de l'état pour la politique qu'il suit.

L'apport principal des méthodes de Monte Carlo réside donc dans la technique qui permet d'estimer la valeur d'un état sur la base de la réception de plusieurs valeurs

successives de récompense cumulée associées à cet état lors de trajectoires distinctes. On s'appuie alors sur la méthode d'estimation présentée à la section 2.3.1.

Soit ainsi $(s_0, s_1, \ldots, s_N)$ une trajectoire générée en suivant la probabilité de transition inconnue $p()$, et $(r_0, r_1, \ldots, r_{N-1})$ les récompenses observées au cours de cette trajectoire (s_N est l'état terminal T de récompense nulle).

Le principe de la méthode de Monte Carlo est de mettre à jour les N valeurs $V(s_k)$, $k = 0, \ldots, N-1$, selon :

$$V(s_k) \leftarrow V(s_k) + \alpha(s_k)(r_k + r_{k+1} + \cdots + r_{N-1} - V(s_k)) \tag{2.5}$$

avec les taux d'apprentissage $\alpha(s_k)$ tendant vers 0 au cours des itérations. La convergence presque sûre de cet algorithme vers la fonction V est assurée sous des hypothèses générales [BER 96].

Cette méthode est qualifiée d'*every-visit* car la valeur d'un état peut être mise à jour plusieurs fois le long d'une même trajectoire. Les termes d'erreur associés à chacune de ces mises à jours ne sont alors pas indépendants, entraînant un biais non nul dans l'estimation de la fonction V sur la base d'un nombre fini de trajectoires [BER 96, page 190]. Une solution simple à ce problème de biais consiste alors à ne mettre à jour la valeur $V(s)$ d'un état que lors de sa première rencontre le long de la trajectoire observée. Cela définit la méthode dite de *first-visit*, qui conduit à un estimateur non-biaisé de la fonction V. Expérimentalement, l'erreur quadratique moyenne de la méthode *first-visit* tend à être inférieure à celle de la méthode *every-visit* [SIN 96].

Les méthodes de Monte Carlo permettent donc d'estimer la fonction de valeur d'une politique π en mettant à jour certaines de ses composantes à la fin de chaque trajectoire observée. Ces méthodes exigent pour fonctionner qu'un grand nombre de contraintes soient remplies. En particulier, il est indispensable que l'apprentissage soit décomposé en une succession d'épisodes de longueur finie, faute de quoi la mise à jour de l'estimation de la valeur des états ne peut pas avoir lieu. Le fait qu'il faille attendre la fin d'une expérience pour apprendre quoi que ce soit justifie l'affirmation selon laquelle ces méthodes ne sont pas incrémentales.

Il est toutefois possible d'améliorer ces algorithmes en autorisant la mise à jour de la fonction de valeur non plus à la fin de chaque trajectoire, mais à la suite de chaque transition du système. Nous développons ici cette réécriture, dont le principe est à la base des méthodes de différence temporelle.

La règle de mise à jour (2.5) de la fonction V peut être réécrite de la manière suivante (le terme γ vaut 1, il est introduit dans les équations pour faire ressortir les

relations avec l'erreur de différence temporelle utilisée dans les méthodes de différence temporelle ; par ailleurs, nous utilisons la propriété $V(s_N) = V(T) = 0$) :

$$V(s_k) \leftarrow V(s_k) + \alpha(s_k) \quad \Big((r_k + \gamma V(s_{k+1}) - V(s_k)) + (r_{k+1} + \gamma V(s_{k+2}) - V(s_{k+1})) + \cdots + (r_{N-1} + \gamma V(s_N) - V(s_{N-1})) \Big)$$

soit encore

$$V(s_k) \leftarrow V(s_k) + \alpha(s_k)(\delta_k + \delta_{k+1} + \cdots + \delta_{N-1}) \tag{2.6}$$

en définissant la différence temporelle δ_k par

$$\delta_k = r_k + \gamma V(s_{k+1}) - V(s_k), \quad k = 0, \ldots, N-1$$

Le terme δ_k[8] est appelé « erreur de différence temporelle [9] » [SUT 88].

Cette erreur δ_k peut être interprétée en chaque état comme une mesure de la différence entre l'estimation courante $V(s_k)$ et l'estimation corrigée à un coup $r_k + V(s_{k+1})$. Son calcul est possible dès que la transition (s_k, s_{k+1}, r_k) a été observée, et cela conduit donc à une version « *on-line* » de la règle de mise à jour (2.6) où il n'est plus nécessaire d'attendre la fin de la trajectoire pour commencer à modifier les valeurs de V :

$$V(s_l) \leftarrow V(s_l) + \alpha(s_l)\delta_k, \quad l = 0, \ldots, k \tag{2.7}$$

dès que la transition (s_k, s_{k+1}, r_k) est simulée et l'erreur δ_k calculée. Selon qu'une trajectoire peut parcourir plusieurs fois le même état ou non, cette version « *on-line* » peut légèrement différer de l'algorithme original (2.6). Toutefois sa convergence presque sûre vers V reste valide. Là encore, une approche de type *first-visit* semble préférable en pratique [SIN 96].

2.4. Les méthodes de différence temporelle

Nous allons nous tourner à présent vers les méthodes de différence temporelle, qui combinent l'incrémentalité de la programmation dynamique avec le recours à l'expérience des méthodes de Monte Carlo.

---

8. On trouve parfois $\delta_k = r_{k+1} + \gamma V(s_{k+1}) - V(s_k), \quad k = 0, \ldots, N-1$, si l'on note r_{k+1} plutôt que r_k la récompense reçue lors du passage de l'état s_k à l'état s_{k+1}.

9. *temporal difference error*

2.4.1. *L'algorithme* TD(0)

Nous avons vu au chapitre précédent qu'il existait plusieurs critères possibles pour représenter la performance que l'agent doit maximiser sur le long terme. Historiquement, le critère qui a donné lieu aux développements les plus importants est le critère dit « γ-*pondéré* ». Tous les algorithmes que nous allons présenter dans cette section, qui sont les algorithmes les plus classiques de l'apprentissage par renforcement, s'appliquent dans le cadre de ce critère.

L'algorithme élémentaire d'apprentissage par renforcement, dit algorithme de « différence temporelle » s'appelle TD [10]. Nous le notons ici TD(0) pour des raisons qui apparaîtront quand nous présenterons les traces d'éligibilité. Cet algorithme repose sur une comparaison entre la récompense que l'on reçoit effectivement et la récompense que l'on s'attend à recevoir en fonction des estimations construites précédemment.

Si les estimations des fonctions de valeur aux états s_t et s_{t+1}, notées $V(s_t)$ et $V(s_{t+1})$, étaient exactes, on aurait :

$$V(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ... \quad (2.8)$$

$$V(s_{t+1}) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \quad (2.9)$$

Donc on aurait :

$$V(s_t) = r_t + \gamma V(s_{t+1}) \quad (2.10)$$

On voit que l'erreur de différence temporelle δ_k mesure l'erreur entre les valeurs effectives des estimations $V(s)$ et les valeurs qu'elles devraient avoir.

La méthode de différence temporelle consiste à corriger peu à peu cette erreur en modifiant la valeur de $V(s_t)$ selon une équation de type Widrow-Hoff, que l'on utilise dans le domaine des réseaux de neurones :

$$V(s_t) \leftarrow V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)] \quad (2.11)$$

Cette équation de mise à jour permet de comprendre immédiatement comment les algorithmes de différence temporelle combinent les propriétés de la programmation dynamique avec celles des méthodes de Monte Carlo. En effet, elle fait apparaître les deux caractéristiques suivantes :

10. *Temporal Difference*

– comme dans les algorithmes de programmation dynamique, la valeur estimée de $V(s_t)$ est mise à jour en fonction de la valeur estimée de $V(s_{t+1})$. Il y a donc propagation de la valeur estimée à l'état courant à partir des valeurs estimées des états successeurs ;

– comme dans les méthodes de Monte Carlo, chacune de ces valeurs résulte d'une estimation locale des récompenses immédiates qui repose sur l'expérience accumulée par l'agent au fil de ses interactions avec son environnement.

On voit donc que les méthodes de différence temporelle et, en particulier, TD(0), reposent sur deux processus de convergence couplés, le premier estimant de plus en plus précisément la récompense immédiate reçue dans chacun des états et le second approchant de mieux en mieux la fonction de valeur résultant de ces estimations en les propageant de proche en proche.

Dans le cas de TD(0), les mises à jour se font localement à chaque fois que l'agent réalise une transition dans son environnement, à partir d'une information se limitant à son état courant s_t, l'état successeur s_{t+1} et la récompense r_t reçue suite à cette transition. Une preuve de convergence de l'algorithme a été proposée par Dayan et Sejnowski [DAY 94].

Par contre, il faut noter que, comme TD(0) estime la fonction de valeur de chacun des états d'un problème, faute d'un modèle des transitions entre les états, l'agent est incapable d'en déduire quelle politique suivre, car il ne peut réaliser un pas de regard en avant pour déterminer quelle est l'action qui lui permettra de rejoindre l'état suivant de plus grande valeur. Ce point explique que l'on préfère avoir recours aux algorithmes qui travaillent sur une fonction de valeur associée aux couples (état, action) plutôt qu'à l'état seul. Ce sont ces algorithmes que nous allons présenter dans ce qui suit. Nous verrons ensuite l'approche alternative proposée par les architectures acteur-critique, consistant à travailler directement sur une politique que l'on cherche à améliorer itérativement au fil de l'expérience.

2.4.2. *L'algorithme* SARSA

Comme nous venons de l'expliquer, la forme de l'équation de Bellman $V = LV$ n'est pas satisfaisante pour en dériver directement un algorithme adaptatif de résolution. Pour cela, Watkins [WAT 89] a introduit la fonction de valeur Q, dont la donnée est équivalente à celle de V lorsqu'on connaît la fonction de transition p.

Définition 1 (Fonction de valeur Q)
À une politique π fixée de fonction de valeur V^π, on associe la nouvelle fonction

$$\forall s \in S, a \in A \quad Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^\pi(s').$$

L'interprétation de la valeur $Q^\pi(s,a)$ est la suivante : c'est la valeur espérée du critère pour le processus partant de s, exécutant l'action a, puis suivant la politique π par la suite. Il est clair que $V^\pi(x) = Q^\pi(x, \pi(x))$, et l'équation de Bellman vérifiée par la fonction Q^* devient :

$$\forall s \in S, a \in A \quad Q^*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_b Q^*(s',b)$$

On a alors

$$\begin{aligned} \forall s \in S \quad V^*(s) &= \max_a Q^*(s,a) \\ \pi^*(s) &= \operatorname*{argmax}_a Q^*(s,a) \end{aligned}$$

L'algorithme SARSA est similaire à l'algorithme TD(0) à ceci près qu'il travaille sur les valeurs des couples (s,a) plutôt que sur la valeur des états. Son équation de mise à jour est identique à celle de TD(0) en remplaçant la fonction de valeur par la fonction de valeur d'action :

$$Q(s_n, a_n) \leftarrow Q(s_n, a_n) + \alpha[r_n + \gamma Q(s_{n+1}, a_{n+1}) - Q(s_n, a_n)] \tag{2.12}$$

L'information nécessaire pour réaliser une telle mise à jour alors que l'agent réalise une transition est le quintuplet $(s_n, a_n, r_n, s_{n+1}, a_{n+1})$, d'où découle le nom de l'algorithme.

Effectuer ces mises à jour implique que l'agent détermine avec un pas de regard en avant quelle est l'action a_{n+1} qu'il réalisera lors du pas de temps suivant, lorsque l'action a_n dans l'état s_n l'aura conduit dans l'état s_{n+1}.

Il résulte de cette implication une dépendance étroite entre la question de l'apprentissage et la question de la détermination de la politique optimale. Dans un tel cadre, il n'existe qu'une seule politique, qui doit prendre en compte à la fois les préoccupations d'exploration et d'exploitation, et l'agent est contraint de réaliser cet apprentissage uniquement sur la base de la politique qu'il suit effectivement. On dit d'un algorithme tel que SARSA qu'il est « *on-policy* ». La dépendance que cela induit entre l'exploration et l'apprentissage complique considérablement la mise au point de preuves de convergences pour ces algorithmes, ce qui explique que de telles preuves de convergence soient apparues beaucoup plus tard [SIN 00] que pour les algorithmes dits « *off-policy* » tels que Q-learning, que nous allons voir à présent.

2.4.3. *L'algorithme Q-learning*

L'algorithme Q-learning se présente comme une simplification de l'algorithme SARSA par le fait qu'il n'est plus nécessaire pour l'appliquer de déterminer un pas de temps à l'avance quelle sera l'action réalisée au pas de temps suivant.

Son équation de mise à jour est la suivante :

$$Q(s_n, a_n) \leftarrow Q(s_n, a_n) + \alpha[r_n + \gamma \max_a Q(s_{n+1}, a) - Q(s_n, a_n)] \qquad (2.13)$$

La différence essentielle entre SARSA et Q-learning se situe au niveau de la définition du terme d'erreur. Le terme $Q(s_{n+1}, a_{n+1})$ apparaissant dans l'équation (2.12) a été remplacé par le terme $max_a Q(s_{n+1}, a)$ dans l'équation (2.13). Cela pourrait sembler équivalent si la politique suivie était gloutonne (on aurait alors $a_{n+1} = \arg\max_a Q(s_{n+1}, a)$). Toutefois, compte tenu de la nécessité de réaliser un compromis entre exploration et exploitation, ce n'est généralement pas le cas. Il apparaît donc que l'algorithme SARSA effectue les mises à jour en fonction des actions choisies effectivement alors que l'algorithme Q-learning effectue les mises à jour en fonction des actions optimales mêmes si ce ne sont pas ces actions optimales que l'agent réalise, ce qui est plus simple.

Cette simplicité a fait la réputation de l'algorithme Q-learning. Il s'agit sans doute de l'algorithme d'apprentissage par renforcement le plus connu et le plus utilisé en raison des preuves formelles de convergence qui ont accompagné sa publication [WAT 92].

Algorithme 2.1 : Le Q-learning

```
/* α_n est un taux d'apprentissage                                  */
Initialiser(Q_0)
pour n ← 0 jusqu'à N_tot − 1 faire
    s_n ← ChoixEtat
    a_n ← ChoixAction
    (s'_n, r_n) ← Simuler(s_n, a_n)
    { mise à jour de Q_n :}
    début
        Q_{n+1} ← Q_n
        δ_n ← r_n + γ max_b Q_n(s'_n, b) − Q_n(s_n, a_n)
        Q_{n+1}(s_n, a_n) ← Q_n(s_n, a_n) + α_n(s_n, a_n)δ_n
    fin
retourner Q_{N_tot}
```

Le principe de l'algorithme Q-learning, défini formellement par l'algorithme 2.1, est de mettre à jour itérativement, à la suite de chaque transition (s_n, a_n, s_{n+1}, r_n),

la fonction de valeur courante Q_n pour le couple (s_n, a_n), où s_n représente l'état courant, a_n l'action sélectionnée et réalisée, s'_n l'état résultant et r_n la récompense immédiate. Cette mise à jour se fait sur la base de l'observation des transitions instantanées et de leur récompense associée.

Dans cet algorithme, N_{tot} est un paramètre initial fixant le nombre d'itérations. Le taux d'apprentissage $\alpha_n(s, a)$ est propre à chaque paire état-action, et décroît vers 0 à chaque passage. La fonction `Simuler` retourne un nouvel état et la récompense associée selon la dynamique du système. Le choix de l'état courant et de l'action à exécuter est effectué par les fonctions `ChoixEtat` et `ChoixAction` et sera discuté plus loin. La fonction `Initialiser` revient la plupart du temps à initialiser les composantes de Q_0 à 0, mais il existe des initialisations plus efficaces.

Il est immédiat d'observer que l'algorithme Q-learning est une formulation stochastique de l'algorithme de *value iteration* vu au chapitre précédent pour les MDP. En effet, ce dernier peut s'exprimer directement en terme de fonction de valeur d'action :

$$
\begin{aligned}
& V_{n+1}(s) = \max_{a \in A} \overbrace{\left\{ r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V_n(s') \right\}}^{Q_n(s,a)} \\
\Rightarrow \quad & Q_{n+1}(s,a) = r(s,a) + \gamma \textstyle\sum_{s' \in S} p(s'|s,a) V_{n+1}(s') \\
\Rightarrow \quad & Q_{n+1}(s,a) = r(s,a) + \gamma \textstyle\sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q_n(s',a')
\end{aligned}
$$

Le Q-learning est alors obtenu en remplaçant le terme $r(s, a) + \sum_{s'} p(s'|s, a) \max_{a' \in A} Q_n(s', a')$ par son estimateur sans biais le plus simple construit à partir de la transitition courante $r_n + \max_{a'} Q_n(s'_n, a')$.

La convergence de cet algorithme est établie [WAT 89, JAA 94a] (la fonction Q_n converge presque sûrement vers Q^*) sous les hypothèses suivantes :

– finitude de S et A,

– chaque paire (s, a) est visitée un nombre infini de fois,

– $\sum_n \alpha_n(s, a) = \infty$ et $\sum_n \alpha_n^2(s, a) < \infty$,

– $\gamma < 1$ ou si $\gamma = 1$

– pour toute politique il existe un état absorbant de récompense nulle.

Rappelons que cette convergence presque sûre signifie que $\forall s, a$ la suite $Q_n(s, a)$ converge vers $Q^*(s, a)$ avec une probabilité égale à 1. En pratique, la suite $\alpha_n(s, a)$ est souvent définie comme $\alpha_n(s, a) = \frac{1}{n_{sa}}$.

2.4.4. *Les algorithmes TD(λ), Sarsa(λ) et Q(λ)*

Figure 2.1. *Q-learning : premier et deuxième essai. On observe que, toutes les valeurs étant initialement nulles, la propagation de valeurs non nulles ne se fait qu'une fois que l'agent a trouvé une première fois la source de récompense et ne progresse que d'un pas à chaque essai de l'agent.*

Les algorithmes TD(0), SARSA et Q-learning présentent le défaut de ne mettre à jour qu'une valeur par pas de temps, à savoir la valeur de l'état que l'agent est en train de visiter. Comme il apparaît sur la figure 2.1, cette procédure de mise à jour est particulièrement lente. En effet, pour un agent ne disposant d'aucune information a priori sur la structure de la fonction de valeur, il faut au moins n expériences successives pour que la récompense immédiate reçue dans un état donné soit propagée jusqu'à un état distant du premier de n transitions. En attendant le résultat de cette propagation, tant que toutes les valeurs sont identiquement nulles, le comportement de l'agent est une marche aléatoire.

Une façon d'améliorer cet état de fait consiste à doter l'algorithme d'une mémoire des transitions effectuées au cours d'une expérience afin d'effectuer toutes les propagations possibles à la fin de cette expérience. Cette mémoire des transitions effectuées précédemment est appelée une *trace d'éligibilité*. Ainsi, Sutton et Barto [SUT 98] ont proposé une classe d'algorithmes appelés « TD(λ) » qui généralisent l'algorithme TD(0) au cas où l'agent dispose d'une mémoire des transitions. Plus tard, les algorithmes SARSA et Q-learning ont été généralisés en SARSA (λ) et Q(λ), le second l'ayant été de deux façons différentes par deux auteurs différents [WAT 92, PEN 96].

Un premier procédé naïf pour accélérer l'apprentissage consiste à stocker directement une liste des couples (état, action) parcourus par l'agent puis, à chaque fois que l'agent reçoit une récompense, propager celle-ci en parcourant la mémoire des transitions en marche arrière depuis la récompense reçue. Avec un tel procédé, plus la mémoire des transitions est longue, plus une récompense reçue est propagée efficacement. Il apparaît donc un compromis entre la quantité de mémoire mobilisée pour apprendre et la vitesse d'apprentissage. Mais une telle méthode ne fonctionne pas sur un horizon infini.

La méthode mise en œuvre dans TD(λ), SARSA (λ) et Q(λ) est plus sophistiquée et fonctionne pour des horizons infinis. Nous discuterons à la section 2.5.1, page 78, une autre solution qui permet d'effectuer plusieurs mises à jour à chaque pas de temps, dans le cadre de l'apprentissage par renforcement indirect. Nous commençons par examiner de plus près les algorithmes TD(λ), SARSA (λ) et $Q(\lambda)$.

2.4.5. *Les architectures acteur-critique*

Historiquement, les architectures acteur-critique ont été les premières architectures informatiques imaginées par des chercheurs pour modéliser l'apprentissage par renforcement [WIT 77, BAR 83].

Nous avons dit que, dans TD(0), on met à jour la fonction de valeur en se fondant sur l'erreur de différence temporelle $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ mais, si on ne dispose pas d'un modèle de la fonction de transition, on ne sait pas comment exploiter la fonction de valeur pour choisir une action. La solution proposée par les architectures acteur-critique consiste à tenir à jour en parallèle une structure représentant la fonction de valeur, appelée « le critique », et une structure représentant la politique, appelée « l'acteur ».

Le critique est nécessaire pour calculer la valeur de δ_t, qui dépend de la fonction de valeur. Mais le terme δ_t est aussi utilisé pour mettre à jour la politique. Si ce terme est positif, l'action réalisée mérite d'être renforcée. S'il est négatif, au contraire, il faut diminuer la tendance de l'acteur à réaliser cette action.

Cette spécification est très générale, elle impose très peu de contraintes sur les formalismes de représentation respectifs de l'acteur et du critique. La seule contrainte forte est que l'acteur soit capable de prendre en compte le signal d'erreur δ_t pour modifier sa propension à réaliser telle ou telle action dans un contexte donné. Quant au critique, tout formalisme capable de fournir une approximation de la fonction de valeur fait l'affaire.

Cependant, un grand nombre d'architectures acteur-critique s'appuient sur des réseaux de neurones formels, en raison de la relative plausibilité biologique de cette architecture. Le terme δ_t vient alors modifier le poids des connexions qui gouvernent la propension d'un réseau à réaliser telle ou telle action.

2.4.6. *Différences temporelles avec traces d'éligibilité : TD(λ)*

L'originalité de TD(λ) est de proposer un compromis entre les deux équations (2.6) et (2.7) présentées dans le cadre de l'algorithme de Monte Carlo itératif. Soit donc

$\lambda \in [0,1]$ un paramètre de pondération. Avec les mêmes notations que précédemment, l'algorithme TD(λ) défini par Sutton [SUT 88] est le suivant :

$$V(s_k) \leftarrow V(s_k) + \alpha(s_k) \sum_{m=k}^{m=N-1} \lambda^{m-k} \delta_m, \quad k = 0, \ldots, N-1 \tag{2.14}$$

On peut essayer de mieux comprendre le rôle du coefficient λ en réécrivant l'équation (2.14) sous la forme

$$V(s_k) \leftarrow V(s_k) + \alpha(s_k)(z_k^\lambda - V(s_k))$$

On a alors

$$\begin{aligned}
z_k^\lambda &= V(s_k) + \sum_{m=k}^{m=N-1} \lambda^{m-k} \delta_m \\
&= V(s_k) + \delta_k + \lambda \sum_{m=k+1}^{m=N-1} \lambda^{m-k-1} \delta_m \\
&= V(s_k) + \delta_k + \lambda(z_{k+1}^\lambda - V(s_{k+1})) \\
&= V(s_k) + r_k + V(s_{k+1}) - V(s_k) + \lambda(z_{k+1}^\lambda - V(s_{k+1})) \\
&= r_k + (\lambda z_{k+1}^\lambda + (1-\lambda) V(s_{k+1}))
\end{aligned}$$

Dans le cas où $\lambda = 0$, il est clair que cela revient à ne considérer qu'un horizon unitaire, comme dans le cadre de la programmation dynamique. On retrouve donc TD(0).

Si $\lambda = 1$, l'équation (2.14) se réécrit

$$V(s_k) \leftarrow V(s_k) + \alpha(s_k) \sum_{m=k}^{m=N-1} \delta_m, \quad k = 0, \ldots, N-1,$$

ce qui est exactement l'équation (2.5) de la méthode de Monte Carlo.

Pour tout λ, les deux approches de type *first-visit* ou *every-visit* peuvent être considérées. De même, une version *on-line* de l'algorithme d'apprentissage TD(λ) décrit par l'équation (2.14) est possible :

$$V(s_l) \leftarrow V(s_l) + \alpha(s_l) \lambda^{k-l} \delta_k, \quad l = 0, \ldots, k \tag{2.15}$$

dès que la transition (s_k, s_{k+1}, r_k) est simulée et l'erreur δ_k calculée.

L'application du TD(λ) pour l'évaluation d'une politique π selon le critère γ-pondéré entraîne certaines modifications des algorithmes standards (2.14) ou (2.15), qu'il est nécessaire de citer ici.

Un calcul en tout point semblable au cas $\gamma = 1$ conduit à une règle du type :

$$V(s_k) \leftarrow V(s_k) + \alpha(s_k) \sum_{m=k}^{m=\infty} (\gamma\lambda)^{m-k} \delta_m \tag{2.16}$$

Il est alors clair que l'absence potentielle d'états finaux absorbants rend inadéquate un algorithme de type *off-line* ne mettant à jour la fonction de valeur V qu'à la fin de la trajectoire, car celle-ci peut être de taille infinie. On définit donc une version *on-line* de (2.16), qui prend la forme suivante :

$$V(s) \leftarrow V(s) + \alpha(s) z_n(s) \delta_n, \quad \forall s \in S, \tag{2.17}$$

dès que la $n$ième transition (s_n, s_{n+1}, r_n) a été simulée et l'erreur δ_n calculée. Le terme $z_n(s)$, dénommé trace d'éligibilité [11] se définit ainsi dans la version la plus proche de l'algorithme TD(λ) original :

Définition 2 (Trace d'éligibilité accumulative)

$$\begin{array}{rcll} z_0(s) & = 0, & \forall s \in S & \\ z_n(s) & = \left\{ \begin{array}{ll} \gamma\lambda z_{n-1}(s) & \textit{si } s \neq s_n \\ \gamma\lambda z_{n-1}(s) + 1 & \textit{si } s = s_n \end{array} \right. & & \end{array}$$

Ce coefficient d'éligibilité augmente donc sa valeur à chaque nouveau passage dans l'état associé, puis décroît exponentiellement au cours des itérations suivantes, jusqu'à un nouveau passage dans cet état (voir figure 2.2).

Dans certains cas, une définition légèrement différente de la trace $z_n(s)$ semble conduire à une convergence plus rapide de la fonction de valeur V :

Définition 3 (Trace d'éligibilité avec réinitialisation)

$$\begin{array}{rcll} z_0(s) & = 0, & \forall s \in S & \\ z_n(s) & = \left\{ \begin{array}{ll} \gamma\lambda z_{n-1}(s) & \textit{si } s \neq s_n \\ 1 & \textit{si } s = s_n \end{array} \right. & & \end{array}$$

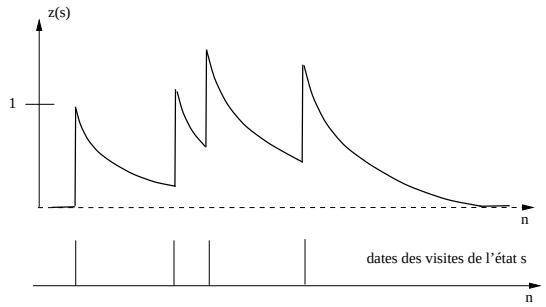


Figure 2.2. *Trace d'éligibilité cumulative : à chaque visite, on ajoute 1 à la valeur précédente, si bien que la valeur de la trace peut dépasser 1.*

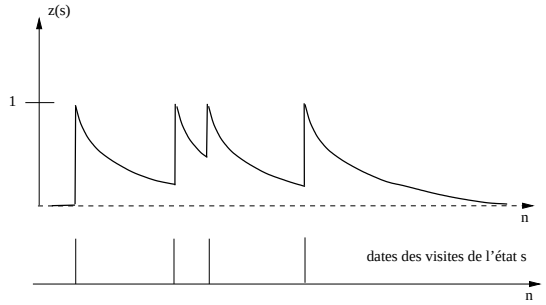


Figure 2.3. *Trace d'éligibilité avec réinitialisation : on remet la valeur à 1 à chaque visite.*

La valeur de la trace est donc saturée à 1, comme le montre la figure 2.3.

La convergence presque sûre de l'algorithme TD(λ) a été montrée pour toute valeur de λ, en *on-line* ou *off-line*, sous les hypothèses classiques de visite en nombre infini de chaque état $s \in S$, et décroissance des α vers 0 à chaque itération n, telle que $\sum_n \alpha_n(s) = \infty$ et $\sum_n \alpha_n^2(s) < \infty$ [JAA 94a, BER 96].

Il est à noter que l'effet du λ est encore mal compris et sa détermination optimale pour un problème donné reste très empirique.

Une implémentation directe de TD(λ) basée sur la trace d'éligibilité n'est bien sûr pas efficace dès que la taille de l'espace d'état S devient trop grande. Une première solution approchée [SUT 98] consiste à forcer à 0 la valeur de toutes les traces $z_n(s) < \varepsilon$, et donc à ne maintenir que les traces des états récemment visités (plus

11. *eligibility trace*, ou encore *activity*.

précisément, on cesse de maintenir un état dont la dernière visite remonte à plus de $\frac{\log(\varepsilon)}{\log(\gamma\lambda)}$ transitions.

Une autre méthode approchée [CIC 95] connue sous le nom de *truncated temporal differences*, ou TTD(λ), revient à gérer un horizon glissant de taille m mémorisant les derniers états visités et à mettre à jour sur cette base à chaque itération n la valeur de l'état visité à l'itération $(n - m)$.

2.4.7. *De TD(λ) à* SARSA *(λ)*

TD(λ) peut être appliqué au problème de l'apprentissage par renforcement pour apprendre une politique optimale. Pour cela, une première approche consiste à coupler TD(λ) à un algorithme gérant l'évolution d'une suite de politiques π_n. En effet, contrairement au Q-learning qui voit la suite Q_n converger vers Q^* sans nécessiter la présence en parallèle d'une suite de politiques π_n, l'algorithme TD(λ) ne sait qu'apprendre la fonction de valeur d'une politique fixée. Dans Q-learning, une telle suite de politiques existe à travers π_{Q_n}, mais l'intérêt du Q-learning est justement que cette suite n'est qu'implicite. On retrouve donc ici un type d'opposition rencontrée au chapitre précédent en programmation dynamique entre *value iteration* et *policy iteration*.

Toutefois, il s'avère que l'algorithme Q-learning intègre directement l'idée maîtresse de TD(λ) de considérer une erreur de différence temporelle. Si l'on reprend la règle de mise à jour du Q-learning

$$Q_{n+1}(s_n, a_n) = Q_n(s_n, a_n) + \alpha_n\{r_n + \gamma V_n(s'_n) - Q_n(s_n, a_n)\}$$

pour la transition observée (s_n, a_n, s'_n, r_n), et dans le cas où l'action a_n exécutée dans l'état s_n est l'action optimale pour Q_n, c'est-à-dire $a_n = \pi_{Q_n}(s_n) = \mathrm{argmax}_b\, Q_n(s_n, b)$, on constate que le terme d'erreur employé est égal à

$$r_n + \gamma V_n(s'_n) - V_n(s_n)$$

qui est exactement celui de TD(0). Cela peut alors se généraliser à $\lambda > 0$, au travers d'un couplage entre les méthodes TD(λ) et Q-learning.

L'algorithme SARSA (λ) [RUM 94] en est une première illustration. Cet algorithme 2.2 reprend directement l'équation (2.17) en l'adaptant à une représentation par fonction de valeur d'action.

La trace d'éligibilité $z_n(s, a)$ est étendue aux couples état-action et l'exploration de l'espace d'états est guidée par la dynamique (sauf lors de la rencontre avec un état terminal).

Algorithme 2.2 : SARSA(λ)

```
/* α_n est un taux d'apprentissage                                  */
Initialiser(Q_0)
z_0 ← 0
s_0 ← ChoixEtat
a_0 ← ChoixAction
pour n ← 0 jusqu'à N_tot − 1 faire
    (s'_n, r_n) ← Simuler(s_n, a_n)
    a'_n ← ChoixAction
    { mise à jour de Q_n et z_n :}
    début
        δ_n ← r_n + γQ_n(s'_n, a'_n) − Q_n(s_n, a_n)
        z_n(s_n, a_n) ← z_n(s_n, a_n) + 1
        pour s ∈ S, a ∈ A faire
            Q_{n+1}(s, a) ← Q_n(s, a) + α_n(s, a)z_n(s, a)δ_n
            z_{n+1}(s, a) ← γλz_n(s, a)
    fin
    si s'_n non absorbant alors s_{n+1} ← s'_n et a_{n+1} ← a'_n
    sinon
        s_{n+1} ← ChoixEtat
        a_{n+1} ← ChoixAction
retourner Q_{N_tot}
```

2.4.7.1. $Q(\lambda)$

La prise en compte des cas où l'action optimale $\pi_{Q_n}(s'_n)$ n'a pas été sélectionnée conduit aux algorithmes $Q(\lambda)$ proposés par Watkins (voir [SUT 98]) et Peng [PEN 94]. La caractéristique du $Q(\lambda)$ de Watkins est de ne considérer un $\lambda > 0$ que le long des segments de trajectoires où la politique courante π_{Q_n} a été suivie. Les deux modifications relativement à SARSA(λ) concernent donc les règles de mise à jour de Q_n et de z_n, comme cela apparaît dans l'algorithme 2.3.

L'inconvénient de cette approche est que, pour des politiques d'apprentissage très exploratrices, les traces z_n sont très fréquemment remises à 0 et le comportement de Q(λ) est alors assez proche du Q-learning original. Le $Q(\lambda)$ de Peng est une réponse à ce problème. Il est aussi possible d'imaginer une application directe de TD(λ) au Q-learning en ne remettant pas la trace z_n à 0 lors du choix d'une action non-optimale. Il existe peu de résultats expérimentaux présentant cette approche (voir toutefois [NDI 99]) ni de comparaisons entre TD(λ), SARSA(λ) et Q(λ) autorisant de tirer des conclusions définitives sur le sujet. La seule véritable certitude issue de nombreuses applications est que la prise en compte des traces d'éligibilité avec $\lambda > 0$ accélère la convergence de l'apprentissage en terme de nombre d'itérations. L'analyse

Algorithme 2.3 : Le Q(λ)

```
/* α_n est un taux d'apprentissage                                  */
Initialiser(Q_0)
z_0 ← 0
s_0 ← ChoixEtat
a_0 ← ChoixAction
pour n ← 0 jusqu'à N_tot − 1 faire
    (s'_n, r_n) ← Simuler(s_n, a_n)
    a'_n ← ChoixAction
    { mise à jour de Q_n et z_n :}
    début
        δ_n ← r_n + γ max_b Q_n(s'_n, b) − Q_n(s_n, a_n)
        z_n(s_n, a_n) ← z_n(s_n, a_n) + 1
        pour s ∈ S, a ∈ A faire
            Q_{n+1}(s, a) ← Q_n(s, a) + α_n(s, a) z_n(s, a) δ_n
            z_{n+1}(s, a) ← { 0              si a'_n ≠ π_{Q_n}(s'_n)
                            { γλ z_n(s, a)   si a'_n = π_{Q_n}(s'_n)
    fin
    si s'_n non absorbant alors s_{n+1} ← s'_n et a_{n+1} ← a'_n
    sinon
        s_{n+1} ← ChoixEtat
        a_{n+1} ← ChoixAction
retourner Q_{N_tot}
```

en terme de temps de calcul est plus complexe, car les algorithmes prenant en compte une trace nécessitent beaucoup plus de calculs à chaque itération.

2.4.8. *L'algorithme R-learning*

Tous les algorithmes que nous avons présentés jusqu'à présent s'appliquaient dans le cas du critère γ-pondéré. L'algorithme R-learning, proposé par Schwartz [SCH 93], est l'adaptation au critère moyen de l'algorithme Q-learning et l'on y retrouve tous les principes évoqués précédemment.

L'objectif de cet algorithme est de construire une politique dont la récompense moyenne ρ_π est la plus proche possible de la récompense moyenne maximale ρ^* d'une politique optimale π^*. Pour cela, le R-learning maintient deux suites entrecroisées ρ_n et R_n. Notons ici que la suite ρ_n n'est mise à jour que lorsque l'action qui vient d'être exécutée était la *greedy-action* maximisant R_n dans l'état courant s_n. La suite réelle des ρ_n est une estimation du critère à optimiser. Comme Q_n dans le Q-learning, R_n représente une forme particulière de la fonction de valeur relative U d'une politique :

Définition 4 (Fonction de valeur R)
À une politique π fixée de fonction de valeur U^π et de gain moyen ρ_π, on associe la nouvelle fonction

$$\forall s \in S, a \in A \quad R^\pi(s,a) = r(s,a) - \rho_\pi + \sum_{s'} p(s'|s,a) U^\pi(s').$$

On a donc là encore $U^\pi(x) = R^\pi(x, \pi(x))$ et l'équation de Bellman vérifiée par ρ^* et R^* devient :

$$\forall s \in S, a \in A \quad R^*(s,a) = r(s,a) - \rho^* + \sum_{s'} p(s'|s,a) \max_b R^*(s',b) \qquad (2.18)$$

avec l'assurance que la politique $\pi_{R^*}(s) = \operatorname{argmax}_a R^*(s,a)$ a pour gain moyen le gain optimal ρ^*.

Comme pour Q-learning, l'algorithme R-learning est une version stochastique de la méthode d'itération sur les valeurs pour l'équation (2.18).

Algorithme 2.4 : Le R-learning

`/* ` α_n `et` β_n `sont des taux d'apprentissage */`
`Initialiser(`R_0,ρ_0`)`
pour $n \leftarrow 0$ **jusqu'à** $N_{tot} - 1$ **faire**
  $s_n \leftarrow$ `ChoixEtat`
  $a_n \leftarrow$ `ChoixAction`
  $(s'_n, r_n) \leftarrow$ `Simuler(`s_n, a_n`)`
  { mise à jour de R_n et ρ_n :}
  début
    $R_{n+1} \leftarrow R_n$
    $\delta_n \leftarrow r_n - \rho_n + \max_b R_n(s'_n, b) - R_n(s_n, a_n)$
    $R_{n+1}(s_n, a_n) \leftarrow R_n(s_n, a_n) + \alpha_n(s_n, a_n)\delta_n$
    $\rho_{n+1} \leftarrow \begin{cases} \rho_n & \text{si } a_n \neq \pi_{R_n}(s_n) \\ \rho_n + \beta_n \delta_n & \text{si } a_n = \pi_{R_n}(s_n) \end{cases}$
  fin
retourner $R_{N_{tot}}$, $\rho_{N_{tot}}$

Bien qu'il n'existe pas de preuve formelle de convergence du R-learning vers une politique gain-optimale, de nombreuses expérimentations montrent que ρ_n approche efficacement ρ^*, avec des taux d'apprentissage $\alpha_n(s,a)$ et β_n tendant vers 0 selon les mêmes conditions que pour le Q-learning et pour le même type de compromis entre exploration et exploitation.

Bien que le R-learning soit moins connu et moins utilisé que le Q-learning, il semble qu'il présente des propriétés de vitesse de convergence plus intéressantes en pratique [MAH 96b]. Si peu de résultats théoriques existent à ce sujet, citons toutefois [GAR 98] où l'on montre qu'en horizon fini, le R-learning est très proche d'une version parallèle optimisée du Q-learning, expliquant ainsi ses meilleurs résultats expérimentaux.

D'autres algorithmes d'apprentissage par renforcement pour le critère moyen ont aussi été proposés [MAH 96b]. Parmi eux, l'algorithme B [JAL 89] est une méthode indirecte qui nécessite une estimation adaptative des fonctions $p()$ et $r()$. Citons aussi les travaux de Mahadevan [MAH 96a] qui a défini un algorithme toujours à base de modèles permettant d'apprendre des politiques biais-optimales, sur la base des équations d'optimalité de Bellman.

Nous allons voir à présent deux autres algorithmes qui visent aussi à maximiser le critère moyen, mais en construisant un modèle des transitions. Ils présentent la particularité théorique de disposer d'une preuve de convergence polynomiale en fonction du nombre d'états et non pas exponentielle comme c'est le cas pour les algorithmes précédents. Ils sont aussi connus pour avoir donné lieu à des prolongements dans le cadre des MDP factorisés, ce qui sera évoqué au chapitre 9, dans le tome 2 de cet ouvrage.

2.5. Méthodes indirectes : apprentissage d'un modèle

Nous avons vu à la section 2.4.4 qu'il existait un compromis entre la vitesse d'apprentissage et la mémoire utilisée pour apprendre. La solution consistant à mettre en œuvre des traces d'éligibilité reste limitée en ceci que l'apprentissage n'opère qu'à partir d'informations extraites du passé immédiat de l'agent. Nous allons voir à présent une approche différente dans laquelle l'agent élabore un modèle de ses interactions avec son environnement puis peut apprendre sur la foi de ce modèle, indépendamment de son expérience courante.

La question principale de ce type d'approche est la suivante : faut-il attendre de disposer d'un modèle le plus exact possible avant d'entamer la phase d'optimisation ? Peut-on entrelacer au maximum identification et optimisation, c'est-à-dire modifier à chaque transition observée la fonction de valeur et le modèle estimé ? Il semble actuellement que cette dernière approche soit préférable, comme par exemple dans ARTDP [12] [BAR 95], où acquisition du modèle, programmation dynamique et exécution sont concurrents. Parmi les algorithmes de ce type les plus connus en apprentissage par renforcement, citons aussi *Dyna* [SUT 90a], *Queue-Dyna* [PEN 93] et *Prioritized Sweeping* [MOO 93].

12. *Adaptive Real-Time Dynamic Programming*

2.5.1. *Les architectures* DYNA

Tous les systèmes d'apprentissage par renforcement indirect [13] sont motivés par le même constat. Plutôt que d'attendre qu'un agent effectue des transitions dans son environnement réel, une façon d'accélérer la propagation de la qualité des situations consiste à construire un modèle des transitions réalisées par l'agent et à se servir de ce modèle pour appliquer un algorithme de propagation indépendamment du comportement de l'agent. En effet, quand un agent interagit avec son environnement, ses actions ne débouchent pas seulement sur une possible punition ou récompense, mais aussi sur une situation ultérieure. L'agent peut donc construire un modèle des transitions dont il fait l'expérience, indépendamment de tout modèle des récompenses.

L'idée de construire un modèle des transitions par apprentissage a été réalisée pour la première fois avec les architectures DYNA [SUT 90b]. En effet, Sutton a proposé cette famille d'architectures pour doter un agent de la capacité à mettre à jour plusieurs valeurs d'actions lors du même pas de temps, indépendamment de la situation courante de l'agent, de façon à accélérer sensiblement l'apprentissage de la fonction de valeur. Cette capacité est réalisée en appliquant au modèle un nombre fixé d'étapes de planification, qui consistent à appliquer un algorithme de programmation dynamique au sein du modèle de l'agent [14].

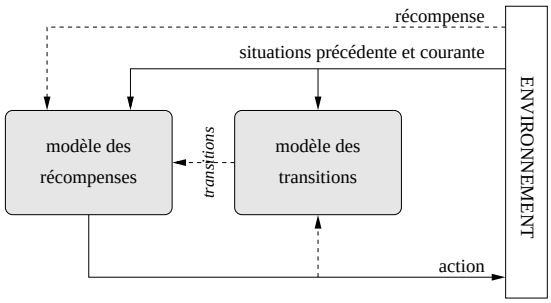


Figure 2.4. *Les architectures* DYNA *combinent un modèle des récompenses et un modèle des transitions. Le modèle des transitions est utilisé pour accélérer l'apprentissage par renforcement. La construction des deux modèles requiert une mémorisation de la situation précédente, ce qui n'est pas explicite sur la figure.*

Les architectures DYNA, présentées sur la figure 2.4, construisent un modèle des transitions effectuées par l'agent dans son environnement et un modèle des récompenses qu'il peut y obtenir.

---

13. *model-based reinforcement learning*
14. Voir le chapitre précédent

Le modèle des transitions prend la forme d'une liste de triplets $\langle s_t, a_t, s_{t+1} \rangle$ qui indiquent que, si l'agent effectue l'action a_t dans l'état s_t, il atteindra immédiatement l'état s_{t+1}.

Des transitions de la forme (s_t, a_t, s_{t+1}) constituent le modèle des interactions d'un agent avec son environnement. Au lieu d'apprendre que tel stimulus doit être suivi de telle action, l'agent apprend alors que, s'il effectue telle action après avoir reçu tel stimulus, alors il doit s'attendre à recevoir tel autre stimulus lors du pas de temps suivant.

Une fois ce modèle construit par apprentissage, on peut appliquer des algorithmes de programmation dynamique tels que l'itération sur les valeurs ou sur les politiques pour accélérer l'apprentissage de la fonction de valeur. Ainsi, le modèle des transitions peut être utilisé indépendamment de la situation courante de l'agent pour propager les valeurs de différents états à partir de diverses sources de récompense. En pratique, ce processus de propagation consiste à réaliser des actions simulées à partir de situations fictives, éventuellement plusieurs fois par pas de temps.

Lorsque l'agent est un robot réel dont chaque action effective peut être longue et difficile à mettre en œuvre, voire dangereuse pour son fonctionnement, réaliser des transitions simulées dans un modèle interne est beaucoup moins coûteux pour propager des valeurs que refaire chaque action laborieusement. En outre, si les buts attribués à l'agent changent, ce qui a pour effet de modifier les récompenses qu'il recevra, alors l'agent est capable de reconfigurer rapidement sa stratégie de comportement plutôt que de « désapprendre » tout ce qu'il a péniblement appris et d'apprendre à nouveau une autre stratégie comportementale.

Par ailleurs, la construction d'un tel modèle peut doter un agent de capacités d'anticipation. En effet, s'il enregistre les associations entre les états dans lesquels il se trouve d'une part et les récompenses qu'il reçoit d'autre part, l'agent peut choisir son action en fonction du caractère désirable pour lui de l'état auquel cette action doit le conduire. Au lieu d'avancer aveuglément vers une récompense attendue, l'agent peut alors mettre en œuvre des capacités de planification à plus long terme. En parcourant par un regard en avant le graphe des transitions d'état en état dont il dispose, l'agent devient capable de prévoir l'évolution de ses interactions avec son environnement en fonction de ses actions à un horizon indéfini. Si bien que, s'il veut atteindre un but dans un certain état, il peut rechercher au sein du graphe la séquence d'action qui le conduit à ce but, avant d'effectuer la séquence d'actions correspondante.

Les architectures *Dyna* constituent une famille de systèmes qui vérifient les principes que nous venons de décrire. Au sein de cette famille, on distingue cependant des variations sur l'algorithme d'apprentissage ou la méthode d'exploration utilisés. Le

système original, *Dyna-PI* [15] repose sur un algorithme d'itération sur les politiques. Sutton [SUT 90b] montre que ce système est moins flexible que le système *Dyna-Q*, qui repose sur le Q-learning. Il propose en outre une version dotée de capacités d'exploration active, nommée *Dyna-Q+* et présente la différence de performance entre ces trois systèmes sur des environnements changeants dans lesquels soit des chemins optimaux sont bloqués, soit, au contraire, des raccourcis apparaissent.

Enfin, tous ces systèmes reposant sur un algorithme de programmation dynamique pour propager efficacement la fonction de valeur ou la fonction de qualité, il est possible de rendre cette programmation dynamique plus efficace avec un algorithme de « balayage prioritaire » tel que *Prioritized Sweeping* [MOO 93], qui met à jour en priorité les valeurs des états dans lesquels l'agent est susceptible de se trouver, ou d'autres variantes telles que *Focused Dyna* [PEN 92], *Experience Replay* [LIN 93] et *Trajectory Model Updates* [KUV 96].

Examinons à présent les algorithmes dotés d'une preuve de convergence en fonction polynomiale de la taille du problème.

2.5.2. *L'algorithme* E^3

Le premier de ces algorithmes s'appelle E^3, pour « *Explicit Explore and Exploit* » [KEA 98]. De même que les architectures DYNA, l'algorithme E^3 repose sur la construction d'un modèle des transitions et des récompenses. Néanmoins, au lieu de chercher à construire un modèle de l'environnement tout entier, il ne mémorise qu'un sous-ensemble des transitions rencontrées, à savoir celles qui jouent un rôle dans la définition de la politique optimale.

Pour construire ce modèle, l'algorithme visite les états de façon homogène, ce qui signifie que, quand il arrive dans un nouvel état, il choisit l'action qu'il a effectuée le moins souvent dans cet état. Il tient alors à jour une probabilité observée de transition vers des états suivants, ce qui revient à se donner un modèle approché du MDP sous-jacent.

Au cœur de l'algorithme E^3 se trouve un mécanisme de gestion du compromis entre exploration et exploitation qui repose sur le principe dit « de la case courrier [16] ». L'idée est que, si l'on visite un nombre fini d'états de façon homogène, il finit toujours par y avoir un état qui a été visité suffisamment souvent pour que l'estimation des probabilités de transition construite sur la base des observations à partir de cet état

15. PI pour *Policy Iteration*. À noter que dans [SUT 98], Sutton appelle ce même système *Dyna-AC*, avec AC pour *Actor Critic*

16. *The pigeon hole principle*

soit proche de la distribution de probabilité effective engendrée par l'environnement. Les auteurs définissent alors une notion d'état « connu » de telle façon que le nombre de visites suffisant pour connaître un état reste polynomial en fonction de la taille du problème.

Le modèle construit par E^3 est un MDP dans lequel figurent tous les états « connus » avec les probabilités de transition observées et un état absorbant qui représente à lui seul tous les états non encore « connus ». L'algorithme est alors face à une alternative :

– soit il existe une politique proche de l'optimum qui repose uniquement sur des états connus et l'algorithme peut trouver cette politique en appliquant un algorithme de programmation dynamique sur le modèle qu'il a construit ;

– soit cette condition n'est pas vérifiée et il faut explorer davantage, donc choisir des actions qui mènent à l'état absorbant afin de mieux connaître les états de l'environnement qu'il regroupe.

Cet algorithme est simple et présente l'intérêt de disposer de preuves qui garantissent le caractère polynomial de la convergence. Pour déterminer dans quelle branche de l'alternative ci-dessus on se trouve, l'algorithme est capable de calculer si explorer davantage au-delà d'un horizon fixé pour mieux connaître le modèle permet d'améliorer de façon significative la meilleure politique connue. Si ce n'est pas le cas, il vaut mieux exploiter qu'explorer.

2.5.3. *L'algorithme* R_{max}

L'algorithme R_{max} [BRA 01] apparaît comme une alternative intéressante à l'algorithme E^3. Il repose sur le même principe général que E^3 consistant à construire un modèle des transitions et à rechercher une politique qui maximise la récompense moyenne sur un horizon fini. Cependant, d'une part, il simplifie la gestion du dilemme entre exploration et exploitation en initialisant toutes les espérances de récompense à une valeur optimiste égale au maximum des récompenses immédiates atteignables dans l'environnement [17] et, d'autre part, il étend le cadre des MDP dans lequel travaille E^3 au cadre des jeux stochastiques à somme nulle, ce qui permet de prendre en compte la présence d'un adversaire. La principale différence provient de ce que l'initialisation des récompenses à une valeur optimiste permet à l'algorithme R_{max} d'éviter d'avoir à gérer explicitement le compromis entre exploration et exploitation. En effet, l'agent se contente d'aller vers les états dans lesquels il s'attend à recevoir une récompense élevée. La récompense attendue peut être élevée pour deux raisons : soit parce que l'état est mal connu et dans ce cas le comportement de l'agent relève de

17. d'où le nom de l'algorithme, R_{max}

l'exploration, soit parce que l'agent sait qu'il a déjà reçu des récompenses dans l'état en question et son comportement relève de l'exploitation.

Les auteurs argumentent la supériorité de leur approche sur celle de l'algorithme E^3 en soulignant que, en présence d'un adversaire, on ne maîtrise pas complètement les transitions du système, donc on ne maîtrise pas totalement le choix entre exploration et exploitation. Avec R_{max}, l'algorithme explore ou exploite « au mieux » compte tenu des choix réalisés par l'adversaire. L'algorithme est globalement plus simple et constitue donc un excellent candidat pour gérer efficacement le compromis entre exploration et exploitation.

Comme l'algorithme E^3, R_{max} repose sur deux hypothèses irréalistes qui compromettent son application en pratique. D'une part, on suppose connu l'horizon T au-delà duquel chercher à améliorer le modèle en explorant ne permet plus d'améliorer la politique de façon significative. D'autre part, on suppose qu'à chaque pas de temps, on est capable de déterminer efficacement la politique optimale sur cet horizon T, compte tenu de la connaissance dont on dispose dans le modèle. La difficulté provient de ce que T peut être grand, si bien que trouver une politique optimale sur cet horizon peut être très long et que donner à T une valeur arbitraire suppose de prendre une valeur encore plus grande que la valeur idéale, ce qui ne fait qu'aggraver le problème.

En vue d'applications pratiques de ces algorithmes conçus essentiellement en fonction de la possibilité de disposer de preuves théoriques de convergence, des variantes ont été proposées pour accélérer la recherche d'une politique optimale en faisant un échantillonnage heuristique [18] des transitions plutôt qu'une exploration systématique homogène [**?**, **?**]. Dans [BRA 03], les auteurs utilisent aussi des valeurs bien inférieures à la valeur de T théorique et montrent que l'algorithme se comporte de mieux en mieux au fur et à mesure que T augmente, au prix d'un temps de calcul croissant.

2.6. Conclusion

L'apprentissage par renforcement est aujourd'hui une discipline très active. À l'interface entre apprentissage automatique et sciences cognitives, contrôle optimal et optimisation par simulation, l'apprentissage par renforcement utilise des techniques variées pour aborder le problème de l'acquisition d'un comportement optimal dans un environnement incertain et dynamique.

Nous avons principalement présenté dans ce chapitre les méthodes classiques de l'apprentissage par renforcement, qui ont historiquement fondé la discipline. Toutefois, l'apprentissage par renforcement étant en plein développement, de nombreuses

18. *heuristic sampling*

méthodes plus récentes n'ont pu être présentées ici. On peut citer par exemple les travaux actuels sur les modèles d'apprentissage par renforcement probablement approximativement corrects (PAC), comme MBIE [**?**] ou Delayed Q-learning l [**?**].

On trouvera enfin dans la suite de cet ouvrage d'autres approches classiques qui partagent également l'emploi de la simulation, telles les méthodes de résolution en ligne (chapitre 10), les méthodes de programmation dynamique avec approximation de la fonction de valeur (chapitre 11) ou encore les méthodes de gradient pour l'optimisation de politiques paramétrées (chapitre 12).

Chapitre 3

Processus Décisionnels de Markov partiellement observables

Si les systèmes rencontrés dans la réalité peuvent souvent être modélisés comme des processus *markoviens*, l'agent chargé de les contrôler ou d'apprendre à les contrôler a rarement accès à une information suffisante pour connaître l'*état* du processus. L'agent *observe* le processus mais ne le *connaît* pas. Le formalisme des *Processus Décisionnels de Markov Partiellement Observable* (ou POMDP) permet justement de modéliser ce genre de situation où un agent n'a accès qu'à des informations souvent partielles sur le processus à contrôler.

EXEMPLE.– Dans l'exemple de l'entretien de la voiture des chapitres précédents (voir pages 18 et 51), nous avons en fait implicitement supposé que nous connaissions l'*état* de la voiture. Bien souvent ce n'est pas le cas, car personne ne va constamment mesurer le taux d'étanchéité du joint de culasse ou l'usure des plaquettes de frein. En nous contentant d'un rapide coup d'oeil, nous n'avons pas accès à l'*état* de la voiture au sens formel mais simplement à une *observation*, souvent incomplète et imprécise, de cet état. Le formalisme des POMDP permet de modéliser ce genre de problèmes et ses solutions indiquent comment choisir optimalement les actions à réaliser en ne s'appuyant que sur ces observations imparfaites. Ppourtant, il faut encore une fois faire l'hypothèse que nous connaissons la dynamique du processus, c'est-à-dire les conséquences des actions (les *transitions*), leur coût (les *récompenses*) mais aussi la probabilité avec lesquelles un état donné de la voiture produit telle ou telle observation (la *fonction d'observation*).

Chapitre rédigé par Alain DUTECH et Bruno SCHERRER.

Cette problématique liée au réalisme des problèmes pouvant être abordés par les méthodes de la programmation dynamique est apparue très tôt. Dés 1965, Aström pose les bases théoriques des POMDP et de leur résolution en utilisant les états de croyance[1] [AST 65]. Il faut attendre les travaux de Smallwood et Sondik pour voir les premiers algorithmes de résolution qui sont très peu efficaces [SMA 73, SON 71, SON 78]. On peut dire que c'est avec l'algorithme WITNESS [CAS 94] puis ITERATIVE PRUNNING [ZAN 96] que la recherche dans le domaine a vraiment pris son essor et, depuis, de nombreux algorithmes exacts ou approchés ont vu le jour.

Ce chapitre est consacré aux POMDP. Dans un premier temps (section 3.1), nous détaillons et explicitons le formalisme en introduisant la notion d'état d'information. Puis, nous montrons que l'application directe des méthodes vues aux chapitres précédents (ce qui revient à essayer de contrôler un POMDP comme si c'était un MDP) est en général vouée à l'échec (section 3.2). Il existe cependant des principes généraux permettant des méthodes exactes de résolution des POMDP (section 3.3) ce qui se traduit par des algorithmes de type « itération sur les valeurs » (3.4) ou « itération sur les politiques » (3.5).

3.1. Définition formelle des POMDP

3.1.1. *Définition d'un* POMDP

Un processus décisionnel de Markov partiellement observable est un MDP dans lequel l'agent ne connaît pas l'état réel du processus : l'agent n'a accès qu'à une observation partielle de cet état ([CAS 98]). On le définit, de manière analogue à un MDP (cf. section 1.2.1, page 18), par : $(\mathcal{S}, \mathcal{A}, \Omega, T, p, O, r, b_0)$ où :

– $\mathcal{S}$ est l'espace d'état ;

– $\mathcal{A}$ est l'espace des actions ;

– Ω est l'espace des observations ;

– T est l'axe temporel ;

– $p()$ sont les probabilités de transition entre états ;

– $O()$ sont les probabilités d'observation sur les états ;

– $r()$ est une fonction de récompense sur les transitions d'états ;

– b_0 est la distribution de probabilité initiale sur les états.

Les seules nouveautés par rapport à un MDP sont donc la distribution initiale des états du processus b_0, l'espace des observations Ω et la fonction d'observation associée $O()$.

1. pour l'anglais « *belief states* »

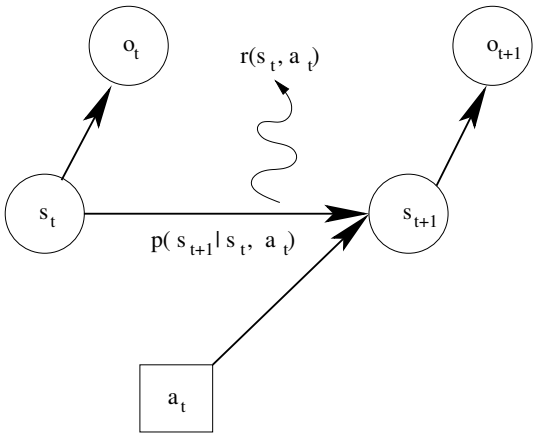


Figure 3.1. ***Vue générale d'un*** POMDP *sous forme d'un diagramme d'influence.*

Le principe général, illustré fig. 3.1 est que, à chaque instant t de T, l'agent ne connaît pas l'état courant $s_t \in \mathcal{S}$ mais peut seulement le percevoir partiellement sous forme d'une observation $o_t \in \Omega$. Cette observation est donnée par la fonction d'observation O. Quand il applique une action $a_t \in \mathcal{A}$ sur le processus, cela modifie alors aléatoirement l'état du processus selon $p()$ pour l'amener dans l'état s_{t+1} alors que l'agent ne perçoit que l'observation o_{t+1} qui dépend de la fonction $O()$. Enfin, comme dans un MDP, l'agent reçoit une récompense $r \in \mathbb{R}$.

Tout comme $\mathcal{S}$ et $\mathcal{A}$, l'espace des observations Ω est supposé fini. Si, dans le cas le plus général, l'observation de l'état courant peut dépendre de la dernière transition en date du processus, le cas le plus classique suppose que cette observation ne dépend que de l'état courant. A un instant t, la probabilité pour l'agent d'observer o dans l'état s est donnée par $\Pr(o|s) = O_t(o|s)$. On impose que $\forall t$, $\forall s$, $\sum_o O_t(o|s) = 1$.

Il est à noter que, même quand la fonction d'observation $O()$ est déterministe, c'est-à-dire qu'à chaque état est associé une et une seule observation, l'agent peut ne pas connaître l'état : deux états peuvent être associés à la même observation. C'est d'ailleurs quand il y a ambiguïté sur l'état qu'il est intéressant de parler de POMDP, car sinon le problème que l'agent doit résoudre possède la structure d'un MDP classique.

C'est donc dans le cadre plus large où les ambiguïtés perceptives peuvent se produire que nous allons nous placer dans la suite de ce chapitre. Nous ferons aussi l'hypothèse que le processus est stationnaire ($O()$ et $p()$ ne dépendent pas du temps).

3.1.2. *Critères de performance*

Tout comme dans le cadre des MDP, la résolution d'un POMDP se fait en cherchant à maximiser un critère de performance donné. Ces critères sont les mêmes que ceux utilisés pour les MDP (voir section 1.2.3, page 22).

Si les critères existent toujours, il n'en est pas forcément de même des fonctions de valeur que nous avons détaillées pour les MDP. En effet, les fonctions de valeur existent bien quand elles sont définies sur l'espace $\mathcal{S}$ des états, or cet espace n'est pas accessible à l'agent. Il est donc temps de parler des états d'informations et donc des espaces associés qui, comme nous le verrons, conditionnent l'existence de ces fonctions de valeur et donc la résolution des POMDP.

3.1.3. *Etat d'information*

L'agent, qui n'a pas accès à l'état, doit choisir ses actions en fonction des informations qui lui sont disponibles, c'est pourquoi nous parlerons d'*état d'information*[2] pour parler des espaces de définition des politiques (ou des fonctions de valeur) permettant *effectivement* à l'agent de contrôler le POMDP de façon optimale.

3.1.3.1. *Définition*

Appelons I_t l'ensemble des informations accessibles à un agent à un instant t. Après avoir effectué l'action a_t, l'agent perçoit le processus sous la forme de l'observation o_{t+1}, ce qui lui permet d'augmenter l'ensemble des informations dont il dispose, par exemple par :

$$I_{t+1} = \tau(I_0, I_1, \ldots, I_t, o_{t+1}, a_t). \tag{3.1}$$

DÉFINITION 3.1.– *Etat d'information d'un* POMDP
I forme un ensemble d'états d'information si la séquence $(I)_t$ définit une chaîne de Markov contrôlée (par les actions), c'est-à-dire si :

$$\forall t,\ \Pr(I_{t+1}|I_0, I_1, \ldots, I_t, o_{t+1}, a_t) = \Pr(I_{t+1}|I_t, o_{t+1}, a_t).$$

Attention, si certains espaces de définition des politiques peuvent paraître naturels (comme par exemple l'ensemble Ω des observations), dans le cas général ces espaces

2. *information state* en anglais

ne sont pas des espaces d'information. La section 3.2 revient plus en détails sur ce problème crucial des POMDP.

On peut alors définir une fonction de transition pour ce processus :

$$\phi(I, a, o, I') \quad = \quad \Pr(I_t = I' | I_{t-1} = I, a, o). \tag{3.2}$$

La récompense associée à ces états d'information s'écrit alors :

$$\rho(I, a) \quad = \quad \sum_{s \in \mathcal{S}} r(s, a) \Pr(s|I). \tag{3.3}$$

En principe, un POMDP peut toujours être transformé en un MDP défini sur un ensemble d'états d'information. Le problème est de trouver une représentation intéressante et utilisable *en pratique* de ces états d'information.

3.1.3.2. *Etat d'information complet*

La façon la plus simple et la plus naïve de représenter les états d'information est d'utiliser *toute* l'information disponible sur le processus depuis le début (depuis l'instant $t = 0$). Dans ce cas, l'état d'information est constitué des historiques complets des observations et actions passées.

Plus formellement, l'*état d'information complet* (noté I_t^C) au temps t est constitué de :

– la distribution de probabilité initiale sur les états b_0,

– l'historique des observations passées et présente $(o_0, \cdots, o_t)$,

– l'historique des actions passées $(a_0, \cdots, a_{t-1})$.

Il est clair que les états d'information complets satisfont à la propriété de Markov énoncée précédemment. Le principal problème de cette représentation est que la taille d'un état d'information grossit à chaque pas de temps. Cette représentation n'est donc pas facilement manipulable, surtout quand on considère des processus à horizon infini.

3.1.3.3. *Statistiques suffisantes*

On peut représenter plus efficacement les états d'information en utilisant des statistiques suffisantes (ou exhaustives) vis-à-vis du contrôle du processus ([BER 95]).

DÉFINITION 3.2.– *Information suffisante*
Une séquence d'états d'information $(I)_t$ définit un processus d'information suffisant quand on a :

– $I_t = \tau(I_{t-1}, o_t, a_{t-1})$,
– $\Pr(s_t|I_t) = \Pr(s_t|I_t^C)$,
– $\Pr(o_t|I_{t-1}, a_{t-1}) = \Pr(s_t|I_{t-1}^C, a_{t-1})$,

où I_{t-1}^C et I_t^C sont des états d'information complets. On peut aussi voir un état d'information suffisant comme un état qui permet de prédire le comportement du processus, ce qui permet ensuite de le contrôler. Ces méta-informations sur le processus satisfont la propriété de Markov et préservent les informations contenues dans les états d'information complets qui sont suffisantes pour contrôler le processus. L'avantage principal des états d'information suffisants est qu'ils peuvent se représenter de façon plus compacte que les états d'information complets. Leur taille n'augmente pas avec le temps, par exemple. Le désavantage vient du fait que leur mise à jour est un peu plus complexe et que leur espace de définition peut être continu.

3.1.3.4. *Etats de croyance (belief states)*

Des états d'information suffisants couramment utilisés sont les états de croyance.

DÉFINITION 3.3.– *Etat de croyance*
Un état de croyance b_t à l'instant t est défini par :

$$b_t(s) = \Pr(s_t = s|I_t^C).$$

Un état de croyance est une distribution de probabilité sur l'ensemble des états. On note $\mathcal{B}$ l'ensemble des états de croyance, c'est l'espace de toutes les distributions de probabilité sur $\mathcal{S}$.

Comme nous l'avons dit, un état d'information suffisant doit être mis à jour après chaque action. Ainsi, si b est état de croyance, nous pouvons exprimer l'état de croyance b_o^a après une transition du processus, c'est-à-dire après que l'agent a effectué une action a et reçu une observation o :

$$\begin{aligned}
b_o^a(s') &= \Pr(s'|b,a,o) \\
&= \frac{\Pr(s',b,a,o)}{\Pr(b,a,o)} \\
&= \frac{\Pr(o|s',b,a)\Pr(s'|b,a)\Pr(b,a)}{\Pr(o|b,a)\Pr(b,a)} \\
&= \frac{\Pr(o|s',b,a)\Pr(s'|b,a)}{\sum_{s''\in\mathcal{S}}\Pr(s''|b,a)\Pr(o|s'',a)} \\
&= \frac{O(o|s')\sum_{s\in\mathcal{S}}\Pr(s'|a,s)\Pr(s)}{\sum_{s\in\mathcal{S}}\sum_{s''\in\mathcal{S}}O(o|s'')\Pr(s''|a,s)\Pr(s)} \\
&= \frac{O(o|s')\sum_{s\in\mathcal{S}}p(s'|s,a)b(s)}{\sum_{s\in\mathcal{S}}\sum_{s''\in\mathcal{S}}O(o|s'')p(s''|s,a)b(s)}.
\end{aligned}$$

On obtient alors

$$b_o^a(s') = \frac{O(o|s') \sum_{s \in \mathcal{S}} p(s'|s,a) b(s)}{\sum_{s \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} O(o|s'') p(s''|s,a) b(s)}. \quad (3.4)$$

Puisque les états de croyance forment un processus markovien, nous pouvons expliciter ce processus, notamment au niveau des fonctions de transition et de récompense. Pour cela, nous allons définir la probabilité conditionnelle d'une observation par

$$\begin{aligned} \omega(b,a,o) &= \Pr(o|b,a) \\ &= \sum_{s \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} O(o|s'',a) p(s''|s,a) b(s). \end{aligned}$$

On peut alors calculer la fonction de transition entre deux états de croyance. Etant donné un état de croyance b et une action a, chaque observation o peut donner lieu à un état de croyance successeur b_o^a différent. La probabilité de transiter vers un état de croyance b' donné est alors égal à la somme de toutes les probabilités de transiter vers des b_o^a égaux à b'. Ce qui s'écrit :

$$\phi(b,a,b') = \Pr(b'|b,a) = \sum_{o \in \Omega} \omega(b,a,o) \delta(b', b_o^a),$$

$$\text{avec} \qquad \delta(x,y) = \begin{cases} 0 & \text{si } x = y, \\ 1 & \text{sinon} \end{cases}$$

Quant à la récompense associée à un état de croyance, on l'obtient aisément par

$$\rho(b,a) = \sum_{s \in \mathcal{S}} r(s,a) b(s).$$

Le processus de Markov ainsi défini sur les états de croyance est difficile à résoudre car il est défini sur un espace d'état continu, celui des distributions de probabilité sur l'espace d'état $\mathcal{S}$. De plus, sauf pour certains POMDP bien particuliers qu'on

appelle « transitoires »[3] (voir section 3.3.2.2), la suite des états de croyance générée par une politique donnée comporte un nombre infini d'états de croyance différents. Néanmoins, nous verrons plus loin (section 3.3) qu'il est parfois possible d'utiliser les propriétés particulières des fonctions de valeurs de ce processus pour proposer des algorithmes exacts ou quasi-exacts.

3.1.4. *Politique*

Les espaces sur lesquels il est possible de définir une politique sont plus divers que dans le cas des MDP. Il est inutile de calculer des politiques définies sur les états ou les historiques d'états puisque ces derniers sont inaccessibles à l'agent. Il est envisageable de définir des politiques sur l'espace des observations ou l'historique des observations. Comme nous le verrons à la section 3.2, il n'est pas possible de garantir l'optimalité de telles politiques mais elles peuvent néanmoins constituer une option viable dans certains cas.

Il est plus intéressant de travailler sur les espaces d'information que nous venons d'aborder dans la partie précédente. Dans l'absolu, il existe une multitude de types d'états d'information possibles, mais qui peuvent tous plus ou moins se ramener à des historiques d'action *et* d'observation sur le processus. C'est notamment le cas des états d'information complets. Nous allons nous intéresser à la manière de représenter les politiques dans ce cas. Cette démarche peut aisément s'adapter à d'autres politiques définies sur d'autres espaces (comme par exemple des historiques d'observation).

Pour un état d'information complet initial I, la politique optimale d'horizon N peut être représentée par un arbre, elle correspond à une sorte de plan conditionnel. En effet, pour un état d'information I, la politique optimale d'horizon 1 est l'action optimale associée à cet état, notons la $a^1(I)$. Intéressons nous maintenant à la politique d'horizon 2 à partir de I, elle commence par une action a. Après avoir choisi cette action, l'agent obtient une observation o et se trouve donc dans le nouvel état d'information I_o^a. S'il veut alors agir optimalement, il choisira alors $a^1(I_o^a)$. On voit que la deuxième action de l'agent dépend de l'observation o perçue, c'est une sorte de plan conditionnel.

Ainsi, une politique d'horizon N pour un état d'information I peut se représenter sous la forme d'un arbre comme celui de la figure 3.2. L'exécution d'un plan conditionnel ou d'une politique d'ordre N à partir d'un état d'information I consiste essentiellement à traverser cet arbre en partant de la racine en suivant, tout à tour, des nœuds « action » et des branches « observation ». Quand on arrive à un nœud, on exécute l'action associée à ce nœud, puis en fonction de l'observation reçue, on navigue vers le nœud suivant le long de la branche associée à l'observation.

---

3. de l'anglais « transient »

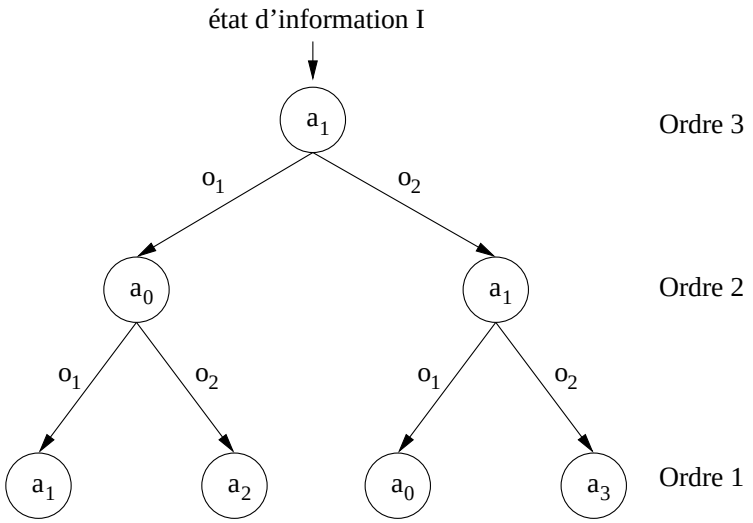


Figure 3.2. Plan conditionnel. *Cet arbre représente une politique d'horizon* 3 *pour un état d'information I donné. C'est aussi une sorte de plan conditionnel. A chaque étape du plan, un nœud contient l'action à effectuer et, en fonction de l'observation reçue, on se déplace vers un nœud d'ordre inférieur en suivant la branche appropriée.*

Les arbres de politiques permettent donc de représenter les politiques d'horizon fini. Par contre, la représentation sous forme d'arbre n'est pas pratique dans le cas des problèmes à horizon infini car les arbres grandissent de manière exponentielle. Une alternative, qui est utilisable dans les cas où les politiques sont *cycliques*, est de représenter les politiques en utilisant des automates à états finis qui permettent de représenter des plans conditionnels qui bouclent. Un exemple d'automate est décrit à la figure 3.3 qui s'exécute comme précédemment : on applique l'action dans le nœud actuel avant de transiter vers le nœud suivant en fonction de l'observation reçue. Le nœud de départ pour appliquer la politique dépend de l'état d'information de départ.

3.1.5. *Fonctions de valeur*

De la même manière que pour les MDP, on peut définir une fonction de valeur pour les POMDP définis sur les états d'information. Nous nous plaçons ici, à titre d'exemple, dans le cadre du critère actualisé, mais les développements faits sur les différents critères pour les MDP en section 1.5 sont utilisables de la même manière.

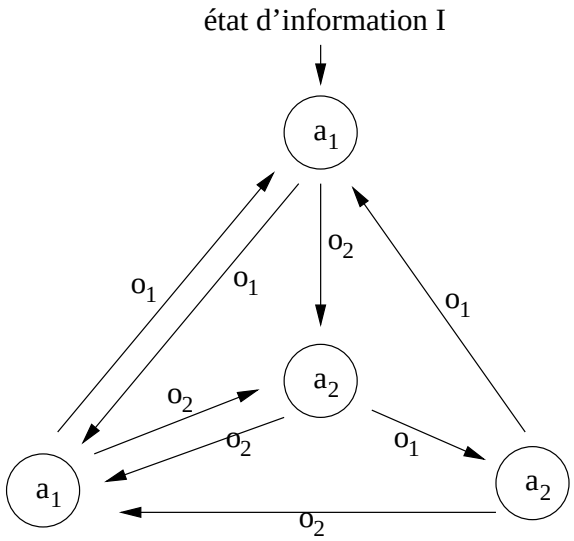


Figure 3.3. Automate à états fini pour politique cyclique infinie. *L'automate permet de représenter des politiques infinies qui présentent des cycles. Comme dans le cas des arbres, on exécute une politique en appliquant l'action dans le nœud actuel avant de transiter vers le nœud suivant en fonction de l'observation reçue.*

Ainsi, l'opérateur d'itération de la fonction de valeur pour une politique π_t définie sur l'espace des états d'information $\mathcal{I}$ s'écrit

$$V_n^{\pi_t}(I) \quad = \quad \rho(I, \pi_t(I)) + \gamma \sum_{I' \in \mathcal{I}} \phi(I, \pi_t(I), I') V_{n-1}^{\pi_t}(I'). \tag{3.5}$$

On peut, de la même manière, écrire l'équation de Bellman définissant la fonction de valeur optimale du MDP défini sur les états d'information

$$\mathcal{V}^*(I) \quad = \quad \max_{a \in \mathcal{A}} \left[\rho(I, a) + \gamma \sum_{I' \in \mathcal{I}(I,a)} \phi(I, a, I') \mathcal{V}^*(I') \right], \tag{3.6}$$

où $\mathcal{I}(I, a)$ est l'ensemble des successeurs de l'état d'information I. Nous avons vu précédemment que ces successeurs se calculaient à partir de I par $I' = \tau(I, o, a)$,

on peut donc en déduire qu'il y a au maximum $|\Omega|$ successeurs à un état d'information et écrire cette équation en sommant cette fois-ci sur l'ensemble des observations possibles, ce qui donne

$$\mathcal{V}^*(I) = \max_{a\in\mathcal{A}}\left[\rho(I,a)+\gamma\sum_{o\in\Omega}\Pr(o|I,a)\mathcal{V}^*(\tau(I,o,a))\right]. \tag{3.7}$$

On peut aussi écrire cette équation en utilisant directement les paramètres du POMDP. On obtient alors :

$$\mathcal{V}^*(I) = \max_{a\in\mathcal{A}}\left[\sum_{s\in\mathcal{S}} r(s,a)\Pr(s|I)\right. \tag{3.8}$$

$$\left.+\gamma\sum_{s_inS}\sum_{s'\in\mathcal{S}}\sum_{o\in\Omega}\Pr(s|I)p(s'|s,a)O(o|s')\mathcal{V}^*(\tau(I,o,a))\right]. \tag{3.9}$$

Comme dans le cas des MDP, l'opérateur associé à cette équation est une contraction pour la norme $\max$ et pour $0 \leq \gamma < 1$ et les mêmes démonstrations assurent l'existence et l'unicité de la solution (voir théorèmes 1.4, 1.5 et 1.6, pages 31-34).

3.2. Problèmes non-markoviens (information incomplète)

Comme nous l'avons signalé, pour un POMDP les observations ne sont pas des états d'information. Autrement dit, un agent ne peut pas contrôler de façon optimale un POMDP en ne se basant que sur l'observation courante du processus pour décider de son action. Dans cette section, nous allons étudier plus en détail les politiques définies seulement sur l'observation courante dans le but de mieux comprendre les différences entre les POMDP et les MDP. Cette compréhension permettra aussi de mieux aborder ensuite les problèmes liés à la résolution des POMDP.

3.2.1. *Politiques adaptées*

Dans cette partie, nous allons uniquement considérer des politiques π de la forme

$$\pi : (\Omega \times T) \longrightarrow \Pi(\mathcal{A}),$$

où $\Pi(\mathcal{A})$ est l'ensemble des distributions de probabilité sur $\mathcal{A}$.

Si, pour essayer de résoudre un POMDP, on adapte les algorithmes définis pour les MDP en assimilant naïvement les observations à des états, on cherche en fait des politiques de la forme que nous venons de rappeler. Nous appelons cette famille de politique des *politiques adaptées*. Et nous allons voir qu'en fait, elles *ne sont pas adaptées* au contrôle optimal d'un POMDP, même si les résultats pratiques peuvent s'avérer satisfaisants.

3.2.2. *Critère pondéré*

3.2.2.1. *Politique adaptée stochastique*

Plaçons-nous dans le cas d'un critère actualisé. Il est alors facile de montrer que, contrairement aux MDP, il n'existe pas de politique adaptée déterministe optimale. L'exemple de la figure 3.4 permet de montrer facilement que :

PROPOSITION 3.1.– *Il existe des* POMDP *où la meilleure politique stochastique adaptée peut être arbitrairement meilleure que la meilleure politique déterministe*.

PREUVE.– La figure 3.4 montre un POMDP avec deux états (1a et 1b) et une seule observation (1). Il n'existe que deux politiques adaptées déterministes (soit tout le temps faire « A », soit tout le temps faire « B »). Au mieux, ces politiques amènent une récompense de $+R$ suivie d'une séquence infinie de $-R$, soit une valeur de $\frac{(1-2\gamma)R}{1-\gamma}$. La politique *stochastique* qui choisit « A » avec une probabilité de 0.5 et « B » avec une probabilité de 0.5 reçoit en moyenne une récompense de 0 à chaque instant. Dès lors, il suffit d'augmenter R et de choisir $\gamma > 0.5$ pour que la différence de valeur entre la politique stochastique et la meilleure politique déterministe soit aussi grande que l'on veut. □

De plus, il est possible, à l'aide d'exemples aussi simples, de montrer que (voir [JAA 94b]) :

PROPOSITION 3.2.– *Il existe des* POMDP *où la meilleure politique stochastique adaptée peut être arbitrairement plus mauvaise que la politique optimale du* MDP *sous-jacent.*

PROPOSITION 3.3.– *Il existe des* POMDP *où la meilleure politique adaptée peut être non-stationnaire.*

3.2.2.2. *Fonction de valeur adaptée*

Les faits précédents amènent à se poser la question de l'existence d'une politique optimale, même stochastique. Pour cela, on peut se poser la question de l'existence d'une fonction de valeur optimale.

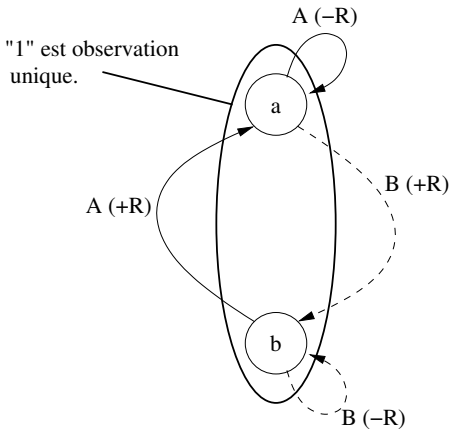


Figure 3.4. Besoin de politiques adaptées stochastiques. *Le* POMDP *de cette figure est constitué de deux états (1a et 1b) qui génèrent tous les deux* la même observation « 1 »*, ce que nous symbolisons par une ellipse.Face à cette observation, l'agent peut choisir entre les actions « A » ou « B » qui, selon l'état sous-jacent, sont associées à des récompenses positives ($+R$) ou négatives ($-R$). On ne peut trouver de politique déterministe optimale.*

Partons d'une politique adaptée $\pi : \Omega \longrightarrow \Pi(\mathcal{A})$, il est possible d'en déduire une politique π' définie sur les états par :

$$\overset{\pi'}{\Pr}(a|s) = \sum_{oin\Omega} \Pr(o|s) \overset{\pi}{\Pr}(a|o) = \sum_{oin\Omega} O(o|s)\pi(a|s). \tag{3.10}$$

Connaissant les états sous-jacents du POMDP, on peut appliquer cette politique π' et en déduire une fonction de valeur $\mathcal{V}_{\pi'}$ qui vérifie :

$$\mathcal{V}_{\pi'}(s) \quad = \quad \sum_{a\in\mathcal{A}} Pr(a|\pi', s) \left[r(s) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s, a)\mathcal{V}_{\pi'}(s') \right].$$

Cette propriété utilise le fait que, pour la politique π', les états sont issus d'un processus qui vérifie la propriété de Markov. Or, ce n'est pas le cas quand on utilise π sur les observations, car les observations ne sont pas des états d'information. On ne peut alors définir la fonction de valeur d'une politique adaptée, c'est-à-dire définie sur les seules observations.

Mais on peut se servir de $\mathcal{V}_{\pi'}$ pour *définir* la fonction de valeur d'une observation comme étant la valeur moyenne des états sous-jacents à cette observation si on avait utilisé π'. Nous appelons cette nouvelle fonction la *fonction de valeur adaptée*.

DÉFINITION 3.4.– *Fonction de valeur adaptée*
Pour une politique adaptée π, il existe une politique π' définie sur les états par l'équation (3.10) qui permet de définir la fonction de valeur adaptée de π de la manière suivante :

$$\vartheta^{\pi}(o) \quad = \quad \sum_{s \in \mathcal{S}} \Pr^{\pi}(s|o) \mathcal{V}_{\pi'}(s), \tag{3.11}$$

où $\Pr^{\pi}(s|o)$ est la distribution asymptotique de probabilité sur les états, c'est-à-dire la probabilité que l'état du MDP *sous-jacent soit s quand on observe o quand t tend vers l'infini et $\mathcal{V}_{\pi'}$ la fonction de valeur de π'.*

Cette définition n'est qu'une définition et ne permet pas à l'agent de calculer la fonction de valeur d'une observation puisqu'il n'a pas accès à s. Néanmoins, munis de cette définition, nous pouvons montrer que, contrairement à un MDP, pour un POMDP, il n'existe pas forcément de politique stationnaire qui maximise la valeur de toutes les observations simultanément. Pour un MDP, la politique optimale maximisait *simultanément* la valeur de tous les états.

La preuve de cette affirmation peut se faire au travers de la figure 3.5. Dans ce POMDP à quatre états (1, 2a, 2b et 3) et trois observations (1, 2 et 3), le seul choix possible d'action se fait pour l'observation « 1 ». Si l'on y augmente la probabilité de choisir l'action « A », on augmente la valeur de l'observation « 1 » et on diminue la valeur de l'observation « 2 ». L'effet inverse se produit si on augmente la probabilité de choisir l'action « B ». On ne peut donc maximiser la valeur de ces deux observations simultanément.

Ainsi, la fonction de valeur adaptée n'a pas du tout les mêmes propriétés que la fonction de valeur d'un MDP. En particulier, on ne peut pas définir de fonction de valeur adaptée optimale, ni donc de politique adaptée optimale. Du moins, en utilisant le critère γ-pondéré.

3.2.2.3. *Convergence des algorithmes adaptés*

Puisqu'il n'existe ni fonction de valeur adaptée optimale, ni politique adaptée optimale, on peut légitimement se demander si les algorithmes définis pour les MDP sont d'une utilité quelconque si on les adapte aux POMDP. En fait, il n'est même pas sûr que ces algorithmes convergent, puisque leur convergence était assurée par l'existence d'une solution optimale dans le cadre des MDP.

Si l'on y regarde de plus près, il y a deux facteurs à prendre en compte. D'une part, la fonction de valeur d'une observation dépend de la probabilité d'occupation des états sous-jacents. On peut alors se dire que les algorithmes qui ne modifient pas cette probabilité sous-jacente ont des chances de converger. D'autre part, nous avons

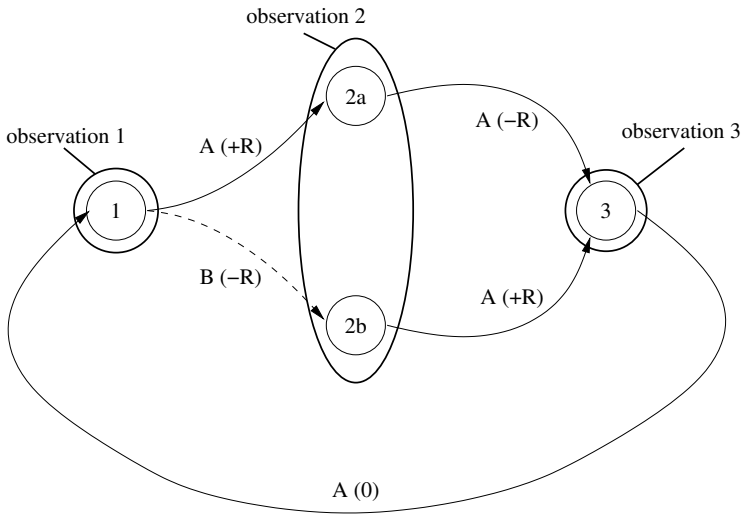


Figure 3.5. Pas de politiques adaptées optimales. *Le* POMDP *de cette figure est constitué de quatre états (1, 2a, 2b et 3) et de trois observations (1, 2 et 3). La seule décision concernant la politique est faite pour l'observation « 1 ». Y augmenter la probabilité de choisir « A » augmente la valeur de « 1 » et diminue celle de « 2 ». C'est un exercice très facile de montrer qu'on ne peut donc trouver de politique maximisant la valeur de toutes les observations.*

vu qu'il n'était pas possible, dans le cas général, de maximiser la fonction de valeur de toutes les observations simultanément. Dès lors, il paraît difficile de prévoir le comportement d'algorithmes qui essaient justement de trouver cette fonction de valeur dominante (comme par exemple *Value Iteration* ou *Policy Iteration* car l'étape non-linéaire de maximisation effectuée à chaque itération est une source d'instabilité potentielle). On peut néanmoins penser que des algorithmes qui explorent de manière stationnaire l'environnement et s'appuient sur une approximation stochastique de la fonction de valeur peuvent avoir une chance de converger. Les résultats démontrés dans la littérature semblent confirmer ces intuitions, puisque la convergence de deux algorithmes entrant dans le cadre énoncé plus haut a été prouvée (voir [SIN 94a]).

TD(0**) :** Dans un POMDP, sous les conditions classiques de convergence, l'algorithme TD(0) converge avec une probabilité de 1 vers la solution du système d'équations ci-dessous

$$\vartheta^{\pi}(o) \quad = \quad \sum_{s \in \mathcal{S}} \overset{\pi}{\Pr}(s|o) \left[r(s) + \gamma \sum_{o' \in \Omega} \overset{\pi}{\Pr}(s, o') \vartheta^{\pi}(o') \right], \tag{3.12}$$

où $\Pr^{\pi}(s, o') = \sum_{s' \in \mathcal{S}} \Pr^{\pi}(s'|s) O(o'|s')$.

Il est à noter que la fonction de valeur vers laquelle on converge n'est pas forcément la fonction de valeur définie à l'équation (3.11). Il est d'ailleurs possible de montrer sur des exemples assez simples que cela n'est pas le cas.

Q-Learning : Dans un POMDP, un algorithme de Q-Learning *qui utilise une politique d'exploration stationnaire* $\pi_{\exp}$ converge vers la solution du système d'équations suivant avec une probabilité de 1 (sous les conditions classiques de convergence).

$$Q(o,a) = \sum_{s \in \mathcal{S}} \Pr^{\pi_{\exp}}(s|o,a) \left[r(s,a) + \gamma \sum_{o' \in \Omega} \overset{a}{\Pr}(s,o') max_{a' \in \mathcal{A}} Q(o',a') \right],$$

où $\Pr^{\pi_{\exp}}(s|o,a)$ est la probabilité asymptotique d'occupation de s sous la politique d'exploration $\pi_{\exp}$ sachant qu'on observe o après avoir effectué l'action a et où $\Pr^a(s,o') = \sum_{s' \in \mathcal{S}} p(s'|s,a) O(o'|s')$.

Il est important de noter que le Q-Learning ne converge que si on explore l'environnement avec une politique stationnaire (pas d'algorithme ϵ-greedy par exemple). En outre, le Q-Learning est limité par le fait qu'il converge vers une politique déterministe dont nous avons vu qu'elle pouvait être clairement sous-optimale.

Ces limitations posent avec insistance le problème de l'apprentissage dans un POMDP car, comme nous le verrons, la recherche de solutions optimales quand on connaît le modèle est en partie résolu.

3.2.3. *Algorithmes adaptés et critère moyen adapté*

Nous venons de voir qu'il n'est pas possible de définir une fonction de valeur qui soit optimale sur *toutes* les observations à la fois. Afin de comparer deux politiques adaptées entre elles et donc de définir une politique adaptée optimale, l'idée est de transformer cette fonction de valeur sur les observations en un scalaire, par exemple en écrivant que l'on cherche à maximiser $\hat{\vartheta} = \sum_{o \in \Omega} P_\Omega(o) \vartheta^\pi(o)$ où P_Ω est une distribution de probabilité sur Ω. Parmi les choix possibles pour cette distribution, on peut naturellement penser à la distribution initiale sur les observations, ou à la probabilité asymptotique des observations. Cette dernière solution est en fait équivalente à utiliser le *critère moyen* et donc le gain associé U^π (voir partie 1.5.4), comme l'ont montré les auteurs de [SIN 94a].

En effet, si on pose $\Pr^{\pi}(o)$ comme étant la probabilité asymptotique d'une observation en suivant une politique π, on peut écrire le critère scalaire ci-dessus comme :

$$\begin{aligned}
\sum_{o\in\Omega} \overset{\pi}{\Pr}(o)\vartheta^{\pi}(o) &= \sum_{o\in\Omega} \overset{\pi}{\Pr}(o) \sum_{s\in\mathcal{S}} \overset{\pi}{\Pr}(s|o)V^{\pi}(s) \\
&= \sum_{s\in\mathcal{S}} \sum_{o\in\Omega} \overset{\pi}{\Pr}(o)\overset{\pi}{\Pr}(s|o)V^{\pi}(s) \\
&= \sum_{s\in\mathcal{S}} \overset{\pi}{\Pr}(s)V^{\pi}(s) \\
&= \sum_{s\in\mathcal{S}} \overset{\pi}{\Pr}(s)r(s,\pi(s)) + \gamma \sum_{s\in\mathcal{S}} \overset{\pi}{\Pr}(s) \sum_{s'\in\mathcal{S}} p(s'|s,\pi(s))V^{\pi}(s') \\
&= U^{\pi} + \gamma \sum_{s'\in\mathcal{S}} \overset{\pi}{\Pr}(s')V^{\pi}(s') \\
&= U^{\pi} + \gamma \sum_{o\in\Omega} \overset{\pi}{\Pr}(o)V^{\pi}(o)
\end{aligned}$$

et donc $\sum_{o\in\Omega} \Pr^{\pi}(o)V^{\pi}(o) = \frac{U^{\pi}}{1-\gamma}$.

Il est alors tentant de chercher une politique adaptée stochastique optimale au sens de ce critère scalaire en cherchant tout simplement la politique adaptée stochastique qui optimise la récompense moyenne. A notre connaissance, le seul algorithme pour trouver une politique de ce type a été proposé par Jaakkola, Singh et Littman [JAA 94b]. Cet algorithme de type « itération sur les politiques » s'appuie sur une méthode de Monte Carlo pour évaluer la récompense moyenne d'une politique, ce qui permet de calculer des *Q-valeurs* et d'améliorer la politique courante. Le problème majeur de cette méthode, outre le fait qu'elle n'a jamais été testée en pratique, est qu'elle ne converge au mieux que vers un maximum local de la fonction de valeur scalaire.

Le chapitre sur les méthodes de gradient pour la recherche de politique optimale (chapitre 12) exposera d'autres méthodes approchées pour résoudre les POMDP, notamment lors de la section 12.2.4, page 369.

3.3. Calculer une politique exacte sur les états d'information

Par l'intermédiaire des états d'information, nous savons transformer un POMDP en un MDP. En théorie, les outils de résolution des MDP peuvent être utilisés pour trouver une politique optimale. En pratique, le problème est plus délicat, notamment à cause de la taille et de la nature de l'espace des états d'information. En fait, la résolution

d'un POMDP ayant un seul état initial, en horizon fini, est un problème PSPACE-dur [PAP 87].

NOTE.– Dans la suite de ce chapitre, nous allons travailler avec le critère actualisé et la fonction de valeur associée V.

3.3.1. *Cas général*

3.3.1.1. *Horizon fini*

En théorie, pour un horizon fini N, il suffit d'utiliser le principe de la programmation dynamique pour, d'une manière similaire à l'algorithme 1.1, trouver la politique et la fonction de valeur optimales. Ainsi, partant de la fonction de valeur $V_0^* = \max_{a \in \mathcal{A}} \rho(I, a)$ au dernier pas de temps, on applique récursivement l'opérateur de programmation dynamique (équ. (3.7), section 3.1.5) N fois de la manière suivante :

$$V_n^*(I) \quad = \quad \max_{a \in \mathcal{A}} \left[\rho(I,a) + \gamma \sum_{o \in \Omega} \Pr(o|I,a) V_{n-1}^*(\tau(I,o,a)) \right]. \tag{3.13}$$

L'équation (3.13) définit aussi un opérateur que nous noterons L. On a alors $V_n^* = LV_{n-1}^*$.

Une fois que l'on connaît la fonction de valeur optimale, la politique optimale μ_n^* s'en déduit par :

$$\mu_n^*(I) \quad = \quad \operatorname*{argmax}_{a \in \mathcal{A}} \left[\rho(I,a) + \gamma \sum_{o \in \Omega} \Pr(o|I,a) V_{n-1}^*(\tau(I,o,a)) \right].$$

3.3.1.2. *Horizon infini*

Plaçons nous maintenant dans le cas d'un horizon infini. L'opérateur L que nous venons de définir possède les mêmes propriétés que l'opérateur utilisé dans les MDP à la section 1.5.2, c'est une contraction (pour toutes fonctions U et V, on a $\|LU - LV\| \leq \gamma \|U - V\|$ où $\|V\| = \max_{\mathcal{I}} V(I)$).

Il est alors possible de trouver une fonction de valeur ϵ-optimale en utilisant une méthode de type itération sur les valeurs dont le pas d'itération est :

$$V_{i+1} \quad = \quad LV_i. \tag{3.14}$$

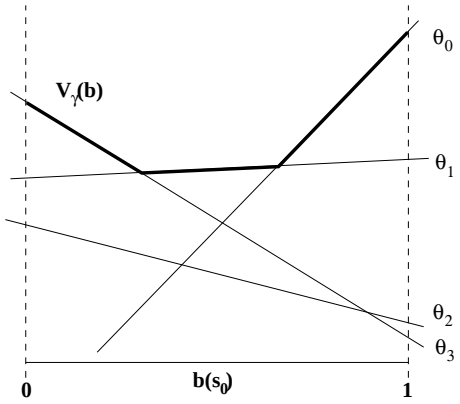


Figure 3.6. fonction de valeur convexe linéaire par morceaux. *La fonction de valeur V d'un* POMDP *avec deux états (s_0 et s_1) est représenté à l'aide de 4 vecteurs de paramètres θ_i, chacun de dimension 2. On trouve l'espace des états d'information le long de l'axe des abscisses et les valeurs sont sur l'axe des ordonnées. Une seule probabilité $b(s_0)$ permet de décrire l'état de croyance car $b(s_1) = 1 - b(s_0)$. Sur cette figure, chaque segment linéaire de la fonction de valeur est tracé avec une ligne fine tandis que la fonction de valeur elle-même est indiquée en gras.*

Puisqu'il existe une fonction de valeur optimale V^* et que L est une contraction, on peut utiliser le théorème de Banach pour prouver que cette méthode d'itération sur les valeurs converge vers V^*. Dès lors, il suffit d'un nombre fini d'itérations de la méthode pour atteindre une solution ϵ-optimale d'où il est possible de déduire une politique optimale.

3.3.2. *Etats de croyance et fonction de valeur linéaire par morceaux*

En pratique, les deux schémas de calcul évoqués dans la partie précédente sont difficilement utilisables. Le problème vient du fait que les espaces d'états d'information sont continus ou de taille conséquente et il peut alors être impossible de calculer ou de représenter les fonctions de valeur et les politiques optimales.

En travaillant avec les états de croyance et les POMDP qui admettent de tels états d'information, il est possible d'exploiter les propriétés particulières des fonctions de valeur pour proposer des algorithmes plus efficaces.

La fonction de valeur optimale pour un problème à horizon fini est linéaire par morceaux et convexe (LPMC) [SON 71]. C'est une propriété très importante car elle permet de représenter la fonction de valeur avec un nombre fini de paramètres, ainsi que le montre la figure 3.6.

Comme la fonction de valeur est définie sur l'espace $\mathcal{B}$ qui est de dimension $|\mathcal{S}|-1$, chaque segment linéaire utilisé dans sa représentation est de dimension $|\mathcal{S}|$ et peut donc être représenté par un vecteur θ avec $|\mathcal{S}|$ coefficients. Pour ces vecteurs, $\theta(s)$ sera la s-ième composante de ce vecteur. L'ensemble des vecteurs permettant de représenter la fonction de valeur LPMC est noté Θ. On dit que Θ représente V, ce qui se traduit par :

$$V(b) = \max_{\theta\in\Theta} \sum_{s\in\mathcal{S}} b(s)\theta(s) \tag{3.15}$$

$$= \max_{\theta\in\Theta} b.\theta. \tag{3.16}$$

Il reste maintenant à démontrer que la fonction de valeur optimale est bien linéaire par morceaux et convexe. Cette démonstration s'appuie sur le théorème suivant démontré par Smallwood.

THÉORÈME 3.1.– *(Fonction de valeur linéaire par morceaux et convexe). ([SMA 73]) Soit L l'opérateur de Bellman défini à l'équation (3.13) et soit V_{init} une fonction de valeur initiale qui est linéaire par morceaux et convexe, définie sur l'espace $\mathcal{B}$ des états de croyance. Alors, pour un* POMDP *admettant des états de croyance, on a :*

– $V_n = L^n V_{\text{init}}$, obtenue après n applications de l'opérateur L sur V_{init}, est elle aussi linéaire par morceaux et convexe sur $\mathcal{B}$;

– V_n peut être représentée par un ensemble fini $\Theta = \{\theta\}$ de vecteurs de taille $|\mathcal{S}|$ par $V_n(b) = \max_{\theta\in\Theta} b.\theta$.

PREUVE.– Nous allons montrer que si, après $i-1$ applications de l'opérateur L, la fonction de valeur V_{i-1} est linéaire par morceaux et convexe, alors la fonction de valeur V_i obtenue après une nouvelle application de l'opérateur est elle aussi LPMC.

Supposons donc que V_{i-1} est LPMC, il existe alors Θ_{i-1} tel que

$$V_{i-1} = \max_{\theta_{i-1}\in\Theta_{i-1}} \sum_{s'\in\mathcal{S}} b_{i-1}(s')\theta_{i-1}(s').$$

Si a est l'action effectuée alors qu'il restait i actions à choisir et o l'observation reçue après. Alors, on peut écrire l'état de croyance b_{i-1} comme étant :

$$b_{i-1} = \Pr(s'|b_i, a, o),$$

ce qui nous permet d'obtenir :

$$V_{i-1} = \max_{\theta_{i-1}\in\Theta_{i-1}} \sum_{s'\in\mathcal{S}} \Pr(s'|b_i, a, o)\theta_{i-1}(s').$$

On peut alors substituer V_{i-1} dans l'équation (3.13) définissant l'opérateur L, ce qui donne :

$$V_i(b_i) = \max_{a \in \mathcal{A}} \left[\rho(b_i, a) + \gamma \sum_{o \in \Omega} \Pr(o|b_i, a) \max_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} \Pr(s'|b_i, a, o) \theta_{i-1}(s') \right],$$

que l'on peut réécrire en :

$$\begin{aligned}
V_i(b_i) &= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} r(s, a) b_i(s) + \gamma \sum_{o \in \Omega} \Pr(o|b_i, a) \max_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} \Pr(s'|b_i, a, o) \theta_{i-1}(s') \right] \\
&= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} r(s, a) b_i(s) + \gamma \sum_{o \in \Omega} \max_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} \Pr(o|b_i, a) \Pr(s'|b_i, a, o) \theta_{i-1}(s') \right] \\
&= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} r(s, a) b_i(s) + \gamma \sum_{o \in \Omega} \max_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} \Pr(s', o|b_i, a) \theta_{i-1}(s') \right] \\
&= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} r(s, a) b_i(s) + \gamma \sum_{o \in \Omega} \max_{\theta \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} \left[\sum_{s \in \mathcal{S}} \Pr(s', o|s, a) b_i(s) \right] \theta(s') \right].
\end{aligned}$$

Soit $\theta_{i-1}^{a,o}(b)$ l'élément optimal de Θ_{i-1} pour b, a et o donnés, c'est-à-dire $\theta_{i-1}^{a,o}(b) = \operatorname{argmax}_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s \in \mathcal{S}} \Pr(s|b_i, a, o) \theta_{i-1}(s)$. Cela revient à chercher le segment de droite qui définit la fonction de valeur pour b, a et o donnés. Alors, nous pouvons écrire :

$$\begin{aligned}
V_i(b_i) &= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} r(s, a) b_i(s) + \gamma \sum_{o \in \Omega} \sum_{s' \in \mathcal{S}} \left[\sum_{s \in \mathcal{S}} \Pr(s', o|s, a) b_i(s) \right] \theta_{i-1}^{a,o}(b_i, s') \right] \\
&= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} b_i(s) \left[r(s, a) + \gamma \sum_{o \in \Omega} \sum_{s' \in \mathcal{S}} \Pr(s', o|s, a) \theta_{i-1}^{a,o}(b_i, s') \right] \right].
\end{aligned}$$

L'expression entre les crochets intérieurs de l'équation précédente peut se représenter par $|\mathcal{A}||\Theta_{i-1}|^{|\Omega|}$ différents vecteur de taille $|\mathcal{S}|$: il faut en effet au maximum un vecteur par choix de a et d'une séquence de $|\Omega|$ vecteurs de Θ_{i-1}. On peut dire que ces vecteurs forment alors l'ensemble Θ_i, ce qui permet d'écrire $V_i(b_i)$ comme étant :

$$V_i(b_i) \quad = \quad \max_{\theta_i \in \Theta_i} \sum_{s \in \mathcal{S}} b_i(s) \theta_i(s).$$

Cela signifie que V_i est bien une fonction linéaire par morceaux et convexe et qu'elle peut se définir par un ensemble fini de vecteur θ_i de Θ_i. Chacun de ces vecteurs θ_i est de la forme :

$$\theta_i(b, s) \quad = \quad r(s, a) + \gamma \sum_{o \in \Omega} \sum_{s' \in \mathcal{S}} \Pr(s', o|s, a) \theta_{i-1}^{a,o}(b, s'). \tag{3.17}$$

Comme la fonction de valeur initiale V_{init} est linéaire par morceaux et convexe, on en déduit aisément qu'après tout nombre fini d'applications de l'opérateur L on obtient bien une fonction de valeur qui est elle aussi LPMC, ce qui conclut la démonstration. □

Pour montrer que la fonction de valeur est LPMC en utilisant le théorème précédent, il ne nous reste plus qu'à montrer que toute fonction de valeur optimale pour un horizon de taille 1 est LPMC. Quand il ne reste plus qu'une action a à choisir, seule la récompense immédiate intervient dans la définition de la fonction de valeur. On a alors immédiatement :

$$\begin{aligned} V_1^*(b) &= \max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} b(s) r(s,a) \\ &= \max_{a \in \mathcal{A}} b.r(a). \end{aligned}$$

V_1^* est donc trivialement LPMC et peut être représentée par au plus $|\mathcal{A}|$ vecteurs.

Ainsi, en utilisant le théorème précédent, on obtient alors immédiatement que toute fonction de valeur optimale pour un horizon fini est linéaire par morceaux et convexe. De plus, la démonstration du théorème étant constructive, on sait aussi que :

– on peut toujours calculer la fonction de valeur optimale pour un horizon fini en un nombre fini d'opérations ;

– la fonction de valeur optimale pour un horizon fini peut être représentée par un ensemble fini de vecteurs de taille $|\mathcal{S}|$;

– la politique optimale est donc calculable elle aussi en un temps fini.

3.3.2.1. *Choix des vecteurs θ*

Une fonction de valeur optimale pour un horizon fini peut être représentée par un nombre fini de vecteurs. Nous avons vu dans la démonstration du Théorème 3.1 qu'il fallait au maximum $|\mathcal{A}||\Theta_{i-1}|^{|\Omega|}$ vecteurs pour représenter V_i. En fait, on peut encore réduire ce nombre en ne considérant que l'ensemble Ω^{poss} des observations possibles après avoir exécuté une action dans un état de croyance donné. La borne devient donc $|\mathcal{A}||\Theta_{i-1}|^{|\Omega|^{\text{poss}}}$. En pratique, il faut beaucoup moins de vecteurs pour représenter la fonction de valeur. Certains vecteurs (comme par exemple le vecteur θ_2 dans la figure 3.6) sont en effet dominés par les autres et leur omission n'influence pas la fonction de valeur. On appelle ce type de vecteur un *vecteur dominé*. Inversement, un vecteur qui permet de calculer la valeur optimale pour au moins un point de l'espace des états de croyance est appelé un *vecteur utile*.

On comprend aisément que, pour faciliter le calcul des fonctions de valeur, il faut minimiser le nombre de vecteurs utilisés pour représenter les fonctions. Pour un état

de croyance I donné, il faut utiliser chaque vecteur de la représentation pour trouver la fonction de valeur. Il est donc très intéressant d'éliminer tous les vecteurs dominés d'une représentation, ce qui est un problème très difficile et [LIT 95a] a montré que cela ne pouvait être fait efficacement que si RP=NP, c'est-à-dire si *tous* les algorithmes de décision non-déterministes ont une probabilité *significative* de succès. La recherche des fonctions de valeur doit donc faire face à une possible explosion du nombre de vecteurs (taille exponentielle en $|\Omega|$), mais aussi à la difficulté de trouver les vecteurs utiles.

Les algorithmes de recherche de fonction optimale que nous allons détailler dans la partie 3.4 explorent plusieurs méthodes pour rendre efficace cette recherche des vecteurs utiles à la représentation.

3.3.2.2. *Horizon infini*

Bien que chaque fonction de valeur V_n^* soit linéaire par morceaux et que

$$\lim_{n\to\infty} \|V_n^* - V^*\| \quad = \quad 0,$$

rien ne dit que la fonction de valeur optimale pour un horizon infini soit elle aussi linéaire par morceaux. [SON 71] a montré qu'il existe une classe de POMDP où V^* est LPMC, il l'appelle la classe des POMDP *transitoires*[4]. Quand le POMDP n'est pas transitoire, on peut approcher aussi précisément que l'on veut la fonction de valeur optimale ainsi que la politique optimale en horizon infini par une politique transitoire admettant une fonction de valeur optimale. On parle de solution ϵ-optimale.

Par contre, toutes les fonctions de valeur en horizon infini, même celles qui ne sont pas transitoires, sont convexes.

3.4. Algorithmes exacts d'itération sur les valeurs

3.4.1. *Etapes de l'opérateur de programmation dynamique*

Le problème crucial l'application de l'opérateur de la programmation dynamique est la construction de l'ensemble de vecteurs Θ_n à partir de Θ_{n-1}. Cette opération est résumée (on peut aussi dire cachée) dans l'équation (3.17). Nous allons détailler le principe de cette construction et en donner un exemple qui essayera d'expliciter les principales notions utilisées dans les algorithmes que nous allons étudier par la suite.

4. de l'anglais « transient »

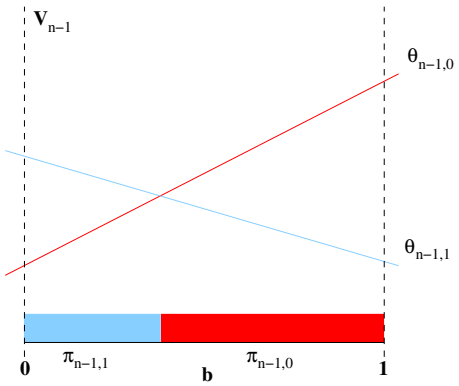


Figure 3.7. ***fonction de valeur à l'étape*** $n-1$. *Ce* POMDP *à deux états sert d'exemple pour expliciter la construction de l'ensemble* Θ_n. *A l'étape* $n-1$, *la fonction de valeur est représentée par deux vecteurs, qui sont chacun associé à une politique. Les régions de dominance sont démarquées par les barres au bas de la figure. Les politiques associées à ces régions sont indiquées sous les barres.*

Partons donc d'un ensemble Θ_{n-1} de vecteurs décrivant la fonction de valeur optimale pour un horizon de taille $n-1$. Chacun des vecteurs θ_{n-1} domine les autres dans une région de l'espace $\mathcal{B}$ des états de croyance et, fait important, représente aussi la meilleur politique à suivre dans cette région. On peut associer chaque vecteur θ_{n-1} à une politique π_{n-1} d'horizon $n-1$. La figure 3.7 illustre le cas qui nous servira d'exemple, avec un ensemble de 2 vecteurs ($\theta_{n-1,0}$ et $\theta_{n-1,1}$). Les régions où dominent les vecteurs sont représentées par la barre grisée le long de l'axe des abscisses.

La construction de l'ensemble Θ_n se comprend plus facilement en la décomposant et c'est aussi la démarche suivie par les deux algorithmes que nous allons présenter. Supposons que nous voulons calculer $V_n^{a1,o1}$ après avoir effectué l'action $a1$ et observé $o1$. On sait alors que la fonction de valeur sera représentée par les vecteurs $\{\theta_n^{a1,o1}\}$ donnés par l'équation (3.17) adaptée à ce cas précis, c'est-à-dire :

$$\theta_n^{a1,o1}(b,s) \quad = \quad \frac{r(s,a1)}{|\Omega|} + \gamma \sum_{s' \in \mathcal{S}} p(s,a1s') O(s',o1) \theta_{n-1}^{a1,o1}(b^{a1,o1},s) \quad (3.18)$$

Cette fonction de valeur sera représentée avec au plus $|\Theta_{n-1}|$ vecteurs. La figure 3.8 illustre cette transformation de la fonction de valeur. Il est à noter que cette fonction de valeur est un peu particulière puisqu'elle prend en compte la probabilité que $o1$ soit observée, ce qui se traduit par le terme $\frac{r(s,a)}{|\Omega|}$ puisque nous avons fait le choix ici de distribuer uniformément la récompense immédiate sur les observations.

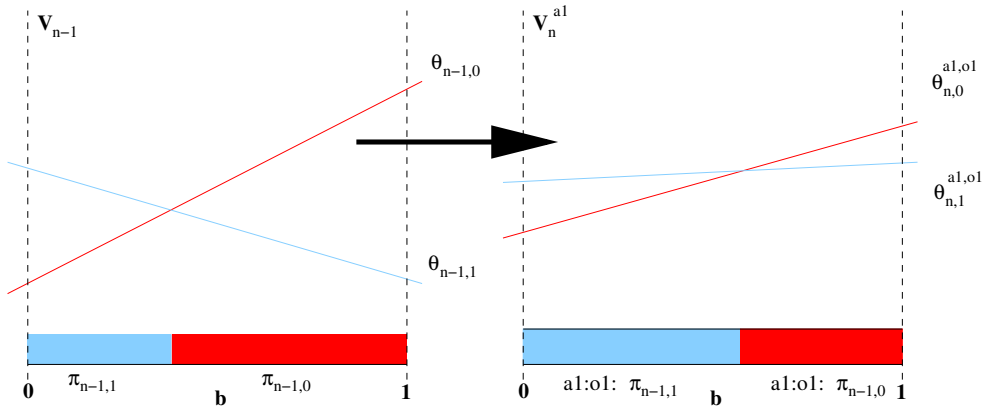


Figure 3.8. ***fonction de valeur à l'étape*** n ***pour*** $a1$ ***et*** $o1$. *Pour une action et une observation données, la nouvelle fonction de valeur se représente avec, au plus, le même nombre de vecteur. Ces vecteurs définissent des nouvelles régions où les politique d'horizon* n *commencent toutes par* $a1 : o1$ *avant d'utiliser la meilleure politique d'horizon* $n - 1$.

Avec $a1$ toujours fixé, on peut ainsi calculer la fonction de valeur pour chaque observation o possible. L'idée est maintenant d'utiliser ces fonctions de valeur pour calculer la fonction de valeur V_n^a reçue après avoir effectué l'action $a1$. C'est aussi une fonction LPMC. C'est en fait la moyenne des fonctions de valeur associés à chaque couple $(a1, o)$. Etant donné que les fonctions $V_n^{a1,o1}$ prennent déjà en compte les probabilités d'observation, il est trivial de montrer que, pour chaque état de croyance b on a :

$$\theta_n^{a1}(b) \quad = \quad \sum_{o \in \Omega} \theta_n^{a1,o}(b). \tag{3.19}$$

De cette manière, nous générons en fait au plus $|\Theta_{n-1}|^{|\Omega^{\text{succ}}(a1)|}$ vecteurs. Pour chaque vecteur, nous aurons un ensemble de politiques qui commencent toutes par l'action $a1$ et qui dépendent ensuite de l'observation reçue. Cette étape est illustrée par la figure 3.9.

Il reste enfin à combiner les fonctions de valeur calculées pour chaque action afin de déterminer la fonction de valeur V_n pour un horizon n. Pour chaque état de croyance, la fonction est représentée par le meilleur vecteur V_n^a et on a :

$$\theta_n(b) \quad = \quad \max_{a \in \mathcal{A}} \theta_n^a(b). \tag{3.20}$$

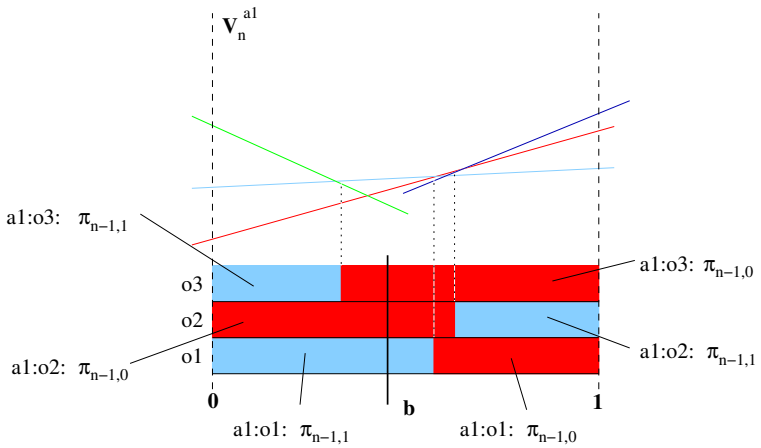


Figure 3.9. ***fonction de valeur à l'étape*** n ***pour*** $a1$. *Pour une action donnée, la nouvelle fonction de valeur se représente avec des vecteurs qui sont sommes des vecteurs* $\theta_n^{a1,o}$. *Ces vecteurs définissent des nouvelles régions où les politique d'horizon* n *commencent toutes par* $a1$ *avant d'utiliser la meilleure politique d'horizon* $n-1$ *en fonction de l'observation qui* sera *perçue. Par exemple, pour l'état de croyance* b *indiqué, après avoir effectué* $a1$*, il faudra utiliser* $\pi_{n-1,1}$ *si* $o = o1$ *et* $\pi_{n-1,0}$ *sinon.*

C'est à cette étape qu'il est possible d'éliminer le plus de vecteurs qui ne sont pas utiles pour représenter la fonction de valeur. La figure 3.10 explicite cette étape en combinant les fonctions de valeur V_n^{a1} et V_n^{a2} de deux actions, en éliminant les vecteurs inutiles (indiqué en pointillés).

3.4.2. *Obtenir une représentation parcimonieuse de* V

Nous définissons ici la notion de représentation parcimonieuse de V et présentons quelques techniques pour obtenir cette représentation parcimonieuse en élaguant les vecteurs dominés.

3.4.2.1. *Région*

Un ensemble Θ donné définit une partition sur l'espace des vecteurs estimés $\mathcal{B}$. Chaque partie de cette partition est associée à un vecteur θ de Θ. Par exemple, sur la figure 3.7, la partition est constituée de deux parties. La région $R(\theta, \Theta)$ associée à un vecteur est la portion de $\mathcal{B}$ où ce vecteur domine les autres.

DÉFINITION.– *Soient un espace d'états de croyance* $\mathcal{B}$ *et une représentation d'une fonction de valeur* Θ*, la région* $R(\theta, \Theta)$ *associée à un vecteur* θ *de* Θ *est définie par :*

$$R(\theta, \Theta) \quad = \quad \{b \mid b.\theta > b.\theta',\ \theta' \in \Theta - \{\theta\},\ b \in \mathcal{B}\}. \tag{3.21}$$

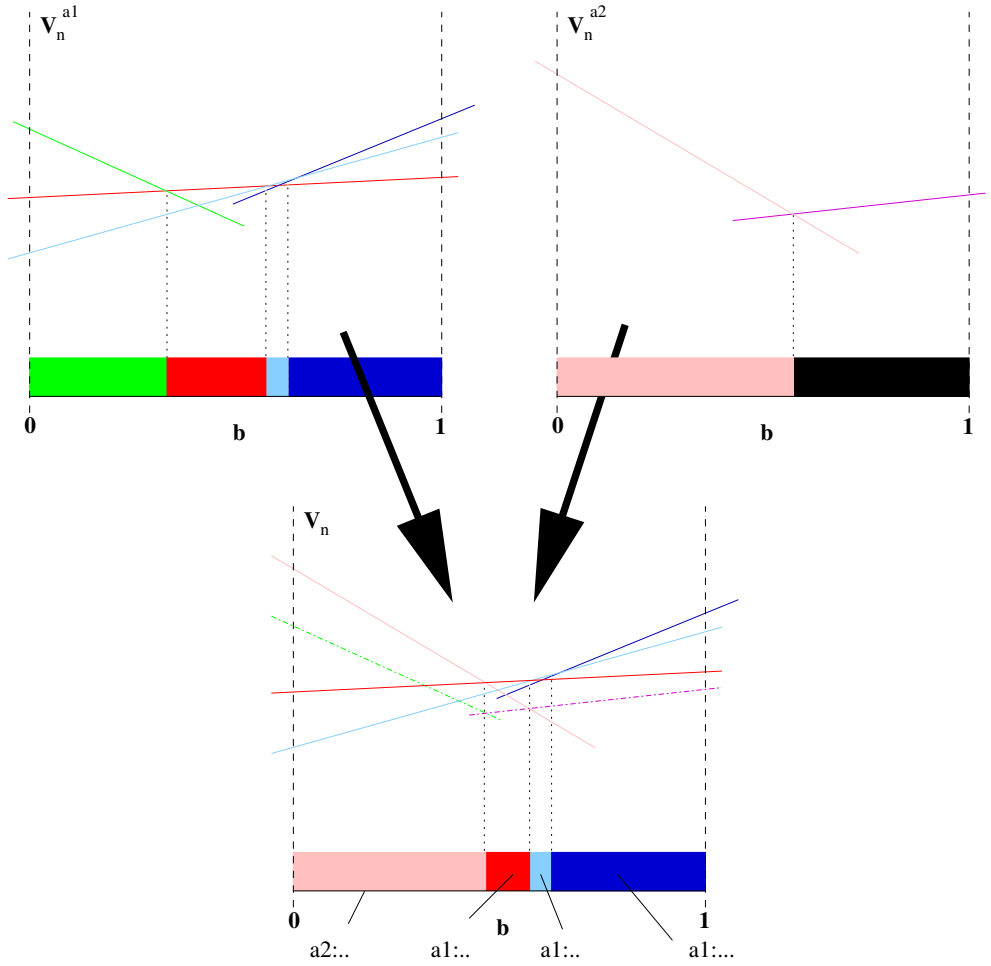


Figure 3.10. ***fonction de valeur à l'étape*** n. *Il faut maintenant chercher la meilleure action, ce qui revient à chercher, pour chaque état de croyance, le meilleur vecteur pour le représenter. Nous avons ici combiné les fonctions de valeur de deux actions* ($a1$ et $a2$)*, ce qui permet en outre d'éliminer des vecteurs inutiles (indiqués en pointillés). Pour chacune des régions déterminées, nous avons indiqué le début de la politique optimale d'horizon* n.

A cause de l'inégalité stricte, les régions ne définissent pas exactement une partition de $\mathcal{B}$ et les points exclus sont des points où plusieurs vecteurs donnent une même valeur à la fonction de valeur. Ces points sont assez particuliers et peuvent poser problème, comme nous le verrons à la partie 3.4.2.5.

La notion de région est très importante. Elle est utilisée par de nombreux algorithmes pour calculer la fonction de valeur. L'algorithme `TrouveVectDansRegion` (Alg. 14) décrit la procédure qui détermine si la région d'un vecteur est vide (c'est alors un vecteur inutile) et, dans le cas contraire, retourne un vecteur particulier de cette région. Il faut pour cela utiliser une méthode de programmation linéaire pour optimiser une fonction sous contraintes qui est définie par la méthode `PrepareProgLin` explicitée dans l'algorithme 15.

Algorithme 3.1 : TrouveVectDansRegion(θ, Θ)

Entrées : Une représentation Θ, un vecteur $\theta \in \Theta$
Sorties : Un point de cette région ou **null**

LP $\leftarrow$ `PrepareProgLin` (θ, Θ)
`ResoudProgLin` (LP)
si `SansSolution` *(*LP*)* **alors**
  retourner null
si *value(*LP*)* ≤ 0 **alors**
  retourner null
retourner `Solution` *(*LP*)*

Algorithme 3.2 : PrepareProgLin(θ, Θ)

Entrées : Une représentation Θ, un vecteur $\theta \in \Theta$
Sorties : Un problème de programmation linéaire
résoudre
  $\max_{\mathbb{R}} \epsilon$
  avec
    $x.(\theta - \tilde{\theta}) \geq \epsilon,\ \forall \tilde{\theta} \in \Theta,\ \tilde{\theta} \neq \theta$
      $x \in \Pi(\mathcal{S})$

3.4.2.2. *Représentation parcimonieuse*

Dans un ensemble Θ quelconque, certains vecteurs ne sont pas nécessaires à la représentation de la fonction de valeur. C'est le cas des vecteurs dominés.

DÉFINITION.– *Soit Θ représentant une fonction de valeur. Un vecteur θ de Θ est dominé si $\forall b \in \mathcal{B}$ on a :* $b.\theta \leq \max_{\theta' \in \Theta} b.\theta'$.

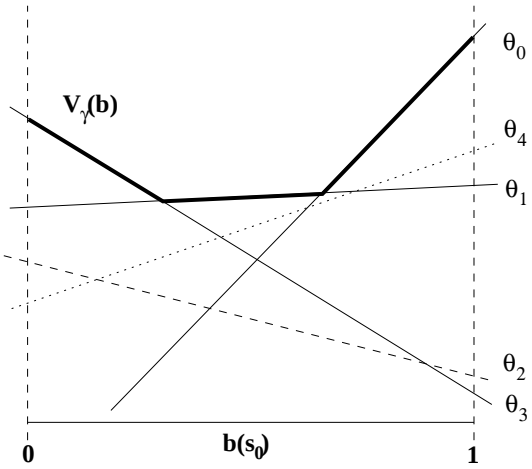


Figure 3.11. ***Représentation parcimonieuse de*** V. *Tous les vecteurs de* Θ *ne sont pas utiles pour représenter* V. *Le vecteur* θ_2, *qui est entièrement dominé par* θ_1, *sera enlevé par la procédure* `VerifDomination`. *Quand au vecteur* θ_4, *il faut la procédure plus complexe* `Elagage` *pour l'éliminer.*

Si θ est dominé, on montre trivialement que Θ et $\Theta - \{\theta\}$ représentent la même fonction de valeur : un vecteur dominé peut être enlevé de Θ sans danger. En fait, on peut montrer (voir par exemple [LIT 96]) qu'une fonction de valeur LPMC possède une unique représentation minimale où aucun vecteur n'est dominé. On dit que cette représentation est *parcimonieuse*. Sur l'exemple de la figure 3.11 on voit bien que les vecteurs θ_2 et θ_4 ne sont pas utiles pour représenter V. Dans ce cas, la représentation parcimonieuse est $\Theta = \{\theta_0, \theta_1, \theta_3\}$.

DÉFINITION.– *Une représentation* Θ *parcimonieuse d'une fonction de valeur LPMC est telle que, pour tout* $\theta \in \Theta$, *la région* $R(\theta, \Theta)$ *n'est pas vide.*

Il reste maintenant à construire cette représentation parcimonieuse à partir d'une représentation donnée.

3.4.2.3. *Elimination des vecteurs dominés*

La *Recherche de Domination Simple* est une procédure très simple pour éliminer des vecteurs inutiles d'une représentation Θ. On recherche les vecteurs qui sont entièrement dominés par un seul autre vecteur. Ce sont des vecteurs θ tels qu'il existe un autre vecteur $\tilde{\theta} \in \Theta$ tel que $\forall s' \in \mathcal{S}$, $\theta(s) \leq \tilde{\theta}(s')$. Bien que cette procédure ne garantisse pas de diminuer Θ ou d'éliminer tous les vecteurs inutiles (voir figure 3.11), elle est très efficace d'un point de vue computationnel.

L'algorithme `VerifDomination` (Alg. 16) détaille la procédure qui permet de nettoyer une représentation des vecteurs entièrement dominés. On y utilise la procédure `EnleveElement` qui retire un élément d'un ensemble.

Algorithme 3.3 : VerifDomination(Θ)

Entrées : Une représentation Θ
Sorties : Une représentation sans vecteur dominé

si $|\Theta| < 2$ **alors**
  retourner Θ
$\tilde{\Theta} \leftarrow \emptyset$
répéter
  $\theta \leftarrow$ `EnleveElement`(Θ)
  si $\nexists \theta' \in \tilde{\Theta}$ *t.q.* $\theta' \geq \theta$ **alors**
    $\tilde{\Theta} \leftarrow \{\theta' | \theta' \in \tilde{\Theta}, .\theta \not\geq \theta'\}$
    $\tilde{\Theta} \leftarrow \tilde{\Theta} \cup \{\theta\}$
jusqu'à $\Theta = \emptyset$
retourner $\tilde{\Theta}$

3.4.2.4. *Elagage*

La méthode la plus simple pour rendre une représentation parcimonieuse consiste à regarder, pour chacun de ses vecteurs, si la région associée est vide. C'est le principe de l'algorithme proposé par Monahan [MON 82], mais c'est une méthode plutôt inefficace. Il existe une méthode d'élagage beaucoup plus efficace proposée par Lark et White [WHI 91] et détaillée dans l'algorithme `Elagage` (Alg. 17).

Le principe de `Elagage` est de construire incrémentalement la représentation parcimonieuse en ayant, à tout moment, un sous-ensemble $\hat{\Theta} \in \Theta$ de cette représentation parcimonieuse. Prenant dans $\tilde{\Theta}$ un nouveau vecteur candidat θ, le fait de chercher si sa région associée *dans* $\hat{\Theta}$ n'est pas vide est rapide (car $\hat{\Theta}$ est petit), par contre une réponse positive n'assure pas que c'est un vecteur dominant (on ne connaît pas encore la vraie représentation) mais simplement que $\hat{\Theta}$ n'est pas encore complète. Il faut encore y ajouter au moins un vecteur dominant et on utilise pour cela la routine `MeilleurVecteur` alors que θ a été remis dans $\tilde{\Theta}$. La routine `MeilleurVecteur` recherche, pour l'état de croyance retourné par `TrouveVectDansRegion`, le meilleur vecteur restant dans $\hat{\Theta}$ qui sera alors ajouté à $\hat{\Theta}$. Il y a une petite subtilité dans le choix de ce meilleur vecteur, comme nous allons le voir maintenant.

3.4.2.5. *Choix d'un vecteur en un point*

Les diverses versions de l'équation (3.17) que nous avons explicitées dans la section 3.4.1 détaillant une application de l'opérateur de la programmation dynamique permettent de calculer le vecteur dominant en un point b quelconque de l'espace des

Algorithme 3.4 : Elagage($\tilde{\Theta}$)

Entrées : Une représentation $\tilde{\Theta}$ de V
Sorties : Une représentation parcimonieuse Θ de V

$\hat{\Theta} \leftarrow \emptyset$
tant que $\tilde{\Theta} \neq \emptyset$ **faire**
  $\theta \leftarrow$ `EnleveElement`$(\tilde{(\Theta)})$
  $b \leftarrow$ `TrouveVectDansRegion`$(\theta, \hat{\Theta})$
  si $b \neq$ **null alors**
    $\tilde{\Theta} \leftarrow \tilde{\Theta} \cup \{\theta\}$
    $\theta^* \leftarrow$ `MeilleurVecteur`$(\tilde{\Theta}, b)$
    $\tilde{\Theta} \leftarrow \tilde{\Theta} - \{\theta\}$
    $\hat{\Theta} \leftarrow \hat{\Theta} \cup \{\theta^*\}$
$\Theta \leftarrow \hat{\Theta}$
retourner Θ

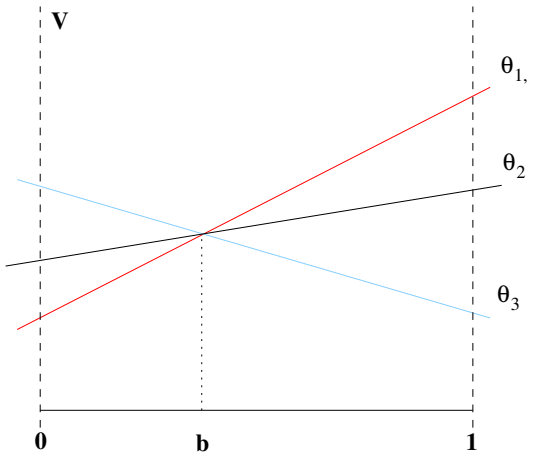


Figure 3.12. ***Vecteur dominant****. Quel est* le *vecteur dominant au point I ?*

états de croyance. Il subsiste un problème potentiel quand plusieurs vecteurs sont candidats en un point (illustré par la figure 3.12) : lequel est *le* vecteur dominant en un point I donné ?

Il faudrait en fait pouvoir tester la validité de ces vecteurs dans un voisinage de b, ce qui n'est pas simple à mettre en place de manière exacte. Une solution plus subtile consiste à doter les vecteurs d'un ordre lexicographique. Pour cela il faut doter les états d'un ordre fixe et arbitraire sur $\mathcal{S}$, on le note $s \prec s'$.

DÉFINITION.– *Le vecteur θ est lexicographiquement plus grand que θ' (noté $\theta \overset{L}{>} \theta'$) s'il existe un état s tel que $\theta(s) > \theta'(s)$ et $\theta(s') = \theta'(s')$ pour tout $s' \prec s$.*

Alors, sans entrer dans les détails de la démonstration, Littman a prouvé (théorème 3.2) que, en cas de doute, le vecteur maximal au sens de cet ordre lexicographique sera bien un vecteur qui fait partie de la représentation parcimonieuse de la fonction de valeur. En effet, la région de ce vecteur sera non vide, et c'est bien sur le vecteur qui est retourné par l'algorithme `MeilleurVecteur` (Alg. 19), en s'aidant de la procédure `MaximumLexicographique` (Alg. 18) qui retourne le maximum lexicographique de deux vecteurs.

THÉORÈME 3.2.– *Soient Θ une fonction de valeur, b un point de l'espace des états de croyance et Λ l'ensemble des vecteurs donnant une valeur maximum de la fonction de valeur en b ($\Lambda = \{\text{argmax}_{\theta \in \Theta}\, b.\theta\}$). Alors, s'il existe un $\lambda^* \subset \Lambda$ tel que $\lambda^* \overset{L}{>} \lambda$ pour tous les autres $\lambda \in \Lambda$, alors $R(\lambda^*, \Theta)$ est non-vide. ([LIT 96])*

Algorithme 3.5 : MaximumLexicographique(θ, $\tilde{\theta}$)

Entrées : Deux vecteurs θ et $\tilde{\theta}$ de Θ
Sorties : Le maximum lexicographique des deux vecteurs

pour chaque $s \in \mathcal{S}$ **faire**
  si $\theta(s) > \tilde{\theta}(s)$ **alors**
    retourner θ
  si $\theta(s) < \tilde{\theta}(s)$ **alors**
    retourner $\tilde{\theta}$
retourner θ

Algorithme 3.6 : MeilleurVecteur(Θ, b)

Entrées : Une représentation Θ, un état de croyance b
Sorties : Le meilleur vecteur de Θ pour cet état

$v^* \leftarrow -\infty$
pour chaque $\theta \in \Theta$ **faire**
  $v \leftarrow b.\theta$
  si $v = v^*$ **alors**
    $v^* \leftarrow$ `MaximumLexicographique`(θ^*, θ)
  si $v > v^*$ **alors**
    $v^* \leftarrow v$
    $\theta^* \leftarrow \theta$
retourner θ^*

3.4.3. *L'algorithme* WITNESS

L'algorithme WITNESS a été proposé par Cassandra, Littman et Kaelbling [CAS 94] en 1994. Il a ensuite été étudié plus formellement afin de prouver son optimalité [LIT 96]. Pour chaque action a de $\mathcal{A}$, l'algorithme calcule une représentation parcimonieuse de Θ_n^a à partir de Θ_{n-1} en explorant un nombre fini de régions de $\mathcal{B}$. C'est dans la façon de choisir ces régions que réside toute l'intelligence de l'algorithme, comme nous allons le voir. Il est à noter que d'autres méthodes s'appuient aussi sur l'exploration de régions, mais pour construire Θ_n directement (voir [SON 71, SMA 73, CHE 88]).

3.4.3.1. *Voisinage d'un vecteur*

La notion du *voisinage* d'un vecteur θ_n^a de V_n^a est cruciale pour l'algorithme WITNESS puisque c'est en vérifiant les voisins d'un vecteur que l'on peut savoir si la construction courante de la représentation parcimonieuse de V_n^a est complète ou pas.

On peut réécrire l'équation (3.18) sous la forme :

$$\theta_n^{a,o} = \frac{r(a)}{|\Omega|} + \gamma P^{a,o}\theta_{n-1}^{a,o}(b^{a,o}), \tag{3.22}$$

où $\theta_{n-1}^{a,o}(b^{a,o})$ est *le* vecteur de Θ_{n-1} qui est le meilleur pour l'état de croyance $b^{a,o}$. Mais si on enlève les références aux états de croyance, on peut aussi construire toute une famille de vecteurs

$$\widetilde{\theta^{a,o}} = \frac{r(a)}{|\Omega|} + \gamma P^{a,o}\theta_{n-1}, \tag{3.23}$$

où θ_{n-1} est simplement *un* vecteur de Θ_{n-1}. On construit ainsi l'ensemble $\widetilde{\Theta}_n^{a,o}$ et, par combinaison, l'ensemble $\widetilde{\Theta}^a$ qui contient la représentation parcimonieuse Θ_n^a de V_n^a. Il y $|\Theta_{n-1}|^{|\Omega|}$ vecteurs possibles dans $|\widetilde{\Theta}_n^a|$. On peut définir la notion de voisinage pour ces vecteurs comme suit :

DÉFINITION 3.5.– *Un vecteur ν de $\widetilde{\Theta}_n^a$ est un voisin du vecteur $\theta_n^a = \sum_{o\in\Omega}\theta_n^{a,o}$ (qui lui est aussi dans Θ_n^a) si*

$$\nu = \tilde{\theta}_n^{a,o'} + \sum_{o\neq o'}\theta_n^{a,o},$$

où $o' \in \Omega$, $\tilde{\theta}_n^{a,o'} \in \widetilde{\Theta}_n^{a,o}$ et $\tilde{\theta}_n^{a,o'} \neq \theta_n^{a,o'}$.

Un vecteur $\tilde{\theta}$ possède $|O|(|\Theta_{n-1}| - 1)$ voisins, on note $\mathcal{N}(\tilde{\theta})$ l'ensemble de ses voisins. Tout l'intérêt des voisins vient du théorème suivant qui dit que, s'il existe un point de $\mathcal{B}$ où il existe un meilleur vecteur, alors c'est vrai aussi pour un des voisins de ce vecteur (voir [CAS 98]).

THÉORÈME 3.3.– *Pour tout $\tilde{\theta}_n^a \in \widetilde{\Theta}_n^a$, il existe un $b \in \mathcal{B}$ et un ${\tilde{\theta'}}_n^a \in \widetilde{\Theta}_n^a$ tels que $b.{\tilde{\theta'}}_n^a > b.\tilde{\theta}_n^a$ si et seulement s'il existe un voisin $\nu \in \mathcal{N}(\tilde{\theta}_n^a)$ tel que $b.\nu > b.\tilde{\theta}_n^a$.*

PREUVE.– Nous allons procéder en deux temps.

– Prouvons que $b.\nu > b.\tilde{\theta}_n^a \implies b.{\tilde{\theta'}}_n^a > b.\tilde{\theta}_n^a$.
Comme $\nu \in \widetilde{\Theta}_n^a$, c'est évident.

– Prouvons que $b.{\tilde{\theta'}}_n^a > b.\tilde{\theta}_n^a \implies b.\nu > b.\tilde{\theta}_n^a$.
Nous allons prouver que ν existe en le construisant. Nous avons

$$\begin{aligned} b.{\tilde{\theta'}}_n^a &> b.\tilde{\theta}_n^a \\ \sum_o b.{\tilde{\theta'}}_n^{a,o} &> \sum_o b.\tilde{\theta}_n^{a,o}. \end{aligned}$$

Comme la première somme est plus grande que la deuxième, il existe forcément une observation o' telle que

$$b.{\tilde{\theta'}}_n^{a,o'} > b.\tilde{\theta}_n^{a,o'}.$$

Nous allons nous servir de ce fait pour construire ν

$$\begin{aligned} \sum_{o \neq o'} b.\tilde{\theta}_n^{a,o} &= \sum_{o \neq o'} b.\tilde{\theta}_n^{a,o} \\ b.{\tilde{\theta'}}_n^{a,o'} + \sum_{o \neq o'} b.\tilde{\theta}_n^{a,o} &> b.\tilde{\theta}_n^{a,o'} + \sum_{o \neq o'} b.\tilde{\theta}_n^{a,o} \\ b.\left({\tilde{\theta'}}_n^{a,o'} + \sum_{o \neq o'} \tilde{\theta}_n^{a,o}\right) &> b.\tilde{\theta}_n^a. \end{aligned}$$

Il suffit de poser $\nu = {\tilde{\theta'}}_n^{a,o'} + \sum_{o \neq o'} \tilde{\theta}_n^{a,o}$ pour terminer la démonstration de cette implication et donc du théorème. □

3.4.3.2. *L'algorithme*

L'algorithme WITNESS construit progressivement une représentation parcimonieuse Θ_n^a d'une fonction de valeur V_n^a pour une action a fixée. Comme décrit dans l'algorithme `Witness` (Alg. 20), l'ensemble $\hat{\Theta}$ grossit en stockant petit à petit les vecteurs qui composent Θ, de sorte que l'on a toujours $\hat{V}(b) \leq V(b)$.

Pour ce faire, l'algorithme choisit d'abord un vecteur b de $\mathcal{B}$ et cherche le meilleur vecteur de la région à laquelle appartient ce vecteur, ainsi que tous les voisins de

ce meilleur vecteur. Ces voisins forment l'ensemble Υ qui servira, en quelque sorte, d'agenda. Un par un, les vecteurs v de l'agenda sont examinés pour :

– soit l'enlever de l'agenda si la région définie par v est vide (car v est alors un vecteur inutile pour V) ;

– soit ajouter le meilleur vecteur de la région définie par v aux vecteurs définissant V et mettre tous les voisins de ce vecteur dans l'agenda. Il est aussi important de remettre v dans l'agenda car, bien que v ne soit pas maximal pour l'instant, nous n'avons pas encore la certitude qu'il est inutile.

Algorithme 3.7 : Witness(Θ_{n-1}, a)

Entrées : Une représentation parcimonieuse Θ_{n-1} de V^*_{n-1}, une action a
Sorties : Une représentation parcimonieuse de $V^{*,a}_n$

$b \leftarrow$ un état de croyance de $\mathcal{B}$
$\hat{\Theta} \leftarrow \{\theta^a_n(b)\}$
$\Upsilon \leftarrow \mathcal{N}(\theta^a_n(b))$
tant que $\Upsilon \neq \emptyset$ **faire**
  $v \leftarrow$ `EnleveElement`(Υ)
  si $v \in \hat{\Theta}$ **alors**
    $b \leftarrow$ **null**
  sinon
    $b \leftarrow$ `TrouveVectDansRegion`($v, \hat{\Theta}$)
  si $b \neq$ **null alors**
    $\hat{\Theta} \leftarrow \hat{\Theta} \cup \{\theta^a_n(b)\}$
    $\Upsilon \leftarrow \Upsilon \cup \{v\}$
    $\Upsilon \leftarrow \Upsilon \cup \mathcal{N}(\theta^a_n(b))$
$\Theta^a_n \leftarrow \hat{\Theta}$
retourner Θ^a_n

La validité de l'algorithme repose sur le théorème 3.3. En essence, quels que soient les vecteurs actuellement présents dans $\hat{\Theta}$, s'il y a un vecteur de $\bar{\Theta}^a_n$ qui serait meilleur en un point b, alors un des voisins v du vecteur $\hat{\theta}$ de $\hat{\Theta}$ qui est *actuellement* le meilleur en b donne aussi une meilleure valeur en ce point b. Comme l'algorithme vérifie tous les voisins des vecteurs de $\hat{\Theta}$, nous sommes sûrs de ne manquer aucun vecteur de Θ^a_n. La preuve formelle est un peu plus complexe.

L'efficacité de l'algorithme peut être améliorée de plusieurs manières. En particulier, [CAS 98] mentionne le fait de choisir avec pertinence les vecteurs de la représentation initiale $\hat{\Theta}$ ou d'éviter de tester plusieurs fois un vecteur v de Υ ou encore de vérifier l'utilité des voisins d'un vecteur *avant* de les ajouter à Υ. Il n'en reste pas moins que la portée pratique de cet algorithme est limitée car, pour des raisons de

mémoire et de temps de calcul, on ne peut appliquer plus de quelques itérations (de l'ordre de 4 ou 5) à des problèmes avec une poignée d'états.

3.4.4. *Elagage itératif (Iterative pruning)*

L'algorithme d'ÉLAGAGE ITÉRATIF est un peu plus efficace que l'algorithme WITNESS. Nous présentons tout d'abord un algorithme d'élagage très simple, puisqu'il parcourt tous les vecteurs pour élaguer ceux qui sont dominés.

3.4.4.1. *Enumération complète*

Pour trouver une représentation parcimonieuse de Θ_n, il est possible d'énumérer tous les vecteurs possibles de cet ensemble et de les élaguer ensuite. C'est la méthode proposée par Monahan [MON 82] et nous allons la détailler un peu pour comprendre le fonctionnement de l'élagage itératif.

Posons $\overline{\Theta}_n^{a,o} = \left\{ \frac{r(a)}{|\Omega|} + \gamma \sum_{s' \in \mathcal{S}} p(s,a,s') O(s',o) \theta_{n-1}^{a,o}(b^{a,o}) \right\}_{(a,o)}$.

C'est l'ensemble de tous les vecteurs possibles de $\Theta_n^{a,o}$. Si on cherche toutes les combinaisons possibles de ces vecteurs (en s'inspirant de l'équation (3.19)), on obtient un nouvel ensemble $\overline{\Theta}_n^a$ qui contient Θ_n^a. Nous appellerons cette opération la *somme croisée* et nous noterons

$$\overline{\Theta}_n^a = \bigoplus_o \overline{\Theta}_n^{a,o}.$$

Ainsi, l'ensemble complet des vecteurs générant V_n s'écrit

$$\overline{\Theta}_n = \bigcup_a \overline{\Theta}_n^a.$$

Et il ne reste plus qu'à élaguer cet ensemble pour obtenir une représentation parcimonieuse

$$\Theta_n = \text{ELAGAGE}\left(\bigcup_a \overline{\Theta}_n^a \right).$$

L'inconvénient de cette méthode est que sa complexité est exponentielle en la taille de Ω. L'idée est alors d'effectuer l'élagage de manière incrémentale.

3.4.4.2. *Enumération incrémentale*

Comme WITNESS, l'algorithme d'élagage incrémental cherche d'abord des représentations parcimonieuses de Θ_n^a pour ensuite les combiner et trouver Θ_n. Comme $\Theta_n^a = \text{ELAGAGE}(\bar{\Theta}_n^a)$, nous avons

$$\Theta_n^a = \text{ELAGAGE}\left(\bigoplus_o \overline{\Theta}_n^{a,o}\right).$$

Puisque la procédure ELAGAGE cherche en fait les vecteurs maximaux, il est facile de montrer que ces derniers sont en fait obtenus par combinaisons de vecteurs eux-mêmes maximaux, ce qui nous permet d'écrire que :

$$\begin{aligned}\Theta_n^a &= \text{ELAGAGE}\left(\bigoplus_o \text{ELAGAGE}(\overline{\Theta}_n^{a,o}))\right) \\ &= \text{ELAGAGE}\left(\bigoplus_o \Theta_n^{a,o}\right).\end{aligned}$$

Il faut conserver l'opérateur ELAGAGE à l'extérieur de la somme croisée car, si tous les vecteurs maximaux sont des combinaisons de vecteurs maximaux, certaines de ces combinaisons sont tout de même inutiles.

En continuant, on obtient :

$$\begin{aligned}\Theta_n^a &= \text{ELAGAGE}\left(\bigoplus_o \Theta_n^{a,o}\right) \\ &= \text{ELAGAGE}\left(\Theta_n^{a,0} \oplus \Theta_n^{a,1} \oplus \Theta_n^{a,2} \oplus \cdots \oplus \Theta_n^{a,|\Omega|-1}\right) \\ &= \text{ELAGAGE}\left(\cdots \text{ELAGAGE}(\text{ELAGAGE}(\Theta_n^{a,0} \oplus \Theta_n^{a,1}) \oplus \Theta_n^{a,2}) \cdots \oplus \Theta_n^{a,|\Omega|-1}\right).\end{aligned}$$

Cette équation résume donc l'algorithme d'élagage incrémental. L'idée, comme le montre la figure 3.13, est d'élaguer les combinaisons de deux ensembles de vecteurs $\Theta_n^{a,0}$ et $\Theta_n^{a,1}$ (représentés par leur partition en régions) pour obtenir un ensemble élagué intermédiaire. Les vecteurs de ce nouvel ensemble sont ensuite combinés avec ceux de $\Theta_n^{a,2}$ et ainsi de suite.

Cette méthode est détaillée par l'algorithme de la table 21. L'ensemble Ψ sert à stocker tous les vecteurs de $\Theta_n^{a,o}$ et les résultats intermédiaires obtenus par élagage de deux de ces vecteurs.

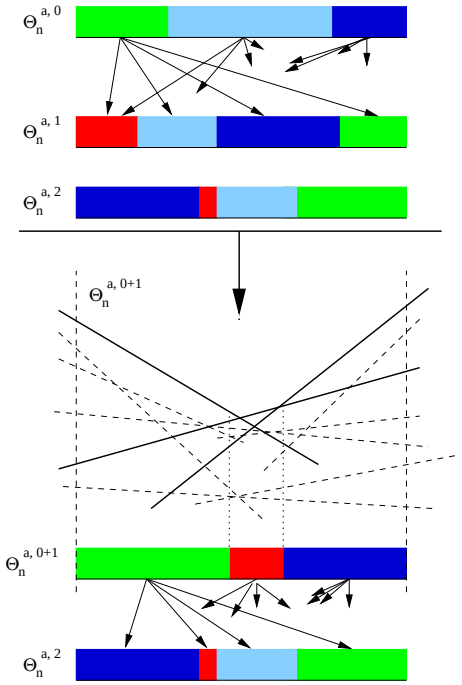


Figure 3.13. ***Elagage incrémental de*** Θ_n^a *Pour construire une représentation parcimonieuse de* Θ_n^a*, l'élagage incrémental part de deux ensembles* $\Theta_n^{a,0}$ *et* $\Theta_n^{a,1}$*. L'ensemble formé par toutes les combinaisons de vecteurs issus de ces deux ensembles est ensuite élagué pour former l'ensemble parcimonieux* $\Theta_n^{a,1+2}$*. C'est ensuite* cet *ensemble qui est combiné avec* $\Theta_n^{a,2}$ *pour créer* $\Theta_n^{a,0+1+2}$ *et ainsi de suite pour obtenir* Θ_n^a*.*

Algorithme 3.8 : ElagageIncremental(Θ_{n-1}, a)

Entrées : Une représentation parcimonieuse Θ_{n-1} de V_{n-1}^*, une action a
Sorties : Une représentation parcimonieuse de $V_n^{*,a}$

$\Psi \leftarrow \bigcup_o \{\Theta_n^{a,o}\}$
tant que $|\Psi| > 1$ **faire**
  $A \leftarrow$ `EnleveElement`(Ψ)
  $B \leftarrow$ `EnleveElement`(Ψ)
  $D \leftarrow$ ELAGAGE$(A \oplus B)$
  $\Psi \leftarrow \Psi \cup \{D\}$
retourner Ψ

3.5. Algorithmes d'itération sur les politiques

A l'instar de l'algorithme d'itération sur les politiques (cf. algorithme 1.5, page 44), il est possible de chercher une solution optimale à un POMDP directement dans l'espace des politiques. Reste alors à bien préciser dans quel espace se fait cette recherche.

Si on se place dans l'espace des politiques définies sur les états de croyance ($\pi : \mathcal{B} \rightarrow \mathcal{A}$), on se trouve dans un espace infini continu où il faut chercher un maximum absolu. L'algorithme proposé par Sondik (voir [SON 78]) permet, en théorie, de trouver une solution exacte pour les POMDP transitoires et une solution ϵ-optimale sinon. Par contre, son application pratique est limitée à des cas très simples au vu de la complexité de chaque itération.

Une alternative intéressante, proposée par Hansen [HAN 98b], s'appuie sur le fait que les politiques des POMDP transitoires peuvent être représentées par des automates avec un nombre *fini* d'état, comme celui de la figure 3.3 page 94. Cette représentation permet de lever la principale difficulté de l'algorithme de Sondik car ce dernier, qui s'appuie sur un découpage de $\mathcal{B}$ en région, doit transformer cette représentation en un contrôleur à états finis équivalent, ce qui est extrêmement coûteux. Hansen choisit de travailler directement avec des contrôleurs à états finis.

Le cœur de l'algorithme repose sur une mise à jour du contrôleur à l'aide de l'opérateur de la programmation dynamique, ce qui permet d'améliorer les performances de la politique.

Sondik a montré que la fonction de valeur d'une politique exprimée comme un contrôleur à états finis est linéaire par morceaux [SON 78]. Soit un tel contrôleur δ, appelons V^δ sa fonction de valeur, qui peut donc être décrite par un ensemble de vecteurs $\{\theta^i\}$, avec exactement un vecteur par nœud i du contrôleur. A chaque vecteur θ^i, lié au nœud i, on peut associer une action $a(i)$ et une transition vers le nœud $l(i,o)$ (ou le vecteur $\theta^{l(i,o)}$) pour chaque observation o. Cette fonction de valeur vérifie l'équation suivante pour chaque nœud i du contrôleur et chaque état s du POMDP :

$$\theta^i(s) \quad = \quad r(s,a(i)) + \gamma \sum_{s',o} \Pr(s'|s,a(i)) \Pr(o|s') \theta^{l(i,o)}. \tag{3.24}$$

En appliquant l'opérateur de la programmation dynamique (cf. section 3.4.1) sur les vecteurs de V^δ, on obtient une nouvelle fonction de valeur $\hat{V}^\delta$. Il est facile de montrer que chaque vecteur $\hat{\theta}^j$ de $\hat{V}^\delta$ est associé à une action $a(j)$ et, pour chaque observation o, à une transition $\hat{l}(j,o)$ vers un vecteur $\theta^{\hat{l}(j,o)}$ de V^δ. Ces vecteurs $\hat{\theta}^j$ de $\hat{V}^\delta$ peuvent être des copies de vecteurs de δ (même action et mêmes liens). Ils peuvent aussi être de nouveaux vecteurs et vont permettre de modifier V^δ (et donc le contrôleur), de la manière suivante :

– si $\hat{\theta}^j$ domine un vecteur θ^i de V^δ, on associe au nœud i l'action et les transitions du nœud j ;

– sinon, on ajoute le nœud j à δ.

Il faut enfin enlever de δ tous les nœuds qui n'ont pas de vecteur associé dans $\hat{V}^\delta$ *mais* qui ne peuvent être atteints par un des autres nœuds de δ auquel est associé un vecteur de $\hat{V}^\delta$. Cette série d'opérations permet de définir un nouveau contrôleur $\hat{\delta}$. C'est le coeur de l'algorithme de Hansen (voir Alg. 22). Dans sa thèse [HAN 98a], Hansen prouve le théorème (voir Théorème 3.4) qui assure qu'une itération améliore la politique, ce qui garantit la convergence vers une solution ϵ-optimale après un nombre fini d'itérations (ou, dans le cas des POMDP transitoires, vers la solution optimale).

Algorithme 3.9 : PolicyIteration(δ, ϵ)

Entrées : Un contrôleur à état fini δ et un réel positif ϵ
Sorties : Un contrôleur à état fini δ^* qui est ϵ-optimal

répéter
  Calculer V^δ à partir de δ en résolvant les équations (3.24)
  Construire $\hat{V}^\delta \leftarrow$ `OpProgrammationDynamique`(V^δ)
  $\hat{\delta} \leftarrow \emptyset$
  pour chaque $\hat{\theta^j} \in \hat{V}^\delta$ **faire**
    si *il existe un nœud i de δ associé à $\hat{\theta^j}$ avec action et liens identiques*
    alors
      ajouter i à $\hat{\delta}$
    sinon si *il existe nœud i tq $\hat{\theta}^j$ domine θ^i* **alors**
      ajouter i a $\hat{\delta}$, avec l'action et les liens de $\hat{\theta}^j$
    sinon
      ajouter un *nouveau* nœud à $\hat{\delta}$ avec action et liens de $\hat{\theta}^j$
  Ajoute à $\hat{\delta}$ tous les autres nœuds de δ qui sont atteignables depuis $\hat{\delta}$
  $\delta \leftarrow \hat{\delta}$
jusqu'à $\|\hat{V}^\delta - V^\delta\| \leq \epsilon(1-\gamma)/\gamma$
retourner $\hat{\delta}$

THÉORÈME 3.4.– *Si un contrôleur à états fini δ n'est pas optimal, une itération de l'algorithme* `PolicyIteration` *le transforme en un contrôleur $\hat{\delta}$ dont la fonction de valeur est au moins aussi bonne pour tous les états de croyance et meilleure pour quelques états de croyance.*

3.6. Conclusion et Perspectives

Les Processus Décisionnels de Markov Partiellement Observables (POMDP) permettent de modéliser et contrôler les systèmes dynamiques incertains dont l'état n'est que partiellement connu. Le contrôleur n'a pas accès à l'état du processus mais doit se contenter d'observations imparfaites de cet état. En général, il n'est pas possible de contrôler optimalement un POMDP en utilisant uniquement l'observation courante du processus. Il faut en effet avoir accumulé suffisamment d'information, comme par exemple l'historique de toutes les observations passées, pour définir une politique optimale. Ainsi, les méthodes classiques s'appuient sur les états de croyance qui sont des résumés suffisants de l'information contenue dans les historiques d'observations. En fait, les méthodes de la programmation dynamique (itération sur les valeurs et itération sur les politiques) peuvent être appliquées sur ces états de croyance.

En pratique, les algorithmes classiques comme WITNESS ou ELAGAGE INCRÉMENTAL utilisent le fait que la fonction de valeur est linéaire par morceau et cherchent à la représenter efficacement. Cependant, ces algorithmes exacts ne peuvent être appliqués à des problèmes comportant plus d'une dizaine d'états à cause de l'accroissement potentiellement exponentiel du nombre d'éléments nécessaires pour représenter la fonction de valeur. Il en va de même d'autres algorithmes classiques que nous n'avons pas détaillés ici, comme par exemple les algorithmes de Monahan [MON 82] ou Cheng [CHE 88]. Une autre alternative est de s'intéresser à la factorisation des états et des observations, comme le fait [GUE 01b]. Il se peut aussi que des travaux beaucoup plus récents (voir [KOL 94, ARA 07]), s'appuyant sur la programmation linéaire pour représenter une politique par l'ensemble des trajectoires qu'elle peut générer, puissent mener à des algorithmes permettant de résoudre de manière exacte des problèmes plus complexes. Pour l'instant, les gains en complexités sont relativement restreints.

Ces travaux théoriques à l'aspect pratique limité ont inspiré des algorithmes pour trouver des solutions approchées aux POMDP. Il est possible de ne chercher la fonction de valeur que pour certains point de l'espace des états de croyance en espérant que cette fonction de valeur approchée sera proche de l'optimale [PIN 03, SPA 05]. D'autres optent pour une recherche « avant » à partir d'un unique état estimé de départ pour ne calculer la fonction de valeur que sur les états de croyance atteignables (voir par exemple [BON 00]). D'autres travaux combinent la programmation dynamique avec une recherche heuristique pour, à chaque étape de l'itération sur la fonction de valeur, restreindre fortement l'ensemble des états estimé sur lesquels la fonction valeur est calculée. Il est ainsi possible de s'attaquer à des problèmes beaucoup plus complexes [SEU 07].

Le problème le plus crucial avec les POMDP reste celui de l'apprentissage. Les méthodes d'apprentissage indirectes (apprendre d'abord un modèle avant de planifier) ne sont pas très performantes car il est difficile d'apprendre la structure cachée

d'un POMDP. Actuellement, il faut utiliser des méthodes d'apprentissage qui cherche des solutions approchées. Les travaux de [MCC 95, DUT 00] font l'hypothèse que le POMDP peut s'appréhender comme un MDP d'ordre k. D'autres travaux apprennent directement dans l'espace des politiques paramétriques en utilisant des montées de gradient pour en trouver les meilleurs paramètres [BAX 00].

Enfin, il faut mentionner les travaux récents autour de l'utilisation des *représentations par états prédictifs*[5]. [LIT 02] ayant montré qu'on peut efficacement représenter un POMDP avec ces états prédictifs, qui sont en fait les probabilités que certaines trajectoires se réalisent dans le futur, des algorithmes d'apprentissage commencent à voir le jour, aussi bien pour découvrir ces états prédictifs que pour apprendre leurs probabilités de réalisation [SIN 03, ABE 07b].

5. de l'anglais « Predictive State Representation (PSR) »

Chapitre 4

Une introduction aux jeux stochastiques

4.1. Introduction

La théorie des jeux [AUM 02] est un formalisme qui vise à étudier les interactions entre agents sachant que de telles interactions peuvent s'étendre de la coopération au conflit. Originellement conçue comme un outil mathématique pour les sciences économiques, il a montré au travers des années son utilité en logique et théorie des ensembles [GRÄ 02b, MAR 75], en biologie et théorie de l'évolution [SMI 02], en informatique [PAP 95, PAR 02, GIE 06], en analyse des conflits [MYE 97] etc.

L'interaction est la pierre angulaire des études sous-tendus par la théorie des jeux. Pour modéliser les interactions, il faut tout d'abord étendre la notion de théorie des jeux à des *jeux dynamiques*, généralement utilisés pour rendre compte de la compétition entre processus évoluant dans le temps. Des transitions stochastiques sont ensuite utilisées pour formaliser l'incertain. Les *jeux stochastiques (ou markoviens)* sont des jeux dynamiques avec des transitions stochastiques. Ils forment, de nos jours, une théorie mathématique mature et riche, utilisée dans beaucoup d'applications comme l'économie, la biologie et tout ce qui tourne autour de l'évolution et des populations, les files d'attentes, les télécommunications, le *model checking* etc. Ces derniers temps, les jeux stochastiques ont pris beaucoup d'importance en informatique aux travers plus particulièrement des systèmes multiagents spécialement pour la décision, la planification et l'apprentissage dans un environnement où interviennent plusieurs agents autonomes [STO 00].

Chapitre rédigé par Andriy BURKOV et Brahim CHAIB-DRAA.

Dans ce chapitre, nous considérons les jeux stochastiques comme étant le modèle multiagent le plus général. Nous présentons de façon détaillée plusieurs algorithmes pour résoudre les problèmes qui peuvent être modélisés par des jeux stochastiques, même si la plupart de ces algorithmes peuvent aussi résoudre des modèles plus simples.

Le reste du chapitre est organisé comme suit. La section 4.2 présente les principales notions de la théorie des jeux classique, telles que le jeu en forme stratégique, le jeu dynamique, l'équilibre de Nash et autres. Dans la section 4.3, le modèle des jeux stochastiques est défini et différents algorithmes pour le résoudre sont présentés et discutés. La section 4.4 conclut ce chapitre.

4.2. Rappel sur la théorie des jeux

4.2.1. *Quelques définitions de base*

Une partie de poker, la formation d'une équipe, ou une négociation entre agents pour la prise de rendez-vous sont autant de jeux différents obéissant à des règles spécifiques. Dans ces jeux, chaque participant ne peut être totalement maître de son sort ; on dit alors que tous les intervenants se trouvent en situation d'interaction stratégique [THI 04]. La théorie des jeux vise à étudier formellement ce type de jeux où le sort de chaque agent participant dans le jeu dépend non seulement des décisions qu'il prend mais également des décisions prises par les autres agents intervenant dans le jeu. Dès lors, le « meilleur » choix pour un agent dépend généralement de ce que font les autres.

Les agents participants à un jeu sont appelés joueurs. Ainsi un joueur est un agent qui pourrait représenter une entreprise, un robot, un consommateur etc. et qui agit pour son propre compte selon le principe de la rationalité qui vise à maximiser soit son utilité soit une mesure de performance donnée[1] comme le précisent Russell et Norvig [**?**]. Ainsi, chaque agent cherche à prendre les « meilleures décisions » pour lui-même et ne fait pas référence à un quelconque « sacrifice » pour autrui. Bien entendu, ceci n'est plus valable si on s'intéresse à des équipes d'agents où les participants poursuivent un objectif commun.

En théorie des jeux, il est important de garder à l'esprit que les agents participants au jeu (appelés dorénavant agents-joueurs) se doivent de choisir leurs propres actions, en tenant compte des actions des autres participants. Ils doivent raisonner sur autrui et se faire une idée aussi précise que possible du comportement possible des autres agents-joueurs. À cet effet, la théorie des jeux admet : i) que chaque agent-joueur

1. Une telle mesure définit le critère de succès du comportement de l'agent.

s'efforce de prendre les meilleures décisions pour lui même et sait que les autres font de même ; ii) que le précédent fait, c'est-à-dire i), est une connaissance commune à tous les agents-joueurs.

4.2.1.1. *Jeux non coopératifs et jeux coopératifs*

En théorie des jeux, on distingue les jeux coopératifs des jeux non-coopératifs. Un jeu est dit *coopératif* si les agents-joueurs peuvent passer entre eux des accords qui les lient de manière contraignante. C'est le cas par exemple si les agents-joueurs s'accordent sur un contrat, un accord devant une autorité etc., où il est prévu une sanction légale en cas de non respect du contrat ou de l'accord. Dans ce cas, on dit que les agents-joueurs forment une *coalition* dont les membres agissent de concert. Lorsque les agents-joueurs n'ont pas la possibilité de former des coalitions, on dit que le jeu est *non-coopératif*. Dans ce type de jeu, on spécifie toutes les options stratégiques offertes aux agents-joueurs, chose qu'on ne fait pas dans les jeux coopératifs.

Un jeu non-coopératif peut être défini de deux manières différentes (qui sont toutefois équivalentes) : stratégique (ou normale) et extensive. Un *jeu en forme stratégique* est une collection de stratégies décrivant les actions de chaque agent-joueur dans toutes les situations concevables du jeu, ainsi que les *gains* obtenus par chacun lorsque les stratégies de tous les agents sont connues. La figure 4.1 représente un exemple de jeu en forme stratégique pour deux agents-joueurs, **Entreprise1** et **Entreprise2** ayant à leur disposition les actions *produit* et *ne produit pas*. Par convention les gains sont reportés sous la forme (x, y) pour une combinaison d'actions $action_{Entreprise1} \times action_{Entreprise2}$ et où x est le gain de l'agent-joueur ligne (ici **Entreprise1**) et y est le gain de l'agent-joueur colonne (ici **Entreprise2**).

		Entreprise2	
		produit	*ne produit pas*
Entreprise1	*produit*	$-3, -2$	$10, 0$
	ne produit pas	$0, 8$	$0, 0$

Figure 4.1. *Exemple de jeu en forme stratégique. Des valeurs dans chacun des cases indiquent les gains (en termes d'utilité) de chaque agent-joueur pour chacune des actions jouées conjointement par les joueurs.*

Un jeu en forme extensive est défini par un arbre qui décrit comment le jeu est joué. Dans ce cas, chaque sommet de l'arbre spécifie le (ou les) agent(s)-joueur(s) qui doit (doivent) choisir une action à ce moment du jeu ainsi que l'information dont chaque agent-joueur dispose lors de la prise de décision. Les gains que chaque agent-joueur peut réaliser après avoir suivi un des chemins possibles au sein de l'arbre sont donnés aux sommets terminaux de l'arbre.

Les jeux en forme extensive ne sont pas traités dans ce chapitre. Pour en savoir plus, le lecteur peut se référer aux ouvrages suivants [YIL 03, FUD 91].

Définissons maintenant les jeux en forme stratégique plus formellement.

4.2.1.2. *Jeu en forme stratégique, stratégie pure*

Un jeu G en forme stratégique est un tuple $\langle Ag, \{A_i : i = 1 \ldots |Ag|\}, \{R_i : i = 1 \ldots |Ag|\}\rangle$. Les éléments constitutifs de ce jeu sont les suivants [THI 04] :

– $Ag = 1 \ldots |Ag|$ est l'ensemble fini des agents-joueurs. Pour éviter toute confusion, un agent-joueur quelconque est noté i. Bien entendu $i \in Ag$.

– a_i désigne la stratégie de l'agent-joueur i. Une telle stratégie décrit de manière précise ce qu'un joueur fait. Par extension, l'ensemble A_i décrit toutes les stratégies disponibles pour le joueur i. Bien entendu, $a_i \in A_i$. Un jeu en forme stratégique est fini, si l'ensemble des actions, A_i, de chaque agent-joueur est fini.

– Dès lors, $\mathbf{a} = (a_1, \ldots, a_i, \ldots, a_{|Ag|}) \in A_1 \times \ldots \times A_i \times \ldots \times A_{|Ag|} \equiv \mathbf{A}$ est une issue du jeu ; autrement dit une combinaison de stratégies où a_i est la stratégie pour l'agent i. Dans le reste du chapitre, $\mathbf{a}_{-i} \in \mathbf{A}_{-i}$ désigne l'ensemble de toutes les stratégies choisies par les agents-joueurs sauf celle du joueur i.

– $R_i(\mathbf{a}) \in \mathbb{R}$ est la fonction de récompense du joueur i. On voit bien que la fonction de récompense de l'agent-joueur i dépend non seulement de sa stratégie a_i, mais aussi de celles des autres joueurs reflétées dans $\mathbf{a}$. Bien entendu, le joueur i préfère strictement l'issue $\mathbf{a}$ à l'issue $\mathbf{a}'$ si $R_i(\mathbf{a}) > R_i(\mathbf{a}')$. Dans le cas, où $R_i(\mathbf{a}) = R_i(\mathbf{a}')$, i est dit indifférent aux deux issues $\mathbf{a}$ et $\mathbf{a}'$.

– Chaque agent-joueur connaît, outre les siens, les ensembles de stratégies et les fonctions de gains de tous les autres joueurs.

Cette dernière hypothèse caractérise le jeu comme étant en *information complète*. À l'inverse, un jeu est dit en *information incomplète* si les agents-joueurs ne connaissent certains éléments du jeu qu'en termes de probabilités.

Une *stratégie pure* reflète une action ou une suite d'actions choisies par chaque agent-joueur de façon déterministe (par opposition à stochastique). Dans certains cas, il est préférable d'avoir recours à une *stratégie mixte* définie comme une distribution de probabilité sur l'ensemble des stratégies pures. Soit A_i un ensemble des stratégies à la disposition d'un joueur i. Désignons alors la stratégie pure d'un agent i comme une simple action, soit $a_i \in A_i$, et la stratégie mixte comme une « politique » (où distribution de probabilité) sur ces stratégies pures, soit $\pi_i = \Delta A_i$. Conformément à cette notation, on va aussi noter par $\pi_i^{a_i}$ la probabilité de l'agent i de jouer une action $a_i \in A_i$ et $\pi_{-i} = \Delta \mathbf{A}_{-i}$ la politique conjointe des autres agents par rapport à l'agent i.

4.2.1.3. *Jeu à somme nulle et minimax*

Un jeu à deux joueurs (notés 1 et 2) est à somme nulle si $R_1(\mathbf{a}) + R_2(\mathbf{a}) = 0$ et ce $\forall \mathbf{a} \in \mathbf{A}$. Autrement dit, les gains de l'un sont les pertes de l'autre et les joueurs

sont donc des opposants. Ce genre de jeu particulier peut trouver une solution via le principe de *minimax* (ou *maximin*).

Minimax est une technique de recherche d'une solution dans les jeux à somme nulle. L'algorithme Minimax a été élaboré en 1928 par John von Neumann [NEU 28]. Il s'applique pour les jeux à 2 joueurs à somme nulle. Les agents sont assignés à un des deux rôles : soit **Max** soit **Min**. Le joueur **Max** est sensé *maximiser* sa récompense, alors que le joueur **Min** est sensé *minimiser* le récompense de **Max** (ce qui revient à maximiser sa propre récompense).

Pour illustrer cela, supposons que le joueur i est celui qui cherche à maximiser. Il dispose de m stratégies a_{ik} avec $k = 1, 2, \ldots, m$. Le joueur j est celui qui cherche à minimiser. Il dispose de n stratégies $a_{jk'}$ avec $k' = 1, 2, \ldots, n$. L'ensemble de tous les gains possibles que i peut obtenir est représenté par une matrice R_i de dimensions $m \times n$ avec l'entrée $R_i(k, k')$. Dés lors, dans cette matrice R_i, l'agent i sélectionne les lignes de R_i, tandis que l'agent j sélectionne les colonnes. Dans ce cas, si le joueur i choisit la stratégie k tandis que j choisit la stratégie k', alors j doit payer à i le gain $R_i(k, k')$. Il convient de noter que les paiements négatifs sont permis car on est dans un jeu à somme nulle. On pourrait aussi dire que i reçoit la quantité $R_i(k, k')$ alors que j reçoit la quantité $-R_i(k, k')$.

Considérons alors l'exemple montré en figure 4.2 ci dessous. Bien entendu, un tel jeu peut être représenté simplement par la matrice de la figure 4.3.

		Min	
Max	$3, -3$	$1, -1$	8,-8
	$4, -4$	$10, -10$	0,0

Figure 4.2. *Jeu à somme nulle. Dans la matrice, les valeurs de la forme $\cdot, \cdot$ représentent respectivement les utilités des agents **Max** et **Min** pour chacune des actions conjointes.*

		Min	
Max	3	1	8
	4	10	0

Figure 4.3. *Autre représentation matricielle d'un jeu à somme nulle. Les valeurs dans la matrice représentent seulement les utilités de l'agent **Max**.*

Dans ce jeu, la question est de savoir quelle option devra choisir un agent rationnel. Pour cela, on peut considérer les *niveaux de sécurité* de chacun des agents [HAU 98]. Il est alors facile de voir que, si **Max** choisit la première ligne, quoi que fasse **Min**,

il fera au moins un gain de 1. En choisissant la deuxième ligne, il risque de faire un gain nul. De façon similaire, en choisissant la première colonne, **Min** n'aura pas à payer plus que 4, tandis que s'il choisit la seconde ou la troisième colonne, il risque de perdre respectivement 10 ou 8. On voit donc que le niveau de sécurité de **Max** est 1 et il est assuré par le choix de la première ligne, tandis que le niveau de sécurité de l'agent **Min** est 4 et il est assuré par le choix de la première colonne. Il convient de noter que :

$$1 = \max_{k} \min_{k'} a_{kk'}$$

$$4 = \min_{k'} \max_{k} a_{kk'}$$

Ceci montre que la stratégie qui assure à l'agent **Max** son niveau de sécurité est la *stratégie maximin*. Symétriquement, la stratégie qui assure à l'agent **Min** son niveau de sécurité est la *stratégie minimax.*

PROPOSITION 4.1 [HAU 98].– *Dans une matrice représentant un jeu à deux joueurs à somme nulle, on a l'inégalité suivante :*

$$\max_{k} \min_{k'} a_{kk'} \leqslant \min_{k'} \max_{k} a_{kk'}$$

Il convient de noter que la solution maximin (ou minimax) est acceptable si les joueurs jouent chacun à leur tour, par exemple, le joueur i commence par choisir une action puis le joueur j fait son choix en fonction de l'action jouée par j. Cependant, si les deux joueurs avaient à jouer simultanément, une étude approfondie du jeu précédent illustré en figure 4.2 montrerait, que dans ce cas, les stratégies maximin et minimax ne sont pas des solutions satisfaisantes pour ce jeu. Dans certains cas, toutefois, le jeu pourrait converger vers une solution stable. Considérons un autre exemple, celui de la figure 4.4, emprunté à [HAU 98].

		Min	
Max	10	−15	20
	20	−30	40
	30	−45	60

Figure 4.4. *Jeu où* Maximin = Minimax.

Il est facile ici de voir que :

$$-15 = \max\{-15, -30, -45\}$$

$$-15 = \min\{30, -15, 60\}$$

Ainsi, la paire correspondant aux stratégies maximin and minimax est donnée par $R_i(kk') = (-15, 15)$ et elle correspond à $(k, k') = (1, 2)$, c'est-à-dire à la première ligne et à la deuxième colonne.

Si dans une matrice de jeu à somme nulle il y a une paire (k^*, k'^*), telle que :

$$a_{kk'^*} \leqslant a_{k^*k'^*} \leq a_{k^*k'}$$

on dit alors que la paire (k^*, k'^*) est un point selle.

PROPOSITION 4.2 [HAU 98].– *Si dans une matrice de jeu à somme nulle, on a :*

$$\max_k \min_{k'} a_{kk'} = \min_{k'} \max_k a_{kk'} = v$$

alors le jeu admet un point selle en stratégies pures.

Si (k^*, k'^*) est un point selle pour une matrice de jeu, alors les joueurs **Max** et **Min** ne peuvent améliorer leur gain unilatéralement en déviant de k^* et k'^* respectivement. On dit alors que (k^*, k'^*) est un *équilibre*, car tous les joueurs ont intérêt à s'y tenir.

4.2.2. *Jeux statiques en information complète*

Comme on a déjà mentionné plus haut, un jeu en forme stratégique est dit en information complète si chaque agent-joueur connaît, outre les siens, les ensembles de stratégies et les fonctions de gains de tous les autres joueurs. On dit qu'un jeu est *statique* lorsque les joueurs choisissent simultanément leurs actions et reçoivent ensuite leurs gains respectifs. Ainsi, le jeu se joue en un seul coup, contrairement aux jeux dynamiques sur lesquels nous reviendrons plus tard. Parmi ces jeux, les jeux en forme stratégique finis à 2 joueurs occupent une place privilégiée car ils sont simples d'un point de vue présentation bien qu'englobant les principales caractéristiques (hormis la complexité engendrée par un nombre élevé de joueurs) qu'on trouve en théorie des jeux. Comme on l'a précisé plus haut, ces jeux peuvent être représentés sous forme stratégiques et donc sous la forme de matrices dans lesquelles le premier joue verticalement en choisissant une ligne de la matrice et le second horizontalement en jouant une colonne. De tels jeux sont aussi appelés jeux matriciels.

Une telle forme de jeu peut s'illustrer par le célèbre jeu du dilemme du prisonnier. Il s'énonce de la façon suivante : Deux suspects (**Suspect1** et **Suspect2**) sont interrogés séparément par un juge pour un délit grave. Le juge ne dispose pas d'éléments de preuve suffisants pour les condamner et l'aveu d'au moins un est indispensable. Dès lors, il propose à chaque accusé la liberté s'il avoue. Par contre s'il nie et que l'autre avoue, il écope d'une peine de 15 ans. Si les deux avouent ils peuvent espérer bénéficier de circonstances atténuantes et recevoir une peine de 8 ans. Enfin si les deux nient, ils seront condamnés pour des délits mineurs à 1 an de prison chacun. Avouer

revient à dénoncer l'autre (en même temps que soi-même). On notera donc *D* comme « dénoncer » et *N* comme « nier » les deux actions. La matrice des gains des deux joueurs correspondante apparaît sur la figure 4.5.

		Suspect2	
		D	*N*
Suspect1	*D*	$-8,-8$	$0,-15$
	N	$-15,0$	$-1,-1$

Figure 4.5. *La matrice des gains des deux joueurs dans le Dilemme du prisonnier.*

On est donc amené à se poser la question : comment doivent se comporter les deux suspects, en supposant qu'ils soient rationnels ? On peut remarquer qu'*Avouer* est une stratégie qui conduit à une peine moins lourde que la stratégie *Nier* et ce, quel que soit le choix effectué par l'autre joueur. Par conséquent, chacun des suspects a intérêt à opter pour cette stratégie en vue de réduire sa peine. Selon la matrice des gains, chacun écope alors d'une peine de 8 ans de prison, ce qui constitue une condamnation assez lourde. Il faut bien voir que cette stratégie mise sur le fait qu'elle est choisie parce qu'elle donne un gain moindre que l'autre stratégie *et ce sans avoir besoin de se faire une idée de ce que va faire l'autre*. Une telle stratégie est appelée, en théorie des jeux, une *stratégie dominante*.

Définition 5 *Dans un jeu en forme stratégique, une stratégie $a_i \in A_i$ est dite dominante pour le joueur i si, quel que soit $\hat{a}_i \in A_i$ et $\hat{a}_i \neq a_i$, on a :*

$$R_i(a_i, \mathbf{a}_{-i}) \geqslant R_i(\hat{a}_i, \mathbf{a}_{-i}), \quad \forall\, \mathbf{a}_{-i} \in \mathbf{A}_{-i}$$

Dans notre exemple, on voit bien que nos deux suspects ont intérêt à jouer leur stratégie dominante tous les deux et à ne pas dévier. La stratégie conjointe (*D*,*D*) est donc un équilibre (sorte de point fixe) pour les deux où chacun n'a pas intérêt à dévier. Plus généralement, si dans un jeu donné, tous les joueurs ont à leur disposition une stratégie dominante, alors ils ont intérêt à la choisir effectivement et, dans ce cas, le résultat du jeu est appelé *équilibre en stratégies dominantes*.

En fait, l'équilibre en stratégies dominantes existe rarement et il faut faire appel à d'autres types de solutions. Pour ces cas, il existe un concept de solution plus faible qui s'appelle l'équilibre de Nash.

4.2.2.1. *Équilibre de Nash*

Définition 6 *On dit qu'une combinaison de stratégies $\mathbf{a}^*$ est un équilibre de Nash si on a l'inégalité suivante pour chaque joueur i :*

$$R_i(a_i^*, \mathbf{a}^*_{-i}) \geqslant R_i(a_i, \mathbf{a}^*_{-i}) \quad \forall\, a_i \in A_i$$

Autrement dit, si le joueur i anticipe que les autres participants au jeu vont choisir les stratégies associées au vecteur $\mathbf{a}^*_{-i}$, il ne peut que maximiser son gain en choisissant la stratégie a^*_i. Celle-ci est en fait la meilleure réponse de i à $\mathbf{a}^*_{-i}$ (notée br_i, pour *best response*) et elle correspond à :

$$br_i : \mathbf{a}^*_{-i} \mapsto \underset{a_i \in A_i}{\operatorname{argmax}} R_i(a_i, \mathbf{a}^*_{-i})$$

Dès lors, l'équilibre de Nash peut aussi s'écrire :

$$\forall i, a^*_i \in br_i(\mathbf{a}^*_{-i})$$

Comme on peut le voir, l'équilibre de Nash constitue une combinaison de stratégies où chaque joueur maximise ses gains compte tenu des actions supposées des autres. Il a donc une propriété de « stabilité » qui est satisfaite pour chacun des joueurs, c'est pourquoi on parle d'« équilibre ».

Dans l'exemple de la figure 4.6 deux entreprises **Entreprise1** et **Entreprise2** ont la possibilité de se lancer dans la production d'un nouveau bien pour lequel les débouchés sont limités, sans qu'il y ait de compromis possible entre elles si toutes deux décident de produire. Ce jeu comporte 2 équilibres de Nash, (*ne produit pas*, *produit*) dont les gains sont (0,8) et (*produit*, *ne produit pas*) dont les gains sont (10,0) chacun correspondant à une situation où l'une des entreprises produit, l'autre s'abstenant de le faire. Cet exemple montre que des deux équilibres, il n'a y a pas un qui apparaisse plus raisonnable qu'un autre.

		Entreprise2	
		Produit	*Ne produit pas*
Entreprise1	*Produit*	$-3, -2$	$10, 0$
	Ne produit pas	$0, 8$	$0, 0$

Figure 4.6. *Multiplicité de l'équilibre de Nash. Ce jeu comporte deux équilibres de Nash en stratégies pures : (*ne produit pas*,* produit*) dont les gains sont (0,8) et (*produit*,* ne produit pas*) dont les gains sont (10,0).*

À cette difficulté de multiplicité d'équilibres, s'ajoute le fait qu'il peut ne pas y avoir du tout d'équilibre de Nash, en stratégies pures, pour un jeu particulier. Un exemple bien connu est celui du jeu de pile ou face (*matching pennies*) présenté sous la forme stratégique en figure 4.7.

Dans ce cas, on pourrait penser à un mécanisme aléatoire (constitué par la composition des loteries[2] des différents intervenants) qui décide pour les joueurs. Pour cela,

2. une loterie est une loi de probabilité de choix, voir le chapitre 5

		Joueur2	
		Pile	*Face*
Joueur1	*Pile*	$1, -1$	$-1, 1$
	Face	$-1, 1$	$1, -1$

Figure 4.7. *Le jeu de pile ou face : un exemple de jeu n'ayant pas d'équilibre de Nash en stratégies pures.*

on suppose que chaque joueur choisit une loterie définie sur l'ensemble des stratégies pures. Techniquement, chaque joueur associe une probabilité p_i à la stratégie a_i et laisse au mécanisme aléatoire le soin de décider. Dans ce contexte, chaque joueur vise maintenant à maximiser ses gains espérés en choisissant la meilleure loterie possible, autrement dit la meilleure stratégie mixte.

Notons qu'en théorie des jeux, le terme « gain espéré » (où « utilité espérée ») d'un agent-joueur est la récompense qu'il s'attend à obtenir étant donné sa politique (stratégie mixte) et les politiques des autres joueurs. Désignons le gain espéré de l'agent i par u_i, alors :

$$\begin{aligned} u_i^{(\pi_i, \pi_{-i})} &= E_{\mathbf{a} \in \mathbf{A}} R_i(\mathbf{a}) \\ &= \sum_{a_i \in \mathcal{A}_i} \sum_{\mathbf{a}_{-i} \in \mathbf{A}_{-i}} R_i(a_i, \mathbf{a}_{-i}) \pi_i^{a_i} \pi_{-i}^{\mathbf{a}_{-i}} \end{aligned} \tag{4.1}$$

Dans le jeu de pile ou face, présenté en figue 4.7, le **Joueur1** a une probabilité p de choisir *Pile* et une probabilité de $1 - p$ de choisir *Face*. Pour le **Joueur2** les deux probabilités sont respectivement de q et $1 - q$. Dès lors, le gain espéré du **Joueur1** est reflétée par la fonction u_1 linéaire en p suivante :

$$u_1 = pqR_1(Pile, Pile) + p(1-q)R_1(Pile, Face) + (1-p)qR_1(Face, Pile) + \\ (1-p)(1-q)R_1(Face, Face)$$

Maximiser ce gain espéré revient donc à chercher $d(u_1)/dp = 0$ soit :

$$qR_1(Pile, Pile) + (1-q)R_1(Pile, Face) = qR_1(Face, Pile) + (1-q)R_1(Face, Face)$$

Si on remplace les R_i par les valeurs indiquées dans la matrice présentée en figure 4.7, on obtient alors :

$$q - (1-q) = -q + (1-q)$$

Soit alors $q = 1/2$.

Le même raisonnement pour le **Joueur2** (où cette fois-ci on chercherait le point où $d(u_2)/dq = 0$) aurait donné $p = 1/2$.

Le résultat (1/2,1/2) est appelé *équilibre en stratégies mixtes* et il correspond au fait que l'un ou l'autre des joueurs choisisse une fois sur deux pile et une fois sur deux face. Mais est-ce qu'on peut trouver un équilibre de ce type dans n'importe quel jeu en forme stratégique ? Si le jeu est fini, la réponse est définitivement oui, grâce au théorème suivant :

THÉORÈME 4.1 [NAS 51].– *Tout jeu en forme stratégique fini admet un équilibre de Nash en stratégies mixtes.*

On pourrait se demander si la solution donnée par l'équilibre de Nash correspond à un mécanisme de coordination efficace. Deux concepts peuvent aider à répondre à cette question : l'efficacité au sens de Pareto et l'optimum de Pareto.

Pour le premier concept, on peut dire qu'une combinaison de stratégie $\hat{\mathbf{a}}$ *domine au sens de Pareto* une autre combinaison $\mathbf{a}$ si :

$$R_i(\hat{a}_i, \hat{\mathbf{a}}_{-i}) \geq R_i(a_i, \mathbf{a}_{-i}) \quad \forall i$$

$$\text{et}$$

$$\exists j \text{ tel que } R_j(\hat{a}_j, \hat{\mathbf{a}}_{-j}) > R_j(a_j, \mathbf{a}_{-j})$$

Une combinaison de stratégies $\hat{\mathbf{a}}^*$ est un *optimum de Pareto* s'il n'existe pas une autre combinaison qui la domine au sens de Pareto. Par exemple, dans le dilemme du prisonnier présenté en figure 4.5, la combinaison (*D*,*D*) est un équilibre de Nash mais la combinaison (*N*,*N*) la domine au sens de Pareto. Il faudra donc retenir que l'équilibre de Nash n'est pas nécessairement un optimum de Pareto.

Clairement, un équilibre de Nash n'est pas nécessairement un optimum de Pareto mais quand il y a multiplicité des équilibres de Nash, un équilibre peut en dominer un autre au sens de Pareto. Le problème est de savoir comment « attirer » les joueurs vers cet équilibre dominant au lieu d'un autre équilibre qui sera dominé.

4.2.3. *Jeux dynamiques en information complète*

Contrairement aux jeux en forme stratégique (ou jeux matriciels) qui sont joués une fois, les jeux dynamiques décrivent les processus étendus dans le temps. Ces processus comportent plusieurs intervenants (agents-joueurs) qui peuvent conditionner leurs comportement au moment présent sur les décisions observables des autres

joueurs dans le passé. Dans cette section, on suppose que le jeu se déroule en plusieurs étapes et que toutes les actions passées sont observables et connues par tous les participants. Dans ce contexte, une étape pourrait représenter, mais pas toujours, une période temporelle. Une stratégie dans un tel jeu spécifie l'action que choisit chaque agent-joueur à chaque étape où il intervient, en fonction de l'état du jeu qui prévaut en ce moment. Dès lors, l'historique du jeu a son importance et chaque agent-joueur choisit l'action qu'il convient de faire en tenant compte de l'histoire passée du jeu.

Généralement, on distingue deux types de jeux dynamiques en information complète. Un premier type où chaque joueur connaît l'ensemble des actions choisies par tous les autres agents-joueurs avant qu'il ne sélectionne sa propre action. Il convient de bien voir ici que c'est le seul joueur qui est censé prendre sa décision à l'étape considérée. Ce jeu est alors appelé *jeu en information parfaite*. Dans le second type, appelé *jeu répété*, plusieurs agents choisissent leurs actions simultanément à une étape donnée du jeu. Pour chaque joueur, les actions des autres joueurs ne sont toutefois pas connues, mais l'historique lui l'est et il influence le choix de chacun.

Un cadre conceptuel général des jeux dynamiques peut être défini comme suit. On désigne par $\mathbf{a}^\theta$ le vecteur des actions conjointes choisies à l'étape θ du jeu par des participants qui interviennent à cette étape. Soit t une étape quelconque du jeu. On définit alors l'historique du jeu à l'étape t, $h^t = \{\mathbf{a}^0, \mathbf{a}^1, \ldots, \mathbf{a}^{t-1}\}$, par la séquence de toutes les décisions prises par les joueurs lors des étapes antérieures $\theta = 0, 1, \ldots, t-1$, avec h^0 comme historique initial. Par ailleurs, toutes les actions passées sont observables et connues par tous les agents-joueurs. Il convient de noter que, pour $\theta > t$, le reste du jeu peut être vu comme un autre jeu induit par l'historique h^t ; ce nouveau jeu est appelé sous-jeu $G(h^t)$.

Un tel cadre formel va maintenant nous permettre de définir ce que l'on attend par stratégie pure au niveau d'un jeu dynamique admettant T étapes. Une stratégie pure pour le joueur i est définie par une suite de T applications A_i^t de H^t vers $A_i(H^t)$, soit donc $A_i^t : H^t \mapsto A_i(H^t)$, où H^t est l'ensemble de toutes les historiques possibles jusqu'au temps t, avec $h^t \in H^t$. Pour un joueur donné i, une stratégie pure est donc une suite de règles de sélection d'une action particulière par un tel joueur à chaque étape du jeu, compte tenu de l'historique qui s'est déroulé jusqu'alors. Contrairement à ce qu'on a vu précédemment dans le cadre d'une stratégie pure dans un jeu statique, ici, la définition prend en compte les décisions choisies antérieurement. Elle permet donc une analyse dynamique des choix.

Considérons d'abord les jeux dynamiques en information parfaite.

4.2.3.1. *Jeux dynamiques en information parfaite*

Soit l'arbre représenté en figure 4.8 où le jeu se déroule en plusieurs étapes. Ainsi pour le joueur 1 (dénoté par un nœud 1), une stratégie consiste à choisir entre les

actions a et b. Pour le joueur 2 (nœud 2) qui intervient juste après, sa stratégie a_2 est une fonction définie sur l'ensemble des stratégies de 1. Le principe ici consiste par exemple à utiliser la *rétroduction*[3] où le raisonnement se fait en sens inverse du déroulement normal du jeu. Dès lors, le jouer 1 va se dire que le joueur 2 va jouer :

– b s'il joue a aboutissant ainsi au gain (2,1) ou

– jouer a s'il joue b aboutissant ainsi au gain de (1,1).

Dès lors, le joueur 1 va jouer a amenant ainsi le joueur 2 à jouer b et aboutissant ainsi à l'équilibre de Nash (2,1) pour lequel aucune déviation unilatérale n'est payante.

En fait, ce raisonnement est sous-tendu par le fait que le joueur 2 est sensé choisir la meilleure action pour lui. Ainsi, si 1 joue a alors 2 joue b, si en revanche 1 joue b alors 2 joue a. Dès lors, l'agent-joueur 1 intègre une telle connaissance et agit en conséquence pour aligner sa meilleure réponse.

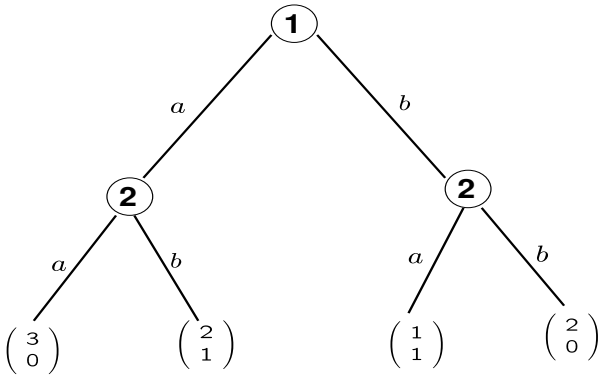


Figure 4.8. *Un exemple de jeux dynamique joué étape par étape.*

Pour en savoir plus sur les jeux dynamiques en information parfaite, le lecteur peut se référer à l'un des ouvrages [YIL 03, FUD 91].

Les jeux répétés, un autre type des jeux dynamiques en information complète, sont de plus grand intérêt pour nous, car ils sont à la base du concept des jeux stochastiques – un des plus puissants modèles d'interaction entre agents rationnels.

3. *backward induction*, qu'on traduit aussi par « induction rétroactive ».

4.2.3.2. *Les jeux répétés*

Avec les jeux répétés, on voudrait modéliser des situations où des agents-joueurs interagissent de manière répétitive chacun avec l'autre, en jouant le même jeu. Contrairement aux jeux dynamiques en information parfaite, les joueurs d'un jeu répété choisissent leurs actions de manière simultanée sans connaître le choix des autres joueurs. Une fois les action choisies, celles-ci sont alors connues par les autres agents et dès lors font partie de l'historique du jeu.

Ainsi, le jeu du dilemme du prisonnier peut par exemple être répété plusieurs fois où l'on aurait par exemple $\{(N, N), (D, N), (D, D), \ldots\}$.

Dans de telles interactions, un joueur peut, à chaque étape, conditionner son comportement sur les comportements passés des autres intervenants. En fait, les jeux répétés sont un cas particulier des jeux dynamiques introduits précédemment. La particularité réside dans le fait que la répétition se fait sur la base d'un même jeu. Quand ils sont engagés dans une situation répétitive, les joueurs doivent non seulement considérer leur gain à court terme mais également leur paiement à long terme. Par exemple, si un dilemme du prisonnier est joué une fois, les joueurs auront tendance à jouer l'équilibre de Nash soit, *(D,D)*. En revanche, si le même jeu est répété par les mêmes deux joueurs, il y aurait peut être une possibilité de faire émerger une coopération (entente implicite) qui aboutirait à *(N,N)*. *L'idée générale sous-tendant les jeux répétés réside dans le fait que les joueurs doivent trouver un compromis entre exploiter le gain à court terme et les gains à long terme.*

On distingue en général deux classes de jeux répétés : (a) jeux répétés finis et ; (b) jeux répétés infinis. Un historique terminal dans un jeu répété fini est n'importe quel historique de longueur T, où $T \leq \infty$ est le nombre de périodes durant lesquelles le jeu est répété. Dans le cas où le jeu est répété infiniment, l'historique terminal est alors de longueur infinie. Tout historique non terminal débute un sous-jeu dans le jeu répété.

Selon la définition d'une stratégie pure d'un jeu dynamique donnée plus haut, on pourrait spécifier dans le cas du dilemme du prisonnier

$$a_i(h^0) = N$$

$$a_i(h^t) = \begin{cases} N & \text{si } a_j^\tau = N, j \neq i, \text{ pour } \tau = 0, 1, \ldots, t-1 \\ D & \text{autrement} \end{cases}$$

Une telle stratégie traduit « commencer par nier dans la première période et continuer ainsi tant que l'autre le fait également dans les périodes précédentes ; si ce n'est pas le cas, dénoncer ». Cette stratégie est appelée la stratégie *grim trigger*. D'autres stratégies peuvent être mises en évidence, en utilisant la même formulation, en particulier : (i) toujours dénoncer (*always Defect–ALL-D*) ; (ii) toujours nier (*always cooperate–ALL-C*·) ; (iii) donnant-donnant (*Tit for Tat–TFT*) etc.

Il faudra également prendre en compte les choix des agents en fonction du temps dans la mesure où ils accordent de l'importance à la date à laquelle ils obtiennent les différents gains : un dollar obtenu maintenant n'aura pas la même valeur qu'un dollar obtenu dans dix jours. Étant donné une séquence $\{\mathbf{a}^0, \mathbf{a}^1, \ldots, \mathbf{a}^t, \ldots\}$ d'actions conjointes, la fonction d'utilité actualisée de cette séquence pour un joueur i, $u_{i,\gamma}$, est la récompense actualisée sur l'ensemble des périodes, soit donc, en désignant comme d'habitude par γ le facteur d'actualisation :

$$u_{i,\gamma} = \sum_{t=1}^{\infty} \gamma^{t-1} R_i(\mathbf{a}^t)$$

Bien entendu, si $R_i(\mathbf{a}^t) = c$ et ce $\forall t$, où c étant une constante quelconque et si $\gamma \in [0, 1[$, alors cette valeur devient[4] :

$$u_{i,\gamma} = c \sum_{t=1}^{\infty} \gamma^{t-1} = \frac{c}{1-\gamma}$$

Jusqu'ici on n'a parlé que des stratégies pures dans le cadre des jeux dynamiques. Une *stratégie mixte* π_i pour le joueur i est une séquence de fonctions, $\pi_i(h^t) : H^t \mapsto \mathcal{A}_i(H^t)$, où $\mathcal{A}_i$ est l'espace des distributions de probabilité sur A_i, soit $\mathcal{A}_i(H^t) = \Delta A_i(H^t)$, qui lie donc les historiques t-périodes possibles à des actions mixtes $\alpha_i \in \mathcal{A}_i$. On doit noter ici que la stratégie d'un joueur i ne peut pas dépendre des valeurs passées des probabilités des autres joueurs mais des valeurs passées de a_i.

Voyons maintenant comment un jeu répété pourrait être analysé en termes de gains escomptés et contentons-nous pour cela du cas des stratégies pures pour l'exemple du dilemme du prisonnier (voir figure **??**). Dans ce cas, les gains des joueurs pour toute la séquence des répétitions du jeu pourraient être représentée par *la moyenne des récompenses* qu'ils obtiennent à chaque période. Dans le cas du **Suspect1** par exemple sa fonction de gain sur l'ensemble du jeu répété *pour toujours* est

$$\overline{u}_1 = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} R_1(\mathbf{a}^t)$$

où R_1 est la fonction de récompense d'un tel suspect. On pourrait de la même façon imaginer qu'un tel suspect utilise *la valeur actualisée de ses récompenses* sur la totalité de la séquence de jeux.

$$u_{1,\gamma} = \sum_{t=1}^{\infty} \gamma^{t-1} R_1(\mathbf{a}^t) \quad \gamma \in [0, 1[$$

4. On pourrait commencer par mettre c en facteur et voir $\rho = 1 + \gamma + \gamma^2 + \gamma^3 + \ldots$ sous la forme d'une série infinie. Si maintenant on s'intéresse à $\rho\gamma = \gamma + \gamma^2 + \gamma^3 + \ldots$ et donc $\rho - \rho\gamma = 1$ menant donc à $\rho = 1/1 - \gamma$.

où γ est le facteur d'actualisation du **Suspect1**. En combinant les deux fonctions précédentes, on obtient *la moyenne actualisée des récompenses* :

$$\overline{u}_{1,\gamma} = (1-\gamma)\sum_{t=1}^{\infty}\gamma^{t-1}R_1(\mathbf{a}^t)$$

où on espère que l'égalité suivante est vérifiée :

$$\lim_{\gamma\to 1}\overline{u}_{1,\gamma} = \overline{u}_1$$

Ainsi donc, il existe plusieurs possibilités pour représenter les gains à long terme des intervenants dans un jeu répété. Voici un exemple tiré de [YIL 03] sur la manière de les utiliser. Reprenons le dilemme du prisonnier illustré en figure **??** répété infiniment et considérons la stratégie *grim trigger* discutée plus haut. Admettons que le joueur **Suspect1** a adopté cette stratégie et le **Suspect2** est au courant de cela. On pourrait se demander quelle est alors la stratégie optimale de **Suspect2**, face à cette stratégie de **Suspect1**. S'il joue tout le temps *N*, il va obtenir un flux continu de récompenses égales à -1 jusqu'à la fin des temps. La valeur actualisée de ces récompenses dans ce cas est :

$$\sum_{t=1}^{\infty}-\gamma^{t-1} = \frac{-1}{1-\gamma}$$

En revanche, s'il commence par *D* dès le premier tour, alors il va obtenir initialement 0 puis -8 pour le reste des périodes, ce qui lui donne une valeur actualisée de :

$$0+(-8)\gamma+(-8)\gamma^2+\ldots = -8(\gamma+\gamma^2+\ldots) = \frac{-8\gamma}{1-\gamma}$$

Le joueur **Suspect2** aura donc intérêt à coopérer dès le début si

$$\frac{-1}{1-\gamma} > \frac{-8\gamma}{1-\gamma} \Leftrightarrow \gamma > \frac{1}{8}$$

La stratégie précédente du **Suspect1** et sa contrepartie qu'on vient de détailler pour le **Suspect2** forment un équilibre de Nash du dilemme du prisonnier répété infiniment en valeurs actualisées pour les valeurs de $\gamma > \frac{1}{8}$. Si on choisissait une autre forme pour la fonction d'utilité des agents (par exemple, la moyenne des récompenses ou la moyenne actualisée des récompenses), on pourrait obtenir une autre solution au même problème. Pour plus de précisions sur ces aspects, le lecteur est encouragé à consulter les ouvrages de référence [FUD 99, GEN 00, OSB 04, YIL 03].

4.3. Jeux stochastiques

Les jeux stochastiques (SG, pour *stochastic games*) étendent les MDP au cas où il y a plusieurs agents-joueurs dans un environnement commun. Ces agents exécutent une action conjointe qui définit la récompense obtenue par les agents et le nouvel état de l'environnement. De l'autre côté, un jeu stochastique peut être vu comme un jeu répété à plusieurs états. Ça veut dire qu'après avoir joué un jeu matriciel (correspondant à un état du jeu stochastique), les agents-joueurs sont transférés dans un autre état du jeu stochastique pour jouer un autre jeu matriciel. C'est donc un modèle hybride réunissant jeux dynamiques (répétés) et MDP.

4.3.1. *Définition et équilibre d'un jeu stochastique*

Formellement, un jeu stochastique est défini par un quintuplet $\langle Ag, \mathbf{S}, \{A_i : i = 1, \ldots, |Ag|\}, \{R_i : i = 1, \ldots, |Ag|\}, T\rangle$ où Ag est l'ensemble des agents et $|Ag|$ est leur nombre, $\mathbf{S}$ l'ensemble fini d'états du jeu, $A_i = \left\{a_i^1, \ldots, a_i^{|A_i|}\right\}$ l'ensemble d'actions de l'agent i, $R_i : \mathbf{S} \times A_1 \times \ldots \times A_{|Ag|} \mapsto \mathbb{R}$ est la fonction de récompense de l'agent $i \in Ag$ et $T : \mathbf{S} \times A_1 \times \ldots \times A_{|Ag|} \times \mathbf{S} \mapsto \mathbb{R}$ est le modèle de transition entre états, dépendant de l'action conjointe des agents. Notons que contrairement aux MDP, les états des jeux stochastiques sont des vecteurs (ou états conjoints) composés des états propres à chaque agent, $\mathbf{S} = S_1 \times \ldots \times S_i \times \ldots \times S_{|Ag|}$, où S_i est l'ensemble des états propres à l'agent i.

À chaque tour du jeu, étant donné l'état courant $\mathbf{s} \in \mathbf{S}$, les agents choisissent les actions $a_1, \ldots, a_{|Ag|}$. Chaque agent i obtient alors la récompense $R_i\left(\mathbf{s}, (a_1, \ldots, a_{|Ag|})\right)$ et le système passe dans l'état $\mathbf{s}'$ en suivant le modèle de transition T, qui vérifie $\sum_{\mathbf{s}' \in \mathbf{S}} T\left(\mathbf{s}, (a_1, \ldots, a_{|Ag|}), \mathbf{s}'\right) = 1$. Une politique $\pi_i : \mathbf{S} \mapsto [0, 1]^{|A_i|}$ pour l'agent i définit une stratégie locale en chaque état au sens de la théorie des jeux. Autrement dit, $\pi_i(\mathbf{s})$ est un vecteur dont les éléments définissent une distribution de probabilité sur les actions du joueur i, spécifiques au jeu en forme normale défini par l'état $\mathbf{s}$.

Le terme d'utilité espérée d'un joueur en théorie des jeux désigne l'espérance de récompense sur les *stratégies des joueurs adverses*, alors que la fonction de valeur des MDP est l'espérance *temporelle* de la récompense. Nous emploierons donc le concept d'utilité $U_i(\mathbf{s})$ en jeu stochastique comme l'espérance temporelle des utilités espérées $u_i(\mathbf{s})$ de l'agent i définies pour chaque état $\mathbf{s}$ de manière similaire aux utilités espérées des jeux en forme normale. Dès lors, $u_i^{(\pi_1, \ldots, \pi_{|Ag|})}(\mathbf{s}) = E_{\mathbf{a} \in \mathbf{A}} R_i(\mathbf{s}, \mathbf{a})$. Par conséquent, les utilités $U_i(\mathbf{s})$ des états pour chaque joueur i, associées à la politique conjointe $\pi \equiv \times_{i=1}^{|Ag|} \pi_i$, sont définies comme l'utilité espérée par l'agent i à partir de

l'état $\mathbf{s}$ si tous les agents suivent cette politique conjointe :

$$\begin{aligned} U_i^{\pi}(\mathbf{s}) &= E\left[\sum_{t=0}^{\infty}\gamma^t u_i^{\pi}(\mathbf{s})^t|\mathbf{s}^0=\mathbf{s}\right] \\ &= u_i^{\pi}(\mathbf{s}) + \gamma\sum_{\mathbf{a}\in\mathbf{A}}\sum_{\mathbf{s}'\in\mathbf{S}} T(\mathbf{s},\mathbf{a},\mathbf{s}')\pi^{\mathbf{a}}(\mathbf{s})U_i^{\pi}(\mathbf{s}') \end{aligned}$$

où $\pi^{\mathbf{a}}(\mathbf{s})$ dénote la probabilité de l'action conjointe $\mathbf{a}$ dans l'état $\mathbf{s}$ selon la politique conjointe π. Comme pour les MDP, ici aussi $\mathbf{s}^0$ est l'état initial et $\gamma \in [0,1[$ est le facteur d'actualisation.

Comme indiqué plus haut, chaque état d'un jeu stochastique peut être vu comme un jeu en forme normale. Le processus de transition entre les deux états est illustré par la figure 4.9.

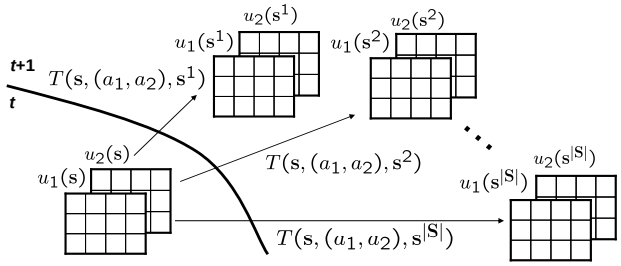


Figure 4.9. *Jeu stochastique à deux joueurs : transitions possibles entre le tour t et le tour $t+1$ lorsque les joueurs jouent l'action conjointe $\mathbf{a}=(a_1,a_2)$; chaque état peut être vu comme un jeu en forme normale.*

Les jeux stochastiques sont un cadre formel permettant de modéliser un environnement multiagent non-coopératif. Le fait d'être non-coopératif signifie que les agents poursuivent des objectifs individuels. Ils peuvent cependant être amenés à se coordonner, voire à coopérer, pour atteindre leur but individuel.

Dans un jeu stochastique, un équilibre de Nash est un vecteur de stratégies $(\pi_1^*,\ldots,\pi_{|Ag|}^*)$ tel que, pour tout état $\mathbf{s}\in\mathbf{S}$ et $i=1,\ldots,|Ag|$,

$$U_i^{(\pi_1^*,\ldots,\pi_{|Ag|}^*)}(\mathbf{s}) \geqslant U_i^{(\pi_1^*,\ldots,\pi_{i-1}^*,\pi_i,\pi_{i+1}^*,\ldots,\pi_{|Ag|}^*)}(\mathbf{s}) \quad \forall\pi_i\in\Pi_i$$

où Π^i est l'ensemble des politiques offertes à l'agent i.

Définition 7 *Une politique (ou stratégie) dans un jeu stochastique est dite stationnaire (ou markovienne) si et seulement si la règle de décision qui associe une action à un état dépend seulement de l'état courant.*

Le théorème ci-après montre l'existence d'un équilibre de Nash en stratégies stationnaires.

THÉORÈME 4.2 [FIN 64].– *Tout jeu stochastique escompté à n-joueurs ($n \geq 2$) possède au moins un équilibre de Nash en stratégies stationnaires.*

4.3.2. *Résolution des jeux stochastiques*

Nous présentons ici quelques algorithmes de résolution de jeux stochastiques. Comme on a pu le constater dans la section précédente, les jeux stochastiques sont des jeux multi-étapes qui peuvent être vus comme une extension des MDP aux multiagents. Étant des jeux, ils n'offrent pas de solution optimale, au sens des MDP, qui soit indépendante des autres joueurs. À l'instar des jeux en forme normale, le concept le plus utilisé comme solution d'un jeu stochastique est un équilibre de Nash en stratégies stationnaires, dont l'existence a été prouvé. Cependant, contrairement aux MDP, la solution (ou l'équilibre) dans la théorie des jeux n'est pas toujours unique (comme, par exemple, dans le jeu de la figure 4.6 de la section 4.2). Le fait d'avoir plusieurs équilibres avec des valeurs différentes pose généralement des problèmes de coordination. En effet, dans ce cas, si les agents choisissent de jouer des équilibres différents, l'action conjointe jouée peut ne pas constituer un équilibre. Selon le jeu joué par les agents-joueurs, cela peut constituer une catastrophe si les valeurs des utilités varient beaucoup d'une action conjointe à une autre. Pour simplifier la présentation de ce qui suit, les équilibres sont supposés uniques dans cette section. Le lecteur intéressé par le problème de coordination dans les jeux stochastiques peut se référer à [LIT 94].

Tous les algorithmes proposés dans le cadre des jeux stochastiques que l'on va voir ont la même propriété commune. Ils sont composés de deux parties principales. La première partie peut être vue comme la partie « différences temporelles » pour résoudre la composante « multi-état » du jeu stochastique ; la deuxième partie, quant à elle, est la partie « jeu » pour trouver une solution de la composante « multi-agent » du jeu stochastique.

Selon ces observations, on peut regrouper les algorithmes de jeux stochastiques en quatre catégories (dont le sens sera défini plus loin) en leur donnant les appellations suivantes :

– Itération sur les valeurs + Théorie des jeux classique ;
– Apprentissage par renforcement + Théorie des jeux classique ;
– Apprentissage par renforcement + Modélisation de l'opposant ;
– Apprentissage par renforcement + Descente du gradient.

4.3.2.1. *Itération sur les valeurs + Théorie des jeux classique*

Nous faisons état dans cette sous-section de deux algorithmes élaborés par des chercheurs venant de la communauté de la théorie des jeux. Le premier algorithme trouve un équilibre de Nash d'un jeu stochastique à somme nulle [SHA 53]. Cet algorithme n'est qu'une extension de la technique d'itération sur les valeurs (vue au chapitre 1) aux cas des jeux stochastiques. Pour faire cela, Shapley utilise une fonction $Valeur$ afin de trouver la valeur d'un état. Cette fonction calcule un équilibre de Nash d'un jeu en forme normale associé à cet état. Pour trouver un équilibre de Nash, Shapley utilise l'algorithme minimax qui a un temps d'exécution polynomial en la taille de la matrice du jeu d'état. La définition complète de l'algorithme de Shapley est donnée par l'algorithme 4.1. Dans ledit algorithme, la valeur $U(\mathbf{s})$ désigne le vecteur d'utilités espérées de tous les agents dans l'état $\mathbf{s}$. Les fonctions $Equilibre$ et $Valeur$ retournent respectivement un équilibre d'un jeu d'état (une politique conjointe) et son vecteur de valeurs.

Algorithme 4.1 : Algorithme de Shapley [SHA 53] pour les SG à somme nulle

initialiser $U(\mathbf{s})^0$ par des valeurs arbitraires
$n \leftarrow 0$
répéter
  pour $\mathbf{s} \in \mathbf{S}$ **faire**
    Construire la matrice
    $G(\mathbf{s}) = \{g_{\mathbf{a}} : g_{\mathbf{a}} = R(\mathbf{a}, \mathbf{s}) + \gamma \sum_{\mathbf{s}' \in \mathbf{S}} T(\mathbf{s}, \mathbf{a}, \mathbf{s}') U(\mathbf{s}')^n\}$
    $U(\mathbf{s})^{n+1} = Valeur(G(\mathbf{s}))$
  $n \leftarrow n + 1$
jusqu'à $\| U(\mathbf{s})^{n+1} - U(\mathbf{s})^n \| < \epsilon \; \forall \mathbf{s}$
pour $\mathbf{s} \in \mathbf{S}$ **faire**
  Construire la matrice $G(\mathbf{s})$
  $\pi(\mathbf{s}) = Equilibre(G(\mathbf{s}))$
retourner $U(\mathbf{s})^n$, $\pi(\mathbf{s}) \; \forall \mathbf{s}$

L'algorithme de Shapley est certain de trouver un équilibre dans n'importe quel jeu stochastique à somme nulle, car cet algorithme lui-même est une conséquence directe du théorème suivant :

THÉORÈME 4.3 [SHA 53].– *Tout jeu stochastique G à somme nulle escompté à horizon infini possède une valeur unique. Cette valeur est donnée par la séquence des*

valeurs uniques des jeux d'état $G(\mathbf{s})\forall\mathbf{s}$. Chaque joueur du jeu G possède une stratégie optimale qui utilise les stratégies minimax mixtes dans chaque jeu d'état $G(\mathbf{s})$.

Kearns, Mansour et Singh [KEA 00] sont allés plus loin. Ils ont proposé un algorithme similaire à celui de Shapley (à savoir d'itération sur les valeurs) mais leur algorithme, appelé FiniteVI, a été conçu pour les jeux stochastiques à somme générale. Les auteurs ont montré que FiniteVI converge vers un équilibre de Nash *à horizon fini*. Ce qui concerne les jeux à horizon infini, il a été récemment montré [**?**] qu'il existe une classe de jeux stochastiques pour lesquels aucun algorithme d'itération sur les valeurs basé sur l'application directe de l'équation de Bellman (ce qui est fait, par exemple, dans FiniteVI) ne convergera pas vers une politique stationnaire.

La structure de l'algorithme FiniteVI est la même que celle de celui de Shapley. Il se différencie toutefois par le fait que (1) l'horizon T est utilisé comme critère d'arrêt ; (2) les fonctions *Équilibre* et *Valeur* sont définies différemment.

Tandis que l'équilibre minimax utilisé par l'algorithme de Shapley est assuré être unique, ce n'est plus le cas dans le cadre des jeux à somme générale ; par contre, il peut y avoir plusieurs équilibres possédants des valeurs différents. Dès lors, comme un seul équilibre doit être choisi à chaque itération, Kearns et ses collègues ont proposé d'utiliser une fonction $f(G(\mathbf{s}))$ choisissant un seul équilibre parmi plusieurs. Cependant, cette fonction $f(G(\mathbf{s}))$ n'a pas été définie, ce qui pose le problème d'application directe de FiniteVI. Il convient de noter également qu'à l'inverse du cas « à somme nulle », il n'existe pas d'algorithme polynomial trouvant un équilibre de Nash dans un jeu à somme générale.

Parmi d'autres algorithmes d'itération sur les valeurs pour les jeux stochastiques, le lecteur peut se référer à [POL 69, HOF 66, BOW 03a].

4.3.2.2. *Apprentissage par renforcement + Théorie des jeux classique*

Les algorithmes de cette catégorie combinent les techniques de recherche d'équilibre dans les jeux en forme normale avec les techniques d'apprentissage par renforcement comme le Q-learning vu dans la section 2.4.3 du chapitre 2.

L'idée sous-tendant les algorithmes de ce type est la suivante. Les agents réalisent un A/R en faisant des actions simultanées et en recevant des récompenses. Après chaque tour du jeu (ou après chaque séquence « action conjointe–transition ») les agents mettent à jour leurs fonctions Q qui associent des valeurs réelles à des paires « action conjointe–état » :

$$Q_i(\mathbf{s},\mathbf{a}) \leftarrow Q_i(\mathbf{s},\mathbf{a}) + \alpha\left(R_i(\mathbf{s},\mathbf{a}) + \gamma Valeur_i(\mathbf{s}') - Q_i(\mathbf{s},\mathbf{a})\right) \qquad (4.2)$$

où la fonction $Valeur_i$ retourne la valeur d'un équilibre d'un jeu G composé des valeurs-Q des agents dans l'état $\mathbf{s}'$. On voit bien ici que, pour pouvoir calculer cette

fonction, l'agent i a besoin d'observer les actions et les récompenses de tous les autres agents dans l'environnement ou que les autres agents doivent lui communiquer cette information.

Dans l'algorithme Minimax-Q de Littman [LIT 94], utilisant ce principe, la fonction $Valeur_i$ retourne au joueur i la valeur minimax d'un jeu composé des valeurs-Q des agents. La politique suivie par les agents est alors la politique de minimax en stratégies mixtes.

Quant à l'algorithme Nash-Q de Hu et Wellman [HU 03], la fonction $Valeur_i$ proposée par les auteurs retourne la valeur d'un équilibre de Nash du jeu d'état. Le fait d'avoir choisi le même équilibre par tous les agent est assuré par une politique de coordination qui prescrit aux agents de choisir toujours un équilibre prédéfini, par exemple le premier. La politique suivie par les agents dans ce cas est alors la politique alignée sur l'équilibre de Nash choisi.

Une définition plus formelle de l'algorithme de base pour les algorithmes Nash-Q et Minimax-Q est représenté par l'algorithme 4.2. Comme cela était le cas des algorithmes de Shapley et de Kearns, la différence entre les deux algorithmes réside dans la définition des fonctions $Equilibre_i$ et $Valeur_i$. Dans le cas de Minimax-Q, ces deux fonctions retournent à l'agent i respectivement la composante π_i de la politique d'équilibre minimax dans l'état $\mathbf{s}$ et la valeur de cet équilibre ; tandis que dans Nash-Q, elles retournent la politique d'un équilibre de Nash et sa valeur respective pour l'agent i.

Dans l'algorithme 4.2, le terme « une certaine exploration » fait référence aux différentes techniques d'exploration discutées au chapitre 2.

Il convient de noter que les preuves de convergence des deux précédents algorithmes se trouvent dans [LIT 94] pour le premier et dans [HU 03] pour le second. Dans le cas de ce dernier, la preuve est faite sous certains restrictions assez contraignantes.

4.3.2.3. *Apprentissage par renforcement + Modélisation de l'opposant*

Les algorithmes de cette catégorie combinent également deux parties provenant de deux champs de recherches distincts : celui de l'apprentissage monoagent et celui de la théorie des jeux dynamiques dont la technique de modélisation de l'opposant fait partie. Si tout est assez clair avec la première partie – un algorithme des « différences temporelles » est utilisé pour traiter le côté « plusieurs états » d'un jeu stochastique (voir la section 2.4 du chapitre 2) – il reste à préciser qu'est ce qu'on entend par « modélisation de l'opposant ».

La modélisation de l'opposant est une technique largement utilisée dans la théorie des jeux dynamiques pour rendre chaque joueur capable de s'adapter à des changements de la politique de ses adversaires [CLA 98, UTH 03].

Algorithme 4.2 : Algorithme de la base pour les algorithmes Minimax-Q et Nash-Q pour un joueur i

Pour tous $\mathbf{s}$, $\mathbf{a}$ et pour tout $j \in Ag$, initialiser $Q_j(\mathbf{s}, \mathbf{a})$ par des valeurs arbitraires.
Mettre dans $\mathbf{s}$ l'état courant.
Construire la matrice $G(\mathbf{s})$ à partir des valeurs Q_j dans $\mathbf{s}$, $\forall j \in Ag$.
Choisir une politique $\pi_i(\mathbf{s}) = Equilibre_i(G(\mathbf{s}))$.
répéter
  Jouer la politique $\pi_i(\mathbf{s})$ avec une certaine exploration.
  Observer l'action conjointe et la mettre dans $\mathbf{a}$.
  Observer le nouvel état et le mettre dans $\mathbf{s}'$.
  Construire la matrice $G(\mathbf{s}')$.
  Choisir une politique $\pi_i(\mathbf{s}') = Equilibre_i(G(\mathbf{s}'))$.
  Pour tout joueur j mettre à jour la valeur $Q_j(\mathbf{s}, \mathbf{a})$ en utilisant l'équation (4.2).
  $\mathbf{s} \leftarrow \mathbf{s}'$, $t \leftarrow t + 1$.
jusqu'à $t < T$
pour $\mathbf{s} \in \mathbf{S}$ **faire**
  Construire la matrice $G(\mathbf{s})$ comme précédemment.
  $\pi_i(\mathbf{s}) = Equilibre_i(G(\mathbf{s}))$.
retourner $\pi_i(\mathbf{s})$ $\forall \mathbf{s}$.

Le premier effort fait en direction de l'adaptation au comportement de l'adversaire dans le cadre d'un jeu répété est dû à Brown [BRO 51]. La technique utilisée sous le nom de « jeu fictif » (*fictitious play*) permet à un agent-joueur d'estimer la stratégie jouée par son adversaire afin de jouer la meilleure réponse à cette estimation.

Dans le jeu fictif, les joueurs jouant un jeu répété maintiennent des croyances empiriques individuelles sur les stratégies suivies par les autres joueurs. Rappelons que, selon notre notation, $\mathbf{A}_{-i}$ désigne l'espace fini de stratégies conjointes des adversaires du joueur i, avec $\mathbf{a}_{-i} \in \mathbf{A}_{-i}$. Dans ce cadre, le modèle du jeu fictif suppose que chaque joueur i choisit ses actions à chaque période pour maximiser son utilité espérée, $u_i^{\hat{\pi}_{-i}}$, étant donnée son estimation des politiques mixtes des adversaires, $\hat{\pi}_{-i}$. Cette estimation prend la forme suivante. On suppose que le joueur i a une fonction de poids initiale qui serait $c_i^0 : \mathbf{A}_{-i} \mapsto \mathbb{R}_+$. Ce poids est mis à jour en ajoutant 1 au poids de chacune des stratégies conjointes adverses, lorsque cette stratégie est jouée par ses adversaires :

$$c_i^t(\mathbf{a}_{-i}) = c_i^{t-1}(\mathbf{a}_{-i}) + \begin{cases} 1 & \text{si } \mathbf{a}_{-i}^{t-1} = \mathbf{a}_{-i} \\ 0 & \text{sinon} \end{cases}$$

Dan ces conditions, la probabilité estimée $(\hat{\pi}_{-i}^{\mathbf{a}_{-i}})^t$ que le joueur i assigne à ses adversaires de jouer une certaine $\mathbf{a}_{-i}$ à la date t est donnée par :

$$(\hat{\pi}_{-i}^{\mathbf{a}_{-i}})^t = \frac{c_i^t(\mathbf{a}_{-i})}{\sum_{\mathbf{a}'_{-i}} c_i^t(\mathbf{a}'_{-i})}$$

Dès lors, la meilleure action à jouer pour l'agent i est celle qui maximise son utilité espérée étant donnée son estimation de la politique adverse (en d'autres mots, sa meilleure réponse à cette politique) :

$$a_i^t = \underset{a_i}{\operatorname{argmax}} \sum_{\mathbf{a}_{-i}} R_i(a_i, \mathbf{a}_{-i})(\hat{\pi}_{-i}^{\mathbf{a}_{-i}})^t$$

Le jeu fictif converge vers un équilibre de Nash dans les jeux appelés « *iterated dominance solvable* », c'est-à-dire les jeux dans lesquels il est possible d'enlever les actions dominées de façon itérative pour obtenir à la fin une seule action ou un ensemble d'actions équivalentes [FUD 99].

La version du jeu fictif adaptée aux jeux stochastiques, étant donnée la fonction de transition $T(\mathbf{s}, \mathbf{a}, \mathbf{s}')$, est présentée sur l'algorithme 4.3, elle est due à Vrieze [VRI 87].

Algorithme 4.3 : Algorithme du jeu fictif dans les SG pour un joueur i

Pour tout $\mathbf{s}$, a_i, initialiser $q_i(\mathbf{s}, a_i) \leftarrow \frac{1}{|\mathbf{A}_{-i}|} \sum_{\mathbf{a}_{-i} \in \mathbf{A}_{-i}} R_i(\mathbf{s}, (a_i, \mathbf{a}_{-i}))$.
$n \leftarrow 0$.
répéter
  pour tous les $\mathbf{s}$ **faire**
    Choisir une action à faire comme étant $a_i^n = \operatorname{argmax}_{a'_i} \frac{q_i(\mathbf{s}, a'_i)}{n}$.
    Jouer l'action a_i^n.
    Observer l'action conjointe et la mettre dans $\mathbf{a}$.
    pour tous les a_i **faire**
      Mettre à jour la valeur $q_i(\mathbf{s}, a_i) \leftarrow$
      $q_i(\mathbf{s}, a_i) + R_i(\mathbf{s}, (a_i, \mathbf{a}_{-i})) + \gamma \sum_{\mathbf{s}' \in \mathbf{S}} T(\mathbf{s}, \mathbf{a}, \mathbf{s}') Valeur(\mathbf{s}')$
      ou $Valeur(\mathbf{s}') = \max_{a_i} \frac{q_i(\mathbf{s}, a_i)}{n}$
  $n \leftarrow n + 1$.
jusqu'à $n < N$
Pour tout $\mathbf{s}$, $\pi_i^{a_i}(\mathbf{s}) \leftarrow 1$ si $a_i = a_i^N$ et $\pi_i^{a_i}(\mathbf{s}) \leftarrow 0$ sinon.
retourner $\pi_i(\mathbf{s})$ $\forall \mathbf{s}$.

Il convient de noter que la version du jeu fictif pour les jeux stochastiques est basée sur la connaissance par tous les agents du modèle de transition $T(\mathbf{s}, \mathbf{a}, \mathbf{s}')$. De

plus, elle nécessite en pratique un mécanisme de coordination entre agents afin qu'ils puissent proposer une action à jouer pour chaque état s de façon synchronisée.

Un algorithme similaire proposé par Gies et Chaib-draa [GIE 06] est appelé « *Q-learning par jeu adaptatif* ». L'approche est basée sur la technique du jeu adaptatif pour les jeux répétés proposée par Young [YOU 93]. L'algorithme de Young est très similaire au jeu fictif. La différence réside dans la manière d'estimer la politique de l'opposant. Tandis que, dans le jeu fictif, tout l'historique est pris en compte, dans le jeu adaptatif l'agent fait un échantillonnage à partir d'un historique récent de taille limitée. Une telle modification permet de prouver la convergence du jeu adaptatif pour une plus large classe des jeux de coordination appelés « *weakly acyclic games* » [YOU 98].

Une autre approche proposée par Claus et Boutilier [CLA 98], appelée « *Joint Action Learners* » ou JALs, est basée sur le Q-learning et par conséquent ne dépend pas du modèle. De plus, comme c'est une technique essai-erreur, elle ne nécessite aucun mécanisme de coordination des agents. Notons que cette technique ressemble beaucoup à celle du jeu fictif à deux différences près. La première réside dans le fait que les Q-valeurs dans JALs sont associées aux paires « état–action conjointe » au lieu de « état–action simple » du jeu fictif. La deuxième différence réside dans l'équation utilisée pour mettre à jour les valeurs-Q. Claus et Boutilier utilisent la mise à jour usuelle du Q-learning (algorithme 4.4) au lieu d'une forme de l'équation de Bellman telle qu'utilisée dans le jeu fictif.

4.3.2.4. *Apprentissage par renforcement + Descente du gradient*

La technique de la descente du gradient a été tout d'abord utilisée dans l'apprentissage multiagent par Singh et al [SIN 94b]. L'algorithme IGA (pour *Infinitesimal Gradient Ascent*) fait une montée du gradient dans l'espace des politiques dans un jeu répété avec deux actions et deux joueurs. Le problème sous-tendu par un tel algorithme peut être représenté par deux matrices de récompenses pour les joueurs ligne et colonne, l and c, comme suit :

$$R_l = \begin{bmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{bmatrix}, \; R_c = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

Les joueurs l et c choisissent simultanément une action de l'ensemble $A_{l,c} = \{1, 2\}$, le joueur ligne l joue une action i et le joueur colonne c choisit une action j ; les récompenses obtenues sont alors l_{ij} et c_{ij} respectivement.

Comme il s'agit d'un jeu à deux actions, une stratégie mixte d'un joueur peut être représentée avec une seule valeur. Soit $\alpha \in [0, 1]$ une probabilité avec laquelle le joueur l choisit l'action 1 et $(1 - \alpha)$ une probabilité de jouer l'action 2. De façon similaire, soit $\beta \in [0, 1]$ et $(1 - \beta)$ les probabilités de jouer les actions 1 et 2 respectivement par le joueur c. L'utilité espérée d'une stratégie $\pi = (\alpha, \beta)$ peut être alors

Algorithme 4.4 : Algorithme JALs pour les SG pour un joueur i proposé par [CLA 98], adapté de [BOW 02a]

Pour tous $\mathbf{s}$ et $\mathbf{a}_{-i}$, initialiser $Q_i(\mathbf{s}, \mathbf{a}_{-i})$ arbitrairement, $c_i(\mathbf{s}, \mathbf{a}_{-i}) \leftarrow 0$ et $c(\mathbf{s}) \leftarrow 0$.
Initialiser $n \leftarrow 0$, $\mathbf{s} \leftarrow \mathbf{s}^0$.
répéter
  Dans l'état $\mathbf{s}$, choisir une action à faire comme étant $a_i^n = \text{argmax}_{a_i} \sum_{\mathbf{a}_{-i}} \frac{c_i(\mathbf{s},\mathbf{a}_{-i})}{c(\mathbf{s})} Q_i(\mathbf{s}, (a_i, \mathbf{a}_{-i}))$.
  Jouer l'action a_i^n avec un certain bruit d'exploration.
  Observer l'action conjointe des autres agents, $\mathbf{a}_{-i}$.
  Observer la récompense, $R_i(\mathbf{s}, (a_i, \mathbf{a}_{-i}))$.
  Observer le nouvel état, $\mathbf{s}'$.
  Mettre à jour la valeur $Q_i(\mathbf{s}, (a_i, \mathbf{a}_{-i})) \leftarrow (1-\alpha) Q_i(\mathbf{s}, (a_i, \mathbf{a}_{-i})) + \alpha \left(R(\mathbf{s}, (a_i, \mathbf{a}_{-i})) + \gamma Valeur(\mathbf{s}') \right)$
  ou $Valeur(\mathbf{s}') = \max_{a_i} \sum_{\mathbf{a}_{-i}} \frac{c_i(\mathbf{s}',\mathbf{a}_{-i})}{n(\mathbf{s}')} Q_i(\mathbf{s}', (a_i, \mathbf{a}_{-i}))$.
  Mettre à jour $c(\mathbf{s}) \leftarrow c(\mathbf{s}) + 1$, $c_i(\mathbf{s}, \mathbf{a}_{-i}) \leftarrow c_i(\mathbf{s}, \mathbf{a}_{-i}) + 1$, $n \leftarrow n + 1$.
jusqu'à $n < N$
Pour tout $\mathbf{s}$, $\pi_i^{a_i}(\mathbf{s}) \leftarrow 1$ si $a_i = \text{argmax}_{a_i} \sum_{\mathbf{a}_{-i}} \frac{c_i(\mathbf{s},\mathbf{a}_{-i})}{c(\mathbf{s})} Q_i(\mathbf{s}, (a_i, \mathbf{a}_{-i}))$ et $\pi_i^{a_i}(\mathbf{s}) \leftarrow 0$ sinon.
retourner $\pi_i(\mathbf{s})$ $\forall$s.

calculée comme suit :

$$u_l^{(\alpha,\beta)} = l_{11}\alpha\beta + l_{22}(1-\alpha)(1-\beta) + l_{12}\alpha(1-\beta) + l_{21}(1-\alpha)\beta$$

$$u_c^{(\alpha,\beta)} = c_{11}\alpha\beta + c_{22}(1-\alpha)(1-\beta) + c_{12}\alpha(1-\beta) + c_{21}(1-\alpha)\beta$$

Pour estimer l'effet d'un changement de sa politique courante, un joueur peut calculer une dérivée partielle de son utilité espérée par rapport à sa stratégie mixte :

$$\frac{\partial u_l^{(\alpha,\beta)}}{\partial \alpha} = \beta u - (l_{22} - l_{12})$$

$$\frac{\partial u_c^{(\alpha,\beta)}}{\partial \beta} = \alpha u' - (c_{22} - c_{21})$$

où $u = (l_{11} + l_{22}) - (l_{21} + l_{12})$ and $u' = (c_{11} + c_{22}) - (c_{21} + c_{12})$.

À chaque pas de temps, le joueur IGA ajuste sa stratégie courante dans la direction du gradient en vue de maximiser son utilité espérée :

$$\alpha_{t+1} = \alpha_t + \eta \frac{\partial u_l^{(\alpha_t,\beta_t)}}{\partial \alpha}$$

$$\beta_{t+1} = \beta_t + \eta \frac{\partial u_c^{(\alpha_t, \beta_t)}}{\partial \beta}$$

où η est la taille d'un pas, typiquement $0 < \eta \ll 1$. Évidemment, la stratégie mixte de l'opposant est supposée être connue par les joueurs.

Singh et ses collègues ont prouvé la convergence de IGA vers un équilibre de Nash (où vers une valeur moyenne équivalente) en *self-play* (c'est-à-dire, quand les deux joueurs utilisent le même algorithme) et ce dans le cas où le pas η tend vers 0 ($\lim_{\eta \to 0}$), d'où le nom de l'algorithme. L'algorithme IGA ne peut pas être appliqué à un grand nombre de problèmes du monde réel pour les deux raisons principales :

1) il suppose une connaissance omnisciente des stratégies courantes d'autrui et

2) il a été conçu pour le cas « deux agents–deux actions » ; une extension au cas « plusieurs agents–plusieurs actions » est loin d'être évidente.

Le premier algorithme « pratique » héritant (en pratique) des propriétés de convergence de IGA est l'algorithme du *Policy Hill Climbing* (PHC) proposé par Bowling et Veloso [BOW 02a]. Cet algorithme n'exige pas la connaissance de la politique courante de l'adversaire. Essentiellement, PHC réalise un « *hill-climbing* » dans l'espace des stratégies mixtes. De ce fait, c'est une simple modification du *Q*-learning monoagent. Il comporte deux parties ; la première est basée sur le *Q*-learning en vue de maintenir les valeurs des actions simples (et non des actions conjointes) dans les états. La deuxième est une partie de théorie des jeux qui maintient la stratégie mixte courante dans chacun des états du système.

Dans PHC, la politique courante est ajustée via une augmentation de la probabilité de choisir l'action ayant la valeur la plus élevée. Ceci est fait en utilisant un petit pas δ. Les valeurs-Q sont mises à jour en utilisant le facteur d'apprentissage α. Notons que si δ égale à 1, l'algorithme devient équivalent au Q-learning monoagent, car l'agent exécutera toujours l'action dont la valeur est la plus élevée. Par conséquent, cette technique est « rationnelle » et converge vers la solution optimale si les autres joueurs suivent une politique stationnaire. Toutefois, si les autres joueurs sont en train d'apprendre leurs politiques, le PHC peut ne pas converger vers une politique stationnaire bien que sa récompense moyenne converge vers la récompense d'un équilibre de Nash, comme cela a été montré par Bowling et Veloso [BOW 02a].

La définition formelle de l'algorithme PHC est présentée via l'algorithme 4.5. Dans cet algorithme (et ceux qui suivent plus bas), l'expression « une certaine exploration » fait référence aux différentes techniques d'exploration des espaces d'états ou d'actions, telles que ϵ-greedy et autres (voir le chapitre 2).

Bowling et Veloso [BOW 02a] ont développé une extension importante des algorithmes IGA et PHC, appelée respectivement WoLF-IGA et WoLF-PHC (le dernier est

Algorithme 4.5 : Algorithme PHC pour les SG pour un joueur i, adapté de [BOW 02a]

Initialiser $\alpha \in (0,1]$, $\delta \in (0,1]$, $\gamma \in (0,1)$.
Pour tous $\mathbf{s} \in \mathbf{S}$ et tous $a_i \in A_i$, initialiser $q_i(\mathbf{s}, a_i) \leftarrow 0$.
Pour tous $a_i \in A_i$, initialiser la politique courante $\pi_i^{a_i}(\mathbf{s}) \leftarrow \frac{1}{|A_i|}$.
L'état courant $\mathbf{s} \leftarrow \mathbf{s}^0$.
répéter
  Dans l'état $\mathbf{s}$, choisir une a_i selon la stratégie $\pi_i(\mathbf{s})$.
  Jouer a_i avec une certaine exploration.
  Observer le nouvel état $\mathbf{s}'$ et la récompense obtenue R_i.
  Mettre à jour $q_i(\mathbf{s}, a_i)$ en utilisant la règle suivante :

$$q_i(\mathbf{s}, a_i) \leftarrow (1-\alpha) q_i(\mathbf{s}, a_i) + \alpha \left(R_i + \gamma \max_{a_i'} q_i(\mathbf{s}', a_i') \right).$$

  Mettre à jour la stratégie courante, $\pi_i(\mathbf{s})$, en utilisant la règle suivante :

$$\pi_i^{a_i}(\mathbf{s}) \leftarrow \pi_i^{a_i}(\mathbf{s}) + \Delta_{\mathbf{s}a_i},$$

  où

$$\Delta_{\mathbf{s}a_i} = \begin{cases} -\delta_{\mathbf{s}a_i} & \text{si } a_i \neq \operatorname{argmax}_{a_i'} q_i(\mathbf{s}, a_i') \\ \sum_{a_i' \neq a_i} \delta_{\mathbf{s}a_i'} & \text{sinon ;} \end{cases}$$

$$\delta_{\mathbf{s}a_i} = \min \left(\pi_i^{a_i}(\mathbf{s}), \frac{\delta}{|A_i| - 1} \right),$$

  en se limitant à la distribution de probabilité légale.
  Mettre à jour l'état courant $\mathbf{s} \leftarrow \mathbf{s}'$.
  $n \leftarrow n + 1$.
jusqu'à $n < N$
retourner $\pi_i(\mathbf{s})$ $\forall \mathbf{s}$.

illustré par l'algorithme 4.6). WoLF (pour *Win or Learn Fast*) est une technique selon laquelle un agent i change le pas d'ajustement de la politique δ selon qu'il « gagne » ou qu'il « perd ». S'il gagne, son δ est petit, s'il perd – il est grand. Bowling a montré de façon formelle qu'une telle alternance assure la convergence de WoLF-IGA en self-play vers un équilibre de Nash dans le cas où il y a deux joueurs qui ont deux actions chacun. En pratique, la convergence de WoLF-PHC a aussi été observée dans plusieurs jeux « problématiques ». Notons que le principe WoLF aide la convergence en donnant aux autres joueurs plus de temps pour s'adapter aux changements de stratégie du joueur considéré. Réciproquement, il permet à un joueur de s'adapter plus

rapidement aux changements des stratégies des autres joueurs quand ces changements sont défavorables.

D'une façon plus formelle, l'algorithme WoLF-PHC exige d'utiliser deux valeurs du pas δ : δ_{perdre} et δ_{gagner}. Le fait de gagner ou perdre est déterminé en comparant la valeur espérée de la politique courante, π, avec la valeur espérée d'une « politique moyenne », $\bar{\pi}$ (voir l'algorithme 4.6). Si la valeur espérée de π est inférieure à celle de $\bar{\pi}$, alors le joueur est considéré comme perdant et la valeur de δ_{perdre} est utilisée, sinon – δ_{gagner} est utilisée pour ajuster sa politique. La politique moyenne d'un joueur i est une moyenne « en ligne » de toute les politiques jouées par i dès le début de l'apprentissage. Au demeurant, l'algorithme WoLF-PHC est identique à PHC (voir l'algorithme 4.5).

4.3.3. *Complexité et extensibilité des algorithmes d'apprentissage multiagent*

Cette section fait état de la complexité informatique de certains algorithmes d'apprentissage multiagent et leurs extensibilité (ou *scalability*) à des problèmes de grande taille.

Comme on l'a précisé plus haut, l'algorithme de Shapley [SHA 53] pour les jeux stochastiques, qui s'appuie sur la technique d'itération sur les valeurs, utilise l'algorithme minimax pour trouver l'équilibre dans chaque état et à chaque itération. Puisque le temps d'exécution de l'algorithme d'itération sur les valeurs est lui-même polynomial en la taille de l'espace d'états, alors, le temps d'exécution de l'algorithme de Shapley est polynomial en le nombre de joueurs et le nombre d'états du jeu stochastique.

Le même raisonnement peut être appliqué pour conclure sur le temps d'exécution de l'algorithme de Kearns et al. [KEA 00]. Le temps d'exécution de l'algorithme de Kearns est quadratique en la taille de l'espace d'états en supposant que le temps d'exécution de la fonction f (censée trouver un équilibre de Nash) est unitaire. Cependant, comme on ne connaît pas d'algorithme polynomial trouvant un équilibre de Nash dans un jeu matriciel, le temps d'exécution de l'algorithme de Kearns, utilisant un des algorithmes connus permettant calculer un Nash, ne peut être polynomial non plus.

L'analyse du temps d'exécution des algorithmes basées sur le Q-learning est plus compliqué. On sait [KOE 96] que le temps d'exécution du Q-learning peut être exponentiel en la taille de l'espace d'états, mais ce temps peut être réduit à un polynôme si certaines restrictions sur le modèle de récompense sont appliquées [KOE 96]. De plus, selon le modèle des jeux stochastiques, la taille de l'espace d'états est elle-même exponentielle en le nombre d'agents (car $|\mathbf{S}| = |S_1 \times S_2 \times \ldots \times S_{|Ag|}| = |S_i|^{|Ag|}$, si tous les S_i ont la même taille). Par conséquent, le temps d'exécution d'un tel algorithme

Algorithme 4.6 : Algorithme WoLF-PHC pours les SG pour un joueur i, adapté de [BOW 02a].

Initialiser $\alpha \in (0,1]$, $\delta \in (0,1]$, $\gamma \in (0,1)$.
Pour tous $\mathbf{s} \in \mathbf{S}$ et tous $a_i \in A_i$, initialiser $q_i(\mathbf{s}, a_i) \leftarrow 0$.
Initialiser le compteur $c(\mathbf{s}) \leftarrow 0$.
Pour tous $a_i \in A_i$, initialiser la politique courante $\pi_i^{a_i}(\mathbf{s}) \leftarrow \frac{1}{|A_i|}$.
L'état courant $\mathbf{s} \leftarrow \mathbf{s}^0$.
répéter
Dans l'état $\mathbf{s}$, choisir une a_i selon la stratégie $\pi_i(\mathbf{s})$.
Jouer a_i avec une certaine exploration.
Observer le nouvel état $\mathbf{s}'$ et la récompense obtenue R_i.
Mettre à jour $q_i(\mathbf{s}, a_i)$ en utilisant la règle suivante :

$$q_i(\mathbf{s}, a_i) \leftarrow (1-\alpha) q_i(\mathbf{s}, a_i) + \alpha \left(R_i + \gamma \max_{a_i'} q_i(\mathbf{s}', a_i') \right).$$

Mettre à jour l'estimation de la politique moyenne, $\bar{\pi}(\mathbf{s})$, comme suit :

$$c(\mathbf{s}) \leftarrow c(\mathbf{s}) + 1,$$
$$\bar{\pi}_i^{a_i'}(\mathbf{s}) \leftarrow \bar{\pi}_i^{a_i'}(\mathbf{s}) + \frac{1}{c(\mathbf{s})} (\pi_i^{a_i'}(\mathbf{s}) - \bar{\pi}_i^{a_i'}(\mathbf{s})),\ \forall a_i' \in \mathcal{A}_i.$$

Mettre à jour la stratégie courante, $\pi_i(\mathbf{s})$, en utilisant la règle suivante :

$$\pi_i^{a_i}(\mathbf{s}) \leftarrow \pi_i^{a_i}(\mathbf{s}) + \Delta_{\mathbf{s}a_i},$$

où

$$\Delta_{\mathbf{s}a_i} = \begin{cases} -\delta_{\mathbf{s}a_i} & \text{si } a_i \neq \operatorname{argmax}_{a_i'} q_i(\mathbf{s}, a_i') \\ \sum_{a_i' \neq a_i} \delta_{\mathbf{s}a_i'} & \text{sinon} \end{cases},$$

$$\delta_{\mathbf{s}a_i} = \min \left(\pi_i^{a_i}(\mathbf{s}), \frac{\delta}{|A_i| - 1} \right),$$

$$\delta = \begin{cases} \delta_{gagner} & \text{si } \sum_{a_i'} \pi_i^{a_i'}(\mathbf{s}) q_i(\mathbf{s}, a_i') > \sum_{a_i'} \bar{\pi}_i^{a_i'}(\mathbf{s}) q_i(\mathbf{s}, a_i') \\ \delta_{perdre} & \text{sinon} \end{cases},$$

en se limitant à la distribution de probabilité légale.
Mettre à jour l'état courant $\mathbf{s} \leftarrow \mathbf{s}'$.
$n \leftarrow n + 1$.
jusqu'à $n < N$
retourner $\pi_i(\mathbf{s})$ $\forall \mathbf{s}$.

comme Nash-Q n'est pas polynomial en la taille de la matrice du jeux d'état (en raison du fait que la procédure utilisée pour calculer un équilibre de Nash est elle-même non polynomiale) ; de plus, ce temps peut être exponentiel en le nombre d'états (en raison de la structure de la fonction de récompense utilisée dans la partie « Q-learning » de cet algorithme).

Comme on peut le constater, l'application directe aux problèmes de grande taille des algorithmes de planification ou d'apprentissage multiagents basés sur le modèle des jeux stochastiques est problématique à cause de leur complexité élevée. Une exception est l'algorithme de Shapley qui a un temps d'exécution polynomial mais qui ne s'applique qu'à des jeux à somme nulle.

Un bon candidat à une application dans les problèmes de grande taille est l'algorithme WoLF-PHC de Bowling [BOW 02a]. Vu sa simplicité structurelle, ses conditions de convergence tolérantes et ses exigences modestes relativement à l'information provenant de l'environnement (en fait, les agents WoLF-PHC doivent percevoir seulement l'état courant et leurs propres récompenses), les techniques connues d'approximation de la fonction peuvent être directement appliquées à cet algorithme. Dans ce contexte, Bowling [BOW 02b] a proposé de combiner : (i) une technique appelée *Tile Coding* [SUT 98] pour généraliser la fonction de valeur, (ii) une méthode de montée du gradient de la politique [SUT 00] comme méthode d'apprentissage de base (au lieu du Q-learning) et (iii) le principe WoLF pour favoriser la convergence vers un équilibre de Nash. Cette méthode a permis d'obtenir un algorithme d'apprentissage multiagent facilement extensible appelé GraWoLF [BOW 02b]. Bowling a montré via des expérimentations que GraWoLF peut être utilisé pour faire de l'apprentissage multiagent dans les problèmes de très grande taille, comme le jeu de cartes Goofspiel [BOW 02b] et l'entraînement des robots compétitifs [BOW 03b] de type RoboCup [KIT 97].

4.3.4. *Au-delà de la recherche d'équilibre*

Ces derniers temps, les chercheurs s'interrogent sur la nécessité de la convergence d'un algorithme d'apprentissage vers un équilibre de Nash. Il existe plusieurs raisons à cela. Tout d'abord, il peut y avoir plusieurs équilibres dans un jeu et il peut ne pas y avoir de méthode de coordination des choix des agents. Ensuite, la complexité de calcul d'équilibre de Nash est peu étudiée et on ne connaît pas d'algorithme polynomial pouvant calculer un tel équilibre.

La troisième raison est plus philosophique. Certains auteurs [SHO 04] attirent l'attention sur le fait que, dans plusieurs cas, comme par exemple dans le dilemme du prisonnier, jouer l'équilibre de Nash peut être catastrophique pour les agents, tandis que l'action coopérative, bien qu'elle ne soit pas un équilibre, serait un choix plus « judicieux ». Autrement dit, l'action coopérative est le seul choix rationnel, comme on l'a vu dans le cas du dilemme du prisonnier répété.

4.3.4.1. *Jeu efficace*

Certains travaux récents ont mis l'accent sur le fait de jouer « plus efficacement » *contre un certain autre type de joueur* et non pas le fait de converger ver une valeur où une politique stable, comme on le fait usuellement. Parmi ces approches, il convient de citer les travaux suivants [CHA 01, TES 04, POW 05b, POW 05a].

Chang et Kaelbling [CHA 01] ont été les premiers à proposer une méthode permettant à un joueur d'exploiter l'algorithme d'apprentissage de son adversaire. En particulier, leur algorithme appelé « PHC-Exploiter » peut l'emporter plus de fois que son adversaire dans les jeux à somme nulle, tels que Pierre-Papier-Ciseaux et ce si l'adversaire utilise l'algorithme PHC [BOW 02a]. En fait, PHC-Exploiter est capable d'estimer la valeur du pas d'apprentissage δ de l'algorithme PHC de l'adversaire et changer sa politique afin d'exploiter cette connaissance.

Tesauro [TES 04] est allé plus loin. Il a proposé une technique qui semble être plus générale que celle de Chang et Kaelbling. En effet, l'algorithme Hyper-Q de Tesauro peut jouer de façon plus efficace dans les jeux à somme nulle contre un joueur sans connaître l'algorithme sous-tendu par ce dernier. Par exemple, Tesauro a montré via les expérimentations que l'algorithme Hyper-Q est plus efficace que PHC et IGA dans le jeu Pierre-Papier-Ciseaux. L'idée sous-tendant l'approche de Tesauro est la suivante. L'agent Hyper-Q discrétise l'espace des stratégies de son adversaire de façon uniforme. Il associe ensuite une valeur-Q à chacune de ses actions simples et à chacune des valeurs discrètes de la stratégie adverse. Pour estimer la stratégie de son adversaire, il utilise une technique d'estimation de distribution de probabilité. Finalement, en interagissant avec l'adversaire, l'agent Hyper-Q apprend des valeurs-Q de chaque paire « stratégie de l'adversaire–action simple ». Il convient de noter qu'une extension de cet algorithme aux jeux stochastiques n'a pas été proposée. Une approche similaire à celle de Tesauro, appelé « ADL » (pour *Adaptive Dynamics Learning*) a été proposée par Burkov et Chaib-draa [BUR 07]. La différence de cette dernière par rapport à Hyper-Q réside dans la manière d'assigner des valeurs-Q ; dans ADL, ces dernières sont assignées à des historiques de taille limitée à la place des distributions de probabilités. Alors les valeurs-Q de ADL ont la forme « historique du jeu–action simple ». Pour apprendre ces valeurs, une règle standard de mis à jour du Q-learning est utilisée.

Un autre point de vue sur l'apprentissage dans les jeux répétés a été proposée par Powers et Shoham [POW 05b, POW 05a]. Leur méthode consiste en une classification des adversaires relativement à différentes classes. Leurs agents, appelés MetaStrategy [POW 05b] et Manipulator [POW 05a] sont programmés en vue d'être capable de jouer différentes politiques (par exemple, telles comme TFT pour le dilemme du prisonnier et autres). En jouant avec un adversaire dont la classe (à savoir, son algorithme d'apprentissage ou sa politique, si elle est fixe) est inconnue, le joueur tente d'estimer cette classe selon ses observations et ensuite il choisit la meilleure politique préprogrammée, dont il dispose, pour jouer contre cette classe d'adversaire.

4.3.4.2. *Minimisation du regret*

Une autre direction de recherche [HAR 00, AUE 95, ZIN 03] vise à apporter des réponses à la question suivante. Étant donné un historique du jeu, dans quelle mesure la politique exécutée par l'agent (par un algorithme d'apprentissage ou celui réalisant une politique fixe) pourrait-elle être améliorée ? Plus précisément, il s'agit de la notion de « regret » mesurant jusqu'à quel degré la performance observée d'un algorithme est pire que la meilleure stratégie pure.

Décrivons cette notion de regret de façon plus formelle et limitons-nous au cas des jeux répétés. Soit $\mathbf{r}_i^t \in \mathbb{R}^{|A_i|}$ le vecteur des récompenses que le joueur i pourrait obtenir au temps t sachant les choix faits au même temps par les autres joueurs. Soit π_i^t la stratégie adaptée par le joueur i au temps t. L'utilité espérée u_i^t de stratégie π_i^t est alors la suivante :

$$u_i^t = \sum_{a_i \in A_i} \pi_i^{a_i,t} r_i^{a_i,t}$$

où $r_i^{a_i,t}$ désigne la récompense de i pour l'action a_i selon $\mathbf{r}_i^t$. En utilisant le produit scalaire de deux vecteurs, on peut aussi écrire :

$$u_i^t = \pi_i^t \cdot \mathbf{r}_i^t$$

Au pas de temps t, le regret $\mathcal{R}_i^t$ de l'agent i pour avoir joué une politique π_i^t au lieu d'une action simple a_i est la différence entre les récompenses de ces deux stratégies, étant donné le choix des stratégies des adversaires :

$$\mathcal{R}_i^t(\pi_i^t, a_i) = r_i^{a_i,t} - u_i^t$$

En désignant par 1_{a_i} une politique de l'agent i dans laquelle la probabilité de 1 est assignée à une action a_i, le regret total $\mathcal{R}_i^T$ de l'agent i pour une séquence de tours du jeu de la taille T s'écrit comme suit :

$$\mathcal{R}_i^T = \max_{a_i \in A_i} \sum_{t=1}^{T} \left((\mathbf{r}_i^t \cdot 1_{a_i}) - (\mathbf{r}_i^t \cdot \pi_i^t) \right)$$

Le regret moyen $\bar{\mathcal{R}}_i$ du même algorithme s'écrit alors comme

$$\bar{\mathcal{R}}_i = \lim_{T \to \infty} \frac{1}{T} \mathcal{R}_i^T$$

Pour un algorithme donné, la propriété de ne pas avoir de regret (*no-regret*) dans le sens du regret moyen signifie que la récompense moyenne que l'agent obtient en utilisant cet algorithme est au moins aussi grande que ce que peut lui apporter une stratégie pure fixe.

Zinkevich [ZIN 03] a proposé une extension de l'algorithme IGA [SIN 94b] aux jeux admettant plus de deux actions et deux joueurs, à savoir aux jeux à somme générale sans aucune contrainte. Ledit auteur a prouvé que son algorithme, appelé GIGA (pour *Generalized Infinitesimal Gradient Ascent*), n'a pas de regret dans les jeux en forme normale. L'algorithme GIGA est similaire à IGA et PHC. La mise à jour de la politique de l'agent i se fait comme suit :

$$\pi_i^{t+1} = P(\pi_i^t + \eta \mathbf{r}_i^t)$$

La fonction $P(\tilde{\pi}_i)$ est une fonction qui réduit la politique $\tilde{\pi}_i$ résultante de la sommation $(\pi_i^t + \eta \mathbf{r}_i^t)$ à une distribution de probabilité légale :

$$P(\tilde{\pi}_i) = \underset{\pi_i \in \Delta A_i}{\operatorname{argmin}} ||\tilde{\pi}_i - \pi_i||$$

où l'opérateur $||\cdot||$ est la norme euclidienne usuelle. Zinkevich a montré que le regret total de GIGA est borné par :

$$\mathcal{R}_i^T \leq \frac{\left(\max_{\pi_i, \pi_i'} ||\pi_i - \pi_i'||\right)^2 \sqrt{T}}{2} + \left(\sqrt{T} - \frac{1}{2}\right) \left(\sup_{\pi_i, t=1\ldots T} ||\mathbf{r}_i^t||\right)^2$$

ou, en supposant que tous les récompenses $r_i^{a_i}$ du joueur i sont bornées par r_i^{max} et en tenant compte que $\max_{\pi_i, \pi_i'} ||\pi_i - \pi_i'|| = \sqrt{2}$, on peut écrire :

$$\mathcal{R}_i^T \leq \sqrt{T} + \left(\sqrt{T} - \frac{1}{2}\right) |A_i| (r_i^{max})^2$$

En utilisant la règle de l'Hôpital[5], on peut trouver que

$$\lim_{T \to \infty} \frac{1}{T} \left(\sqrt{T} + \left(\sqrt{T} - \frac{1}{2}\right) |A_i| (r_i^{max})^2\right) = 0,$$

ce qui nous amène à conclure que $\bar{\mathcal{R}}_i \leq 0$ et, donc, GIGA n'a pas de regret au sens du regret moyen.

Quant à lui, Bowling [BOW 04] a montré que le principe WoLF appliqué à GIGA fait converger ce dernier vers un équilibre en *self-play*. Son WoLF-GIGA hérite alors les propriétés du *no-regret* de GIGA et converge vers un équilibre de Nash en stratégies mixtes dans les jeux à somme générale à deux joueurs–deux actions.

---

5. Cette règle est définie comme suit : si f et g sont deux fonctions dérivables en a, s'annulant en a et telles que le quotient $\frac{f'(a)}{g'(a)}$ soit défini, alors $\lim_{x \to a} \frac{f(x)}{g(x)} = \frac{f'(a)}{g'(a)}$.

	Connu ou observé				Type	Tous	Propriétés
Algorithme	$\mathbf{S}$	R_i	$\mathbf{A}_{-i}$	$\mathbf{R}_{-i}$	équ.	jeux	théoriques
[SHA 53]	connu	connu	connu	connu	Nash	non	oui
[KEA 00]	connu	connu	connu	connu	Nash	oui	oui
Minimax-Q	connu	connu	observé	observé	Nash	non	oui
Nash-Q	connu	connu	observé	observé	Nash	non	oui
Jeu fictif	connu	connu	observé	non	Nash	non	oui
[GIE 06]	observé	observé	observé	non	Nash	non	non
JALs	observé	observé	observé	non	Nash	non	non
WoLF-PHC	observé	observé	observé	non	Nash	non	oui
GIGA-WoLF	observé	observé	observé	non	Nash	non	oui
Hyper-Q	observé	observé	observé	non	?	?	non
ADL	observé	observé	observé	non	?	?	non
Manipulator	observé	observé	non	non	?	?	oui
MetaStrategy	observé	observé	non	non	?	?	oui

Tableau 4.1. *Comparaison des algorithmes évoqués.*

4.3.5. *Discussion*

Il est intéressant de voir comment les différents algorithmes introduits précédemment peuvent être comparés entre eux. À cet effet, le tableau récapitulatif 4.1 présente les différentes caractéristiques des algorithmes d'apprentissage ou de planification multiagent vus dans ce chapitre. Ce tableau est inspiré des travaux d'Aras [DUT 06].

Dans ce tableau, tous les algorithmes introduits dans ce chapitre sont situés relativement à des exigences concernant l'observabilité de certaines propriétés du jeu. La colonne $\mathbf{S}$ indique si les états de l'environnement sont supposés connus par les agents ou si les agents n'en ont qu'une perception partielle ; la colonne R_i indique si les agents connaissent leurs récompenses ou bien s'ils sont capables de les observer en interagissant avec les autres. Les colonnes $\mathbf{A}_{-i}$ et $\mathbf{R}_{-i}$ indiquent si les agents sont supposés capables de connaître par avance ou d'observer les actions pouvant être faites par les autres joueurs. « Non » dans les cellules signifie que les agents ne peuvent pas connaître ni observer certaines propriétés du jeu. Les colonnes « Type équi. » et « Tous jeux » indiquent si l'équilibre de Nash est atteint par chacun des algorithmes et si ce fait est valide pour n'importe quel jeu, ou bien il existe certaines restrictions (comme par exemple le fait que le jeu soit à somme nulle, ou que seulement deux actions–deux joueurs soient permises). La colonne « Propriétés théoriques » indique si oui ou non certaines propriétés théoriques des algorithmes sont prouvées.

Il convient de noter que les points d'interrogation dans les cellules du tableau 4.1 signifient que cet aspect n'est pas bien étudié et/ou compris dans la littérature. Comme certains des algorithmes évoqués sont assez récents, il reste beaucoup de travail à faire en vue d'expliquer leur comportement dans différentes situations et d'étudier leurs propriétés de convergence, leur complexité et ainsi de suite.

	$\mathcal{B}_0$	$\mathcal{B}_1$	$\mathcal{B}_\infty$
$\mathcal{H}_0$	Minimax-Q, Nash-Q		Bully
$\mathcal{H}_1$			Godfather
$\mathcal{H}_\infty$	Q-learning, (WoLF)-PHC, Jeu fictif	ADL avec $\|h\| = 1$	Poids multiplicatif

Tableau 4.2. *Classification des algorithmes proposée par Chang et Kaelbling et adaptée de [CHA 01] pour les algorithmes présentés dans ce chapitre.*

Une autre classification intéressante des algorithmes jouant aux jeux (répétés ou stochastiques) a été proposée par Chang et Kaelbling [CHA 01]. Ces auteurs ont proposé une classification des algorithmes via le produit cartésien des stratégies possibles et des croyances possibles sur les stratégies des adversaires. Dans ce contexte, les stratégies possibles d'un agent ont été classées selon la longueur de l'historique gardé en mémoire, de $\mathcal{H}_0$ jusqu'à $\mathcal{H}_\infty$. Évidemment, en ayant plus de mémoire, les agent peuvent élaborer des politiques plus complexes et plus « astucieuses ». Dans le même ordre d'idée, un agent peut classer ses adversaires selon le même principe. S'il croit que son adversaire n'a pas de mémoire, il le classe dans $\mathcal{B}_0$. S'il croit que la mémoire de son adversaire est illimitée alors il le classe dans $\mathcal{B}_\infty$. La classification proposée par Chang et Kaelbling, adaptée pour refléter certains algorithmes évoqués dans ce chapitre, est présentée par le tableau 4.2. Notons que les algorithmes dénotés comme « Bully », « Godfather » et « Poids multiplicatif » n'ont pas été discutés dans ce chapitre. Pour plus de détails, le lecteur peut se référer aux ouvrages suivants [LIT 01, FRE 99].

4.4. Conclusion et perspectives

Nous avons présenté dans ce chapitre un survol de la théorie des jeux et son application à la prise de décision dans les environnements multiagents. À cet effet, les jeux en forme normale et les jeux dynamiques comme modèles d'interaction entre les agents rationnels ont été tout d'abord discutés. Ces modèles ont ensuite été étendus au monde multi-état en les combinant avec les MDP. Ceci a alors fait émerger un modèle plus puissant, appelé « jeux stochastiques », qui permet de décrire des interactions complexes du monde réel étendu dans le temps.

Plusieurs algorithmes de planification et d'apprentissage utilisant le formalisme des jeux stochastiques ont été présentés et discutés. On a ainsi pu constater que certains d'entre eux possèdent une bonne base théorique tandis que d'autres ne sont encore étudiés que via des expérimentations. Comme ce domaine de recherches est assez jeune et en période de croissance rapide, on peut s'attendre à ce que, dans un proche avenir, ces algorithmes soient convenablement analysés d'un point de vue théorique et que d'autres approches intéressantes soient proposées.

Hormis les assises théoriques manquantes, il existe d'autres problèmes qui doivent être résolus afin de permettre à ces algorithmes d'être appliqués à des problèmes du monde réel. Tout d'abord, la complexité des algorithmes reste assez élevée. En fait, ce problème est caché dans le modèle même des jeux stochastiques dont le nombre d'états croît exponentiellement avec le nombre d'agents. De plus, certains algorithmes exigent de percevoir des actions conjointes de tous les autres joueurs, ce qui complique le processus d'apprentissage. À cela s'ajoute le problème induit par la multitude des équilibres dans un jeu et pour lequel la coordination sur le choix d'un seul équilibre n'est pas évidente pour le moment.

Il convient de noter que ce chapitre n'a pas abordé une autre branche de recherches dans le domaine de la décision dans l'incertain dans les environnements multiagents que sont les jeux stochastiques partiellement observables (POSG, pour *Partially Observable Stochastic Games*). Dans ce modèle, les jeux en forme normale sont combinés avec le modèle des POMDP ce qui permet, en utilisant ce formalisme, de résoudre des problèmes où les agents ont une vue partielle de l'état de l'environnement. De bons exemples de cette problématique et des approches proposées pour la résolution de celle-ci se trouvent dans [HAN 04, EME 04].

Une autre problématique similaire à celle modélisée par les POSG, est appelée DEC-POMDP (pour *Decentralized* POMDP ou POMDP décentralisés), qui fait objet du chapitre 8. La différence entre les DEC-POMDP et les POSG réside dans le fait que les agents dans les DEC-POMDP sont censés être coopératifs et les politiques, exécutées par les agents de façon distribuée, sont calculées de façon centralisée [BER 02]. Alors que dans les POSG, on ne fait pas une telle hypothèse : bien que les agents puissent en général être coopératifs (avoir des récompenses identiques par exemple), cette coopération peut seulement être désirée (par exemple, l'algorithme utilisé peut la rapporter sous certaines conditions) et non pas assurée. Le cadre POSG est donc plus général.

Remerciement : les auteurs tiennent à remercier M. Bowling pour l'aide qu'il a prodiguée relativement à la partie technique du gradient de la politique.

Chapitre 5

Critères non classiques

5.1. Introduction

La modélisation et la résolution d'un problème de décision séquentielle dans l'incertain par un MDP classique imposent certaines hypothèses fortes : problème mono-critère, connaissance complète et précise de l'environnement à tout instant, connaissance du modèle lui-même, cadre d'incertitude probabiliste bien défini...

Parmi ces hypothèses, nous avons vu que certaines pouvaient être relâchées. Les POMDP permettent de prendre en compte la connaissance partielle de l'environnement. Les méthodes d'apprentissage par renforcement permettent de se passer de la connaissance du modèle lui-même. Nous allons nous intéresser plus particulièrement dans ce chapitre aux deux autres limitations, celles d'un cadre mono-critère et d'une représentation probabiliste bien définie de l'incertain.

Plus précisément, nous allons commencer par décrire le formalisme des MDP multicritère ou 2V-MDP [**?**, **?**, **?**] étendant le cadre MDP à la décision multicritère. Dans ce cadre, nous présentons un algorithme [MOU 04] permettant de calculer des politiques *satisfaisantes*, c'est-à-dire aussi proches que possible d'un point *idéal*.

Ensuite, nous décrirons une première approche pour la résolution de MDP dont le modèle est mal connu. Dans cette approche dite *robuste* [**?**, BAG 01, NIL 04, NIL 05], on reste dans le cadre probabiliste classique des MDP, à la différence près que le modèle des fonctions de transition et de récompense du MDP est mal connu. Grâce à

Chapitre rédigé par Matthieu BOUSSARD et Maroua BOUZID et Abdel-Illah MOUADDIB et Régis SABBADIN et Paul WENG.

certaines hypothèses (connaissance d'un intervalle de probabilités de transition et de récompenses), il est possible de proposer une version robuste de l'algorithme d'itération sur les valeurs.

Mais la spécification même d'intervalles de probabilité pour la fonction de transition d'un MDP peut être impossible. De même, il peut être particulièrement arbitraire de spécifier des récompenses numériques, additives, pour certains MDP. Dans certains cas, les connaissances et les préférences d'un agent s'expriment plus fidèlement par un préordre sur les vraisemblances des transitions et sur ses préférences sur ces transitions. La *Théorie des Possibilités* [DUB 88] offre un cadre permettant de représenter de telles connaissances et préférences *qualitatives*. Nous présenterons donc dans ce chapitre une approche des MDP basée sur la théorie des possibilités [SAB 98]. Nous verrons que cette approche permet également de prendre en compte l'observabilité partielle des états (ou du modèle) dans les MDP possibilistes et qu'elle permet la résolution de problèmes représentés de manière concise.

Enfin, nous présenterons le cadre des MDP algébriques [PER 05, WEN 06b], généralisant à la fois MDP multicritères et MDP possibilistes dans le cas d'un horizon temporel fini. Ce cadre permet non seulement d'unifier ces approches, mais aussi de mettre en valeur des conditions algébriques garantissant la validité d'un algorithme de programmation dynamique. Ces conditions peuvent être alors utilisées pour tester si de nouvelles représentations de préférences et/ou de nouvelles représentations de l'incertain peuvent être exploitées ensemble en planification dans l'incertain.

EXEMPLE.– Si nous revenons à l'exemple récurrent de l'entretien d'une voiture (voir page 18), les méthodes décrites dans le présent chapitre permettent, entre autre, de traiter les deux problèmes suivants :

– il paraît plus réaliste d'estimer les conséquences d'une action en disant "si on ne répare pas cette fuite d'huile, il est quasiment sûr que le moteur lâchera" que de prétendre savoir que "la probabilité que votre moteur lâche est de $87,72\%$". La théorie des possibilités permet d'utiliser le cadre des MDP avec un savoir exprimé de la première manière.

– plutôt que de ne s'intéresser qu'au coût d'une action, il est possible de combiner plusieurs critères. Par exemple, on pourrait chercher à optimiser à la fois le coût, le temps d'immobilisation de la voiture, les répercussion écologiques et l'encombrement du garage. C'est exactement ce que permettent de modéliser les MDP multicritères.

5.2. Les approches multicritères

Dans cette section, nous allons présenter des modèles décisionnels permettant de relâcher une des hypothèses des MDP classiques qu'est la fonction de récompense réelle scalaire additive. En effet, plusieurs problèmes peuvent être formalisés par des MDP dont la fonction de récompense est multidimensionnelle, représentant différents

critères à optimiser. Devant de tels problèmes, le formalisme des processus décisionnels de Markov multicritères (2V-MDP [1]) a pour but de proposer des solutions. La section 5.2.1 présentera des notions de décision multicritère, puis la section 5.2.2 détaillera le formalisme des 2V-MDP ainsi que les algorithmes de résolution, un exemple illustratif sera explicité pour montrer le fonctionnement du modèle.

5.2.1. *Décision multicritère*

Le but de la décision multicritère [VIN 89, KEE 76] est de choisir une solution préférée dans un ensemble de choix prenant en compte plusieurs aspects (critères), comme par exemple le prix, la qualité, l'apparence etc... Ces critères peuvent être contradictoires (on peut difficilement maximiser la qualité et minimiser le prix en même temps). Le fait d'avoir des critères multiples au sein du processus décisionnel implique que, d'un certain point de vue, un choix, constitué d'un ensemble de critères, peut être arbitrairement préféré à un autre, sans faire pour autant un *« mauvais »* choix. Ceci implique que l'on ne puisse plus définir aisément l'opérateur max utilisé traditionnellement pour définir le choix optimal.

Définition 8 *Un point* $x = (c_1, c_2, \ldots, c_i, \ldots, c_n)$ *de* $\mathbb{R}^n$ *domine un autre point* $x' = (c'_1, c'_2, \ldots, c'_i, \ldots, c'_n)$ *de* $\mathbb{R}^n$ *si :*

$$\wedge \begin{cases} \forall i, c_i \geq c'_i \\ \exists i, c_i > c'_i \end{cases}$$

Sur la figure 5.2.1, la zone grisée correspond aux points dominants le point D_0. La solution à un problème de décision multicritère ne doit pas être dominée par une autre. Elle doit en effet faire partie de l'ensemble Pareto-optimal.

Définition 9 *L'ensemble Pareto-optimal est formé des éléments non dominés.*

Nous introduisons deux points de références, le point Idéal et le point AntiIdéal [BEL 04].

Définition 10 *Le point* $Ideal = (c_1, \ldots, c_i, \ldots, c_n)$ *est défini comme étant le point maximisant tout les critères simultanément. C'est un point de référence, il n'appartient pas nécessairement à un choix possible. Soit pour un ensemble de choix* $E \in \mathbb{R}^n$*,*

$$Ideal_i = \max_j v_j(c_i^{'}), v_j \in E$$

1. Vector-Valued Markov Decision Process

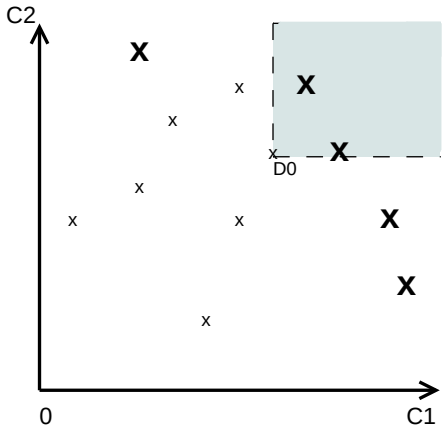


Figure 5.1. *Dominance avec deux attributs*

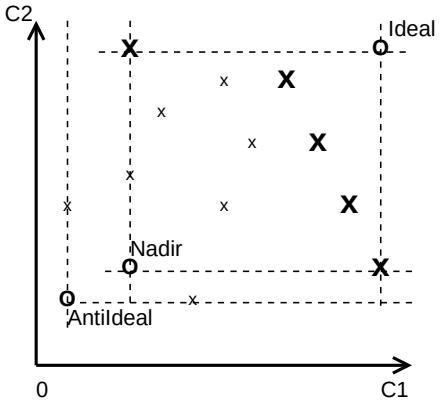


Figure 5.2. *Idéal, AntiIdéal et Nadir*

Le point AntiIdéal est son opposé, c'est-à-dire qu'il minimise tous les critères. Il sert de référence comme étant le pire choix.

$$AntiIdeal_i = \min_j v_j(c_i^{'}), v_j \in E$$

Il est possible de raffiner cet encadrement en définissant le point de $Nadir$. Il définit le point minimisant tous les critères seulement pour les choix appartenant à l'ensemble Pareto-optimal. Avec ces points, nous obtenons un encadrement partant du plus mauvais choix jusqu'au meilleur. Pour plus de détails, le lecteur est invité à consulter [GRA 02a]. Dans la partie suivante, nous allons construire un processus de décision complet.

5.2.2. *Processus décisionnel de Markov multicritères*

Les 2V-MDP [MOU 04] sont donc une extension des MDP où la prise de décision dépend de plusieurs critères. Ces critères sont introduits dans le processus de décision à travers la fonction de récompense, ce qui implique des répercussions sur le reste du formalisme.

Soit $Z = \{z_1, z_2, \ldots, z_n\}$ un vecteur de critères z_i, où chaque z_i représente un des critères du résultat. Une action a_j prise dans un ensemble d'actions $\mathcal{A} =$

$\{a_1, a_2, \ldots, a_m\}$ agit sur un certain nombre de critères (quelques-uns ou tous) transformant le vecteur Z en $Z' = \{z'_1, z'_2, \ldots, z'_n\}$. Il convient donc de modifier la définition du MDP afin d'insérer la prise en compte de ces critères. Ainsi, un MDP multicritère se définit par :

– un ensemble d'états S,

– un ensemble d'actions A,

– une fonction de transition $p(s, a, s'), s, s' \in S, a \in \mathcal{A}$,

– une fonction de récompense $\overrightarrow{r(s)} = \{r_1(s), r_2(s), \ldots, r_i(s), \ldots, r_n(s)\}$ où chaque $r_i(s)$ représente la récompense obtenue dans l'état s pour le critère i (de la même manière que dans les MDP monocritère).

Reprenons maintenant l'équation de Bellman qui permet de dériver, par le calcul de la récompense espérée de chaque état $V(s)$, une politique optimale. Dans sa formulation monocritère, elle s'écrit sous la forme :

$$V(s) = R(s) + \max_a \sum_{s'} p(s, a, s')V(s')$$

Dans le cas multicritère, la fonction de récompense retourne un vecteur. Plus formellement, soit :

$$\overrightarrow{V}(s) = \begin{pmatrix} v_1(s) \\ v_2(s) \\ \ldots \\ v_n(s) \end{pmatrix} = \begin{pmatrix} r_1(s) \\ r_2(s) \\ \ldots \\ r_n(s) \end{pmatrix} + \max_a \sum_{s'} p(s, a, s') * \begin{pmatrix} v_1(s') \\ v_2(s') \\ \ldots \\ v_n(s') \end{pmatrix}$$

Où chaque $v_i, i \in 1 \ldots n$ sont les valeurs des différents critères. Il apparaît que, dans le cas général, l'application directe de l'opérateur $\max$ est impossible, une même action ne maximisant pas simultanément tous les critères. Il en découle des problèmes quant à la convergence de tels processus vers un point fixe. De nombreuses approches on été développées afin de résoudre ce problème [**?**, **?**]. Certaines approches cherchent à trouver tous les chemins efficaces [**?**]. Celles-ci souffrent du fait qu'il peut y avoir un nombre exponentiel (en le nombre d'états) de politiques non dominées.

5.2.2.1. *Opérateurs de décision multicritère*

Nous cherchons ici à obtenir une politique satisfaisante et non pas toutes les politiques. L'opérateur $\max$ a donc besoin d'être redéfini dans le cadre multicritère afin d'obtenir une seule politique à l'issue de l'algorithme. Pour cela, nous nous basons sur la norme pondérée de Tchebychev [BEL 04].

Définition 11 *Norme pondérée de Tchebychev :*

$$\forall p, q \in \mathbb{R}^n, s^{\infty}_{\omega,q}(p) = \max_{i \in \{1,\ldots,n\}} \omega_i \|p_i - q_i\|$$

Cette norme nous permet de définir, pour un point p, une distance à un point de référence q. Pour q, nous choisissons le point Idéal (Définition 10). Il reste à définir les poids ω_i de la norme. Nous les utiliserons afin de normaliser tous les critères entre eux. Les ω_i sont donc définis :

$$\omega_i = \frac{\alpha_i}{\|Ideal_i - AntiIdeal_i\|}$$

où le paramètre α_i permet de réintroduire au besoin des priorités entre les critères [BOU 07]. L'intérêt d'utiliser cette norme (avec la normalisation) est de pouvoir exprimer, pour chaque action, le regret qu'a un agent d'avoir choisi une action par rapport à l'action idéale. A partir de cette norme, nous pouvons construire un opérateur de décision qui remplacera le $\max$ de l'équation de Bellman dans les 2V-MDP. Néanmoins, il apparaît que la norme pondérée de Tchebychev ne conserve pas la propriété d'optimalité de Bellman [GAL 06]. Néanmoins, nous avons pu, au cours de nombreuses expérimentations, montrer le bon comportement des agents suivant ce critère d'optimalité. C'est pourquoi nous avons construit notre opérateur $\max$ sur ce modèle.

Un opérateur décisionnel : le LexDiff

Soit v_i la valeur de la politique courante pour le critère i et v_i^* la valeur de la politique optimale (toujours pour le critère i).

Nous définissons un nouveau vecteur, appelé vecteur d'utilité et noté $\overrightarrow{V^u}$, construit à partir de la norme pondérée de Tchebychev, représentant pour chaque critère la distance normalisée (par les ω_i) au point Idéal $q = (v_1^*, v_2^*, \ldots, v_n^*)$ (pondérée si nécessaire par les coefficients α_i présents dans les ω_i). Ainsi, pour un état $s \in S$:

$$\overrightarrow{V^u}(s) = \begin{cases} v_0^u(s) = \omega_0 * \|v_0(s) - v_0^*(s)\| \\ v_1^u(s) = \omega_1 * \|v_1(s) - v_1^*(s)\| \\ \vdots \\ v_n^u(s) = \omega_n * \|v_n(s) - v_n^*(s)\| \end{cases}$$

Cet opérateur nous garantit une solution Pareto-optimale, tout en préservant une société égalitaire. C'est à dire que la solution choisie ne laissera aucun critère trop se dégrader, même si pour cela, elle perd un peu en utilité globale. Nous utilisons un algorithme de programmation dynamique afin de calculer la politique optimale (au

Algorithme 5.1 : Résolution des 2V-MDP

pour tous les *critères* i **faire**
  Calculer $V_i^*(s)$
$V' \leftarrow 0$
pour $t = 0 \dots T-1$ **faire**
  $V \leftarrow V'$
  pour tous les $s \in S$ **faire**
    pour tous les $a \in A$ **faire**
      $\overrightarrow{V}^{tmp}(a) \leftarrow \overrightarrow{R}(s) + \sum_{s' \in S} P(s,a,s')\overrightarrow{V}(s')$
    pour tous les *critères* i **faire**
      $Ideal_i \leftarrow \max_a V_i^{tmp}(a)$
      $Anti-Ideal_i \leftarrow \min_a V_i^{tmp}(a)$
      $\omega_i \leftarrow \frac{\alpha_i}{Ideal_i - AntiIdeal_i}$
    $V_i^u(a) \leftarrow \omega_i * \|V_{tmp}(a) - V_i^*(s)\|$
  pour tous les $a \in A$ **faire**
    Calculer $V^u(a)$
  $ActionOpt \leftarrow \text{leximin}_a \overrightarrow{V^u(a)}$
  $\overrightarrow{V'(s)} \leftarrow \overrightarrow{V^{tmp}(ActionOpt)}$
retourner V'

sens de l'opérateur). On peut trouver un algorithme similaire dans [**?**]. L'algorithme 5.1 permet de calculer une politique à partir du critère LexDiff.

Pour déterminer le vecteur $\overrightarrow{V^u}(s)$, il est nécessaire de calculer les points $Ideal$ et $AntiIdeal$. Pour le point $Ideal$, il faut effectuer n optimisations monocritère. Chacune de ces valeurs optimales est calculée en utilisant l'algorithme value iteration par exemple. Pour n critères, nous devons donc déterminer n fonctions de valeur optimales. Cela n'augmente pas nécessairement de beaucoup le temps de calcul global. En effet, comme toutes ces optimisations sont indépendantes, il est facile de les paralléliser. Il est aussi possible d'accélérer ce calcul en optimisant un critère tout en gardant les valeurs des autres critères qui, par la suite, serviront à initialiser les calculs des critères suivants. L'utilisation du point $AntiIdeal$ peut conduire à une mauvaise normalisation (en considérant une valeur minimale plus basse que celle des choix appartenant à l'ensemble des solutions Pareto-optimales). Le point $Nadir$ est plus juste, mais est difficile à calculer. C'est pourquoi, nous utiliserons une heuristique classique afin de se rapprocher tout de même du $Nadir$: nous utiliserons la plus petite valeur rencontrée lors des différentes optimisations [EHR 03].

EXEMPLE.– Application numérique : La figure 5.3 présente un MDP avec trois actions

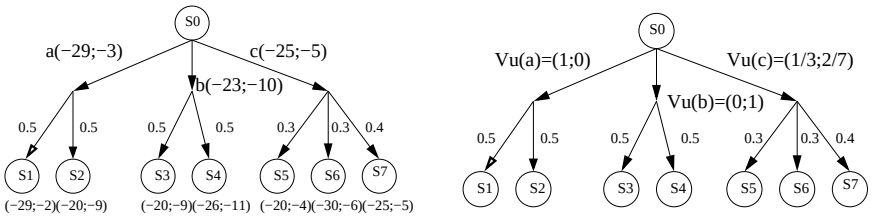


Figure 5.3. *2V-*MDP *: application de l'opérateur LexDiff*

$\{a, b, c\}$. Les probabilités de transition sont indiquées le long des arcs et les récompenses sous les feuilles. Ce MDP comporte ainsi deux critères. Les différentes étapes de la prise de décision sont énumérées ci-dessous :

– calcul des points $Ideal$ et $AntiIdeal$

$$\begin{array}{rcrcl} & & Ideal & = & (-23; -3) \\ Nadir & = & AntiIdeal & = & (-30; -11) \end{array}$$

– calcul des ω_i :

$$\begin{array}{rcl} \omega_1 & = & \frac{1}{\|-23-(-29)\|} = \frac{1}{6} \\ \omega_2 & = & \frac{1}{\|-3-(-10)\|} = \frac{1}{7} \end{array}$$

– calcul des V^u

$$\begin{array}{rcl} \overrightarrow{V^u(a)} & = & (1; 0) \\ \overrightarrow{V^u(b)} & = & (0; 1) \\ \overrightarrow{V^u(c)} & = & (\frac{1}{3}; \frac{2}{7}) \end{array}$$

– tri par ordre lexicographique

$$\begin{array}{rcl} \overrightarrow{V^u(a)} & = & (1; 0) \\ \overrightarrow{V^u(b)} & = & (1; 0) \\ \overrightarrow{V^u(c)} & = & (\frac{2}{7}; \frac{1}{3}) \end{array}$$

– sélection par leximin

$$\textit{Lexdiff}(V) = c$$

5.3. Prise en compte de la robustesse dans la résolution des MDP

Il est courant que le modèle d'un MDP (fonction de transition et fonction de récompense) soit connu de manière incertaine. Les raisons pour cela sont liées à la façon d'obtenir le modèle : via un expert humain ou en faisant des statistiques. Si l'on veut tenir compte de cette incertitude, une première difficulté est de décider comment la modéliser. Mais cette difficulté est directement liée au problème que l'on se pose, lequel peut être :

– Quelles parties du modèle est-il le plus utile d'améliorer pour prendre de meilleures décisions ?

– Comment planifier ses actions en tenant compte de cette incertitude ?

Le premier problème est celui rencontré quand on cherche la meilleure approximation d'un modèle, en discrétisant les états ou les actions, ou en utilisant des fonctions d'approximation. **[cf ? ? ?]** Nous nous intéressons ici au deuxième problème, en reformulant l'objectif de la planification comme celui de trouver une politique optimale face au pire modèle possible. On parle alors de *planification robuste*. On se retrouve ici face à un jeu à deux joueurs (le « planificateur » contre le « modeleur ») et à somme nulle, lequel peut être résumé par le problème d'optimisation suivant :

$$\arg\max_{\pi\in\Pi}\min_{m\in M} V(\pi, m),$$

où Π est l'ensemble des politiques, M l'ensemble des modèles possibles et $V(\pi, m)$ la valeur d'une politique π pour un modèle m (différents critères sont possibles).

La planification robuste nous amène à faire deux remarques à propos de la fonction de récompense :

– Alors qu'une récompense peut être liée à une transition $(s, a) \to s' : r(s, a, s')$, il est usuel de ne faire dépendre la récompense que de s et $a : r(s, a) = E_{s'}[r(s, a, s')]$. Ne connaissant pas ici le vrai modèle, on ne peut calculer cette espérance. Il est donc préférable de garder la formulation $r(s, a, s')$.

– $r(s, a, s')$ est une grandeur incertaine. Mais, parmi ses valeurs possibles, la pire est toujours la plus petite. On fait donc par la suite l'hypothèse que $r(s, a, s')$ prend cette pire valeur.

Notons d'abord que la planification robuste s'intéresse aux modèles possibles, pas à la distribution de probabilité sur les modèles. On n'a donc besoin que de spécifier l'ensemble des modèles possibles. Un choix simple et pratique est de modéliser l'incertitude sur une grandeur (ici $p(s'|s, a)$) par un intervalle :

$$p(s'|s, a) \in [P^{\min}(s'|s, a), P^{\max}(s'|s, a)].$$

La figure 5.4 illustre cette modélisation de l'incertitude sur la transition depuis une paire (état, action) (s, a). Ici, un triangle est un simplexe représentant toutes les distributions de probabilité possibles pour une transition à trois états atteignables ($P(s'_i) = 1$ au sommet s'_i). Le trapèze sur le triangle de gauche correspond à la contrainte fournie par l'intervalle de probabilités pour s'_1. Sur le triangle de droite, les modèles possibles sont à l'intersection des trois contraintes.

On notera que l'ensemble des modèles possibles pour une paire (état, action) forme un polygone convexe (dans le cas de la modélisation à l'aide d'intervalles). Cette convexité est une propriété facilitant souvent la recherche d'un optimum : si la fonction à optimiser est elle aussi convexe, il n'y a qu'un optimum global.

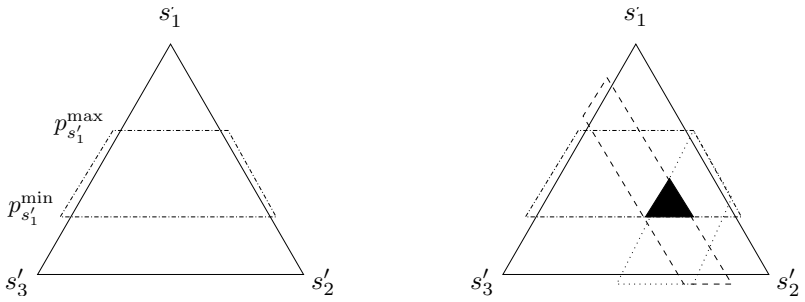


Figure 5.4. *Modélisation de l'incertitude sur la transition depuis une paire (état, action) (s, a) à l'aide d'intervalles.*

La planification robuste telle qu'elle a été décrite permet d'utiliser des algorithmes optimisant les décisions localement (i.e. au niveau d'un état) à la condition que la propriété suivante soit vérifiée :

Propriété 1 *La distribution de probabilité $p(\cdot|s, a)$ est indépendante d'une paire (état, action) à l'autre.*

Faire cette hypothèse accroît le nombre de modèles possibles et permet de se ramener à un jeu alterné, où planificateur et modeleur jouent à tour de rôle. On peut ainsi reformuler l'algorithme d'itération sur les valeurs sous une forme robuste en alternant une étape de minimisation et une étape de maximisation, comme le montre l'algorithme 5.2 avec critère γ-pondéré.

Algorithme 5.2 : Itération sur les valeurs robuste

initialiser $V_0 \in \mathcal{V}$
$n \leftarrow 0$
répéter
  pour $s \in S$ **faire**
    pour $a \in A$ **faire**
      $Q_{n+1}(s, a) \leftarrow$
      $\min_{m^a_s \in \mathcal{M}^a_s} \sum_{s' \in S} p_{m^a_s}(s'|s, a) \left[r(s, a, s') + \gamma V_n(s')\right]$
    $V_{n+1}(s) \leftarrow \max_{a \in A} Q_{n+1}(s, a)$
  $n \leftarrow n + 1$
jusqu'à $||V_{n+1} - V_n|| < \epsilon$
pour $s \in S$ **faire**
  $\pi(s) \in \arg\max_{a \in A} Q_n(s, a)$
retourner V_n,π

On notera qu'il existe toujours une politique optimale déterministe pour le planificateur. De plus, si l'ensemble des modèles possibles est convexe, il existe un pire modèle à l'une des extrémités de cet ensemble (dans le cas de l'utilisation d'intervalles, il s'agit d'un sommet du polygone). Cela permet de définir des procédures simples pour trouver le pire modèle local.

Le lecteur intéressé par la planification robuste trouvera plus de détails dans la litérature suivante [**?**, BAG 01, NIL 04, NIL 05, **?**]. La principale difficulté est liée à la propriété 1. Il existe de nombreux cas dans lesquels celle-ci n'est pas vérifiée, comme en planification avec tâches concurrentes (voir chapitre 15). On ne peut alors plus utiliser la programmation dynamique. Il faut se ramener à des approches par recherche directe de politiques [BUF 05].

5.4. Processus Décisionnels de Markov Possibilistes

Un des apports de l'intelligence artificielle à la théorie de la Décision en général a été la proposition et l'étude de critères de décisions alternatifs au critère traditionnel de l'utilité espérée. Parmi ces critères de décision alternatifs, des critères « qualitatifs », plus adaptés aux problématiques de l'intelligence artificielle (élicitation de connaissances / préférences, communication Homme / Machine), ont été utilisés dans le cadre de la décision séquentielle dans l'incertain. En particulier, une contrepartie qualitative des MDP / POMDP [FAR 98, SAB 01a] a été récemment proposée. Nous allons décrire ce modèle dans la suite de ce chapitre.

5.4.1. *Contreparties possibilistes de l'utilité espérée*

Une distribution de possibilité décrit la connaissance que l'on a de la valeur prise par un ou plusieurs attributs mal connus décrivant l'état d'un système. Par exemple, l'âge d'un homme, la taille d'un immeuble... Dans notre cas, une distribution de possibilité sera utilisée pour modéliser la connaissance imparfaite que l'on a du monde dans un problème de décision dans l'incertain, distinguant les états « plausibles » ou normaux, des états peu vraisemblables ou surprenants.

Plus formellement, une distribution de possibilité π sur un ensemble d'états S est une application de S dans $(L, <)$, une échelle ordonnée finie ou bornée. Cette échelle est supposée équipée d'une fonction de renversement n, bijection de L dans L telle que si $\alpha > \beta \in L$, alors $n(\beta) > n(\alpha)$. 1_L et 0_L représentent respectivement le plus grand et plus petit élément de L et $n(0_L) = 1_L$ et $n(1_L) = 0_L$. Si $L = [0, 1]$, on choisit en général $n = 1 - \cdot$. Dans le cas où L est finie (ce que nous supposerons par la suite, S étant fini), n est la fonction de renversement de L.

La fonction $\pi : S \rightarrow L$ modélise cette connaissance, avec les conventions suivantes :

– $\pi(s) = 0_L$ signifie que s est considéré comme impossible ;

– $\pi(s) = 1_L$ signifie que s est un état « normal » ou totalement possible ;

– $\pi(s) > \pi(s')$ signifie que s est plus vraisemblable que s'.

Notons qu'il se peut très bien que plusieurs états aient une possibilité de 1_L : cela signifie que tous ces états sont également plausibles et sont plus plausibles que tous les autres. Le cas extrême d'ignorance est celui où tous les états partagent une possibilité de 1_L : tous les états sont possibles et rien ne permet de les départager. Au contraire, si un seul état a une possibilité de 1_L et tous les autres sont impossibles (possibilité 0_L), alors on est dans un état de connaissance parfaite. Cette description de la connaissance en termes de distribution de possibilité est assez flexible dans la mesure où tous les degrés de l'échelle L compris entre 0_L et 1_L sont utilisables pour modéliser le degré de possibilité $\pi(s)$ des différents états. En général et cela sera le cas dans ce chapitre, la seule contrainte sur la distribution π est qu'il existe un état s de possibilité 1_L (normalisation) : quelle que soit notre connaissance sur l'état du monde, il existe au moins un état accepté comme « normal ».

Une distribution de possibilité peut être utilisée pour modéliser des *connaissances* incomplètes sur l'état réel du monde. On peut toutefois donner une interprétation différente de cette fonction en termes de *préférence* sur l'état du monde : Dans ce cas, $\pi(s)$ représente le degré auquel s est une situation souhaitable pour un agent (on trouvera dans [DUB 96] une discussion détaillée sur l'interprétation d'une distribution de possibilité en termes de préférence). Nous allons maintenant examiner le cas où deux distributions de possibilité sont utilisées conjointement pour modéliser connaissances et préférences dans des problèmes de décision qualitative dans l'incertain.

Les auteurs de [DUB 95] ont proposé une contrepartie ordinale de la théorie de l'utilité espérée, basée sur la théorie des possibilités. Dans le cadre de la décision non séquentielle, S et X sont respectivement les ensembles (finis) d'états possibles du monde et de conséquences possibles des actions. En supposant que les informations sur les connaissances et les préférences du décideur sont qualitatives, il est raisonnable de représenter à la fois la connaissance incomplète de l'état du monde par une distribution de possibilité π sur S et les préférences graduelles sur les conséquences par une autre distribution de possibilité notée μ sur X, ces deux distributions prenant leurs valeurs sur une échelle commune L finie, totalement ordonnée et de plus petit et plus grand éléments respectivement 0_L et 1_L. L'existence de cette échelle commune se justifie naturellement dans le cadre d'une axiomatisation des critères de décision possibilistes *à la Savage*, telle que proposée dans [DUB 98].

L'incertitude de l'agent sur l'effet d'une action a effectuée dans l'état du monde s est représentée par une distribution de possibilité $\pi(\cdot|s,a) : X \rightarrow L$. La distribution $\pi(x|s,a)$ évalue dans quelle mesure x est une conséquence plausible de l'action a appliquée en s. $\pi(x|s,a) = 1_L$ signifie que x est une conséquence tout à fait plausible, alors que $\pi(x|s,a) = 0_L$ signifie que x est impossible.

De la même manière, les conséquences sont également ordonnées en termes de niveau de satisfaction par une fonction d'utilité qualitative $\mu : S \times A \times X \rightarrow L$. $\mu(s,a,x) = 1_L$ signifie que x est une conséquence tout à fait satisfaisante de a en x,

alors que $\mu(s, a, x) = 0_L$ signifie que x n'est absolument pas satisfaisante. Notons que nous supposerons toujours que π est normalisée, alors que μ peut très bien ne pas l'être (rien ne garantit pour un problème de décision fixé qu'une conséquence totalement satisfaisante puisse être atteinte).

[DUB 95, DUB 98] ont proposé et axiomatisé les deux critères de décision suivants :

$$u^*(a, s) = \max_{x \in X} \min\{\pi(x|s, a), \mu(s, a, x)\} \tag{5.1}$$

$$u_*(a, s) = \min_{x \in X} \max\{n(\pi(x|s, a)), \mu(s, a, x)\} \tag{5.2}$$

où n est la fonction de renversement de L.

u^* peut être vu comme une extension du critère $maximax$ qui évalue les couples (état, action) suivant l'utilité de leur meilleure conséquence possible, alors que u_* est une extension du critère $maximin$ qui les évalue suivant la pire conséquence possible. Utiliser u^* correspond à une attitude optimiste (on se focalise sur les meilleures conséquences possibles d'une action, en ignorant les pires), alors que u_* correspond à une attitude prudente (on se focalise sur les pires conséquences possibles d'une action, en ignorant les meilleures).

Les utilités qualitatives possibilistes, bien que fort différentes de l'utilité espérée, ne sont pas totalement distinctes de celles-ci. En fait, il est possible de démontrer que des relations de préférence basées sur les utilités possibilistes optimistes et pessimistes peuvent toujours être raffinées par une relation de préférence basée sur une utilité espérée, elle même exprimable de manière qualitative en fonction des distributions de possibilités et d'utilité qualitative uniquement [FAR 03, FAR 05].

EXEMPLE.– Considérons l'exemple, proposé par ([SAV 54], pages 13 à 15) pour illustrer le critère de l'utilité espérée : Le problème est de cuisiner une omelette... Nous avons déjà cassé cinq œufs dans un bol et nous avons le sixième, dont l'état de fraîcheur semble douteux, en main. Trois actions sont disponibles : Casser l'œuf dans l'omelette (CO), le casser à part dans une tasse (CT) ou le jeter directement (J). Considérons que l'échelle finie $L = T = \{0, a, b, c, d, 1\}$ où $0 < a < b < c < d < 1$, équipée d'un renversement d'ordre n est suffisante pour exprimer conjointement l'incertitude sur l'état du monde et les préférences sur les conséquences. En particulier, les conséquences sont ordonnées en termes de préférence dans la Table 5.1.

Les degrés entre parenthèses représentent un codage intuitif de l'ordre de préférence entre les conséquences. Deux états du monde sont possibles (frais (F), pourri (P)), de possibilité respectives $\pi(F)$ et $\pi(P)$, avec $\max(\pi(F), \pi(P)) = 1$ (normalisation de la distribution de possibilité représentant l'incertitude).

Actions/État	œuf Frais (F)	œuf Pourri (P)
CO	omelette à 6 œufs (1)	rien à manger (0)
CT	omelette à 6 œufs, tasse à laver (d)	omelette à 5 œufs, tasse à laver (b)
J	omelette à 5 œufs, œuf gâché (a)	omelette à 5 œufs (c)

Tableau 5.1. *États, actions et conséquences dans l'exemple de l'omelette de Savage.*

$$
\begin{aligned}
u_*(CO) &= \min(1, \max(n(\pi(P)), 0)) = n(\pi(P)) \\
u^*(CO) &= \pi(F) \\
u_*(CT) &= \min(d, \max(n(\pi(P)), b)) \\
u^*(CT) &= \max(\min(\pi(F), d), b) \\
u_*(J) &= \min(\max(n(\pi(F)), a), c) \\
u^*(J) &= \max(a, \min(\pi(P), c))
\end{aligned}
$$

Le critère u_* recommande la prudence (CT) dès lors que l'on est ignorant de l'état de l'œuf ($\pi(P)$ et $\pi(F)$ supérieurs ou égaux à $n(a) = d$), ce qui semble plus réaliste que les théories qualitatives suggérant d'oublier les états qui ne sont pas les plus plausibles (comme dans [BOU 94]). Ces dernières, en se focalisant soit sur l'état (F), soit sur l'état (P) ne recommandent en aucun cas l'action (CT), qui semble la plus « intuitive ». Il est à noter que, dans cet exemple, l'attitude optimiste est moins « intuitive », puisqu'elle recommande l'action (CO) en cas d'incertitude.

5.4.2. *Programmation Dynamique Possibiliste*

5.4.2.1. *Horizon fini*

Dans [FAR 98], la théorie de la décision possibiliste a été étendue au cas séquentiel en horizon fini N. Dans ce cadre, l'utilité d'une politique $\delta : S \times H \rightarrow A$ dans un état initial s_0 est définie suivant le cas (pessimiste ou optimiste) par un critère d'utilité qualitative appliqué aux trajectoires possibles (et pas aux états /conséquences possibles) :

$$u_*(\delta, s_0) = \min_{\tau} \max\{n(\pi(\tau|s_0, \delta)), \mu(\tau, \delta)\} \tag{5.3}$$

$$u^*(\delta, s_0) = \max_{\tau} \min\{\pi(\tau|s_0, \delta)), \mu(\tau, \delta)\} \tag{5.4}$$

où, si $\tau = \{s_0, \ldots, s_N\}$,

$$
\begin{aligned}
\mu(\tau) &= *_{i \in 0 \ldots N} \mu(s_i, \delta(s_i)) \\
\pi(\tau|s_0, \delta) &= \min_{i \in 0 \ldots N-1} \pi(s_{i+1}|s_i, d_i(s_i)).
\end{aligned}
$$

$*$ est un opérateur agrégeant les degrés de préférence associés à chaque transition. En pratique, on utilise dans le cas de l'horizon fini soit $*_{i\in 0\ldots N}\mu(s_i,\delta(s_i)) = \min_{i\in 0\ldots N}\mu(s_i)$, soit $*_{i\in 0\ldots N}\mu(s_i,\delta(s_i)) = \mu(s_N)$.

Pour se faire une idée intuitive de ces critères en décision séquentielle, considérons les cas simplifiés suivants :

i) Les possibilités de transition ne prennent que des degrés 0_L ou 1_L, de même que les degrés d'utilité, qui ne sont associés qu'à l'état final : dans ce cas, le critère pessimiste donne une utilité maximale à toutes les stratégies qui ne génèrent que des trajectoires dont l'état final est satisfaisant. Le critère optimiste, lui, sélectionne les stratégies qui génèrent au moins une trajectoire menant à un état satisfaisant.

ii) Même cas, mais l'opérateur d'agrégation $*$ est le min : sont satisfaisantes les stratégies qui soit ne génèrent que des trajectoires dont toutes les transitions sont satisfaisantes (cas pessimiste), soit génèrent au moins une trajectoire dont toutes les transitions sont satisfaisantes (cas optimiste).

iii) Possibilités de transition 0_L ou 1_L, mais les préférences prennent toutes les valeurs possibles de l'échelle L : Le degré de satisfaction d'une stratégie est soit le degré de satisfaction de la *pire* trajectoire possible (cas pessimiste) soit celui de la *meilleure* (cas optimiste) trajectoire possible, où le degré de satisfaction d'une trajectoire est soit le degré de satisfaction de son état final, soit le degré de satisfaction de sa pire transition.

iv) Cas général : on retrouve les critères de décision possibilistes pour la décision à une étape, l'espace d'état étant remplacé par celui des trajectoires, l'espace des conséquence, soit par S_N, soit par l'espace des $N-uplets$ de transitions et l'espace des actions par celui des politiques.

Les contreparties possibilistes (pessimistes et optimistes) des équations de Bellman sont :

– Dans le cas pessimiste (pour $* \equiv \min$) :

$$u_*^t(s) = \max_{a\in A_s}\min_{s'\in S_{t+1}}\min\{\mu(s,a,s'),\max\{n(\pi(s'|s,a)),u_*^{t+1}(s')\}\}$$

$$u_*^N(s) = \mu(s) \qquad (5.5)$$

– Dans le cas optimiste :

$$u^{*t}(s) = \max_{a\in A_s}\max_{s'\in S_{t+1}}\min\{\mu(s,a,s'),\pi(s'|s,a),u^{*t+1}(s')\}$$

$$u^{*N}(s) = \mu(s) \qquad (5.6)$$

Dans [FAR 98] on montre que les politiques calculées récursivement (*backwards*) par applications successives de (5.5) (resp. (5.6)) optimisent le critère u_* (resp. u^*).

Notons qu'à cause de l'idempotence de l'opérateur min, le calcul *backwards* ne calcule qu'un sous-ensemble de l'ensemble des politiques maximisant u_* (resp. u^*).

Néanmoins, ces politiques vérifient la propriété de *cohérence dynamique* : toute sous-politique (de t' à N) d'une politique optimale de t à N (avec $t' \geq t$) est optimale pour le critère choisi (voir [FAR 98]).

5.4.2.2. *Itération sur les valeurs*

Considérons, maintenant dans le cadre qualitatif (possibiliste) les problèmes stationnaires en horizon infini. Pour être plus exact et étant donné qu'il n'existe pas de correspondance dans le cas possibiliste avec le critère γ-pondéré qui permette d'associer un degré de préférence à une trajectoire de longueur infinie, on se limitera dans le cas possibiliste au cas de problèmes à horizon *indéfini* : il existe une fonction d'utilité possibiliste μ sur les états terminaux des trajectoires, une action arbitraire $no-op$ laissant le système dans son état actuel et on cherche une stratégie permettant d'amener le système dans un état final satisfaisant (à coup sûr, ou « possiblement » suivant qu'on est pessimiste ou optimiste), éventuellement en n'effectuant que des transitions satisfaisantes.

Il est possible de définir une version possibiliste de l'algorithme *itération sur les valeurs* pour résoudre ce type de problème. Cet algorithme [SAB 01a] utilise une version possibiliste $\tilde{Q}(s,a)$ de la fonction Q utilisée en apprentissage par renforcement. $\tilde{Q}(s,a)$ évalue l'utilité (pessimiste ou optimiste) de l'action a dans l'état s.

Comme dans le cas stochastique, les stratégies possibilistes optimales peuvent être obtenues en itérant les mises à jour suivantes :

– Cas pessimiste :

$$\tilde{Q}_{t+1}(s,a) = \min_{s' \in S} \min\{\mu(s,a,s'), \max\{n(\pi(s'|s,a)), u_{*t}(s')\}\}, \qquad (5.7)$$

où $u_{*t}(s) = \max_a \tilde{Q}^*_t(s,a)$ et $\tilde{Q}^*_t(s, no-op) = \mu(s)$.

– Cas optimiste :

$$\tilde{Q}^*_{t+1}(s,a) = \max_{s' \in S} \min\{\mu(s,a,s'), \pi(s'|s,a), u^*_t(s')\}, \qquad (5.8)$$

où $u^*_t(s) = \max_a \tilde{Q}^*_t(s,a)$ et $\tilde{Q}^*_t(s, no-op) = \mu(s)$.

Cet algorithme converge en un nombre fini d'itérations (l'algorithme s'arrête dès que $\tilde{Q}_{t+1} = \tilde{Q}_t$). Ceci est facile à prouver, en constatant que la suite de fonctions $(\tilde{Q}^*_t)_t$ est non-décroissante et prend ses valeurs dans l'ensemble L, fini. Au passage, notons que le nombre d'itérations est borné par la taille de l'ensemble des fonctions Q : $|A| \times |S| \times |L|$. Puisqu'une itération de l'algorithme nécessite $|S| \times |A|$ mises à jour, la complexité de la recherche d'une politique optimale est $O(|S|^2 \times |A|^2 \times |L|)$.

Notons également que, contrairement à l'algorithme d'itération sur les valeurs stochastique, l'initialisation de u_* (ou u^*) n'est pas arbitraire (la fonction μ sur S est utilisée pour initialiser μ).

EXEMPLE.– Considérons l'exemple de la figure 5.5, dans lequel un robot doit atteindre le coin inférieur droit de la figure. Une politique le menant dans une des cases voisines

du coin inférieur droit sera partiellement satisfaisante. Les cases noires de la figure représentent des obstacles. La fonction d'utilité μ associée au problème (les degrés de satisfaction ne sont associés qu'aux états finaux du système) également représentée dans la figure 5.5, est définie par : $\mu(s_{33}) = 1$, $\mu(s_{23}) = \mu(s_{32}) = 0,5$ et $\mu(s) = 0$ pour les autres états.

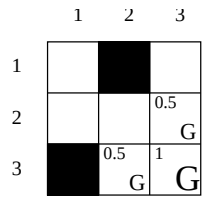


Figure 5.5. *Espace d'état et fonction d'utilité.*

Les actions disponibles sont de déplacer le robot vers le (H)aut, (B)as, (D)roite et (G)auche, ou de (R)ester en place. Si le robot choisi de (R)ester en place, sa position restera identique, avec *certitude*. Par contre, s'il choisit une des autres actions, il se déplacera vers la case désirée avec une possibilité maximale ($\pi = 1$), mais il pourra éventuellement dériver vers une des cases voisines, avec des degrés de possibilités donnés dans la figure 5.6 (pour l'action D, les autres s'obtiennent par symétrie).

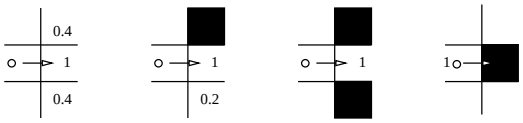


Figure 5.6. *Possibilités de transition pour l'action D.*

Si la case de destination choisie est un obstacle, la position du robot ne change pas (comme si l'action R avait été choisie).

Calculons maintenant la politique (pessimiste) obtenue après une itération de la mise à jour (eq. [5.7]). Pour tous les couples (s, a), nous obtenons :

$\tilde{Q}^1(s,a) = min_{s' \in S} max(1 - \pi(s'|s,a), \mu(s'))$ (π ne dépend pas de l'étape de calcul) et

$u_*^1(s) = max_{a \in \{H,B,G,D,R\}} \tilde{Q}^1(s,a)$.

La figure 5.7 décrit l'utilité pessimiste de chaque action après une itération, ainsi que la politique courante calculée pour tous les états d'utilité non nulle. L'action retournée pour chaque état est unique, excepté pour les états s_{33} et s_{22} pour lesquels B et D peuvent être retournées.

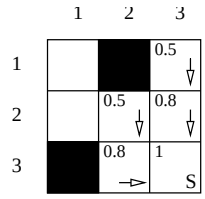


Figure 5.7. *Politique calculée après une itération.*

Il suffit maintenant de réitérer les mises à jour jusqu'à convergence de la fonction de valeur obtenue. Le processus est décrit dans la figure 5.8, où l'on constate que la convergence est obtenue après cinq itérations.

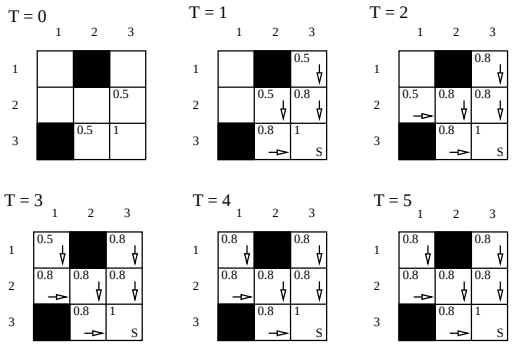


Figure 5.8. *Calcul itératif d'une politique pessimiste optimale.*

Le nombre d'itérations requis pour calculer une politique optimale est de l'ordre du plus long chemin déterministe reliant un état de départ quelconque à l'état but. Ce nombre est toujours inférieur à la taille de l'espace d'états.

Dans cet exemple, la politique optimale optimiste est identique à la politique optimale pessimiste. Seule la fonction de valeur associée (représentée dans la figure 5.9) diffère.

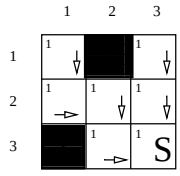


Figure 5.9. *Politique optimale optimiste.*

5.4.2.3. *Itération sur les politiques*

Une version possibiliste de l'algorithme d'itération sur les politiques peut également être définie. Cet algorithme (ici dans le cas où il n'y a pas d'utilités intermédiaires) alterne comme l'algorithme classique des MDP des phases *d'évaluation* et *d'amélioration* de la politique courante :

– Évaluation : Répéter, jusqu'à convergence de u_*^δ :

$$\forall s \in S, u_*^\delta(s) = \min_{s' \in S} \max\{n(\pi(s'|s, \delta(s))), u_*^\delta(s')\} \tag{5.9}$$

– Amélioration :

$$\forall s \in S, \delta(s) \leftarrow argmax_{a \in A} \min_{s' \in S} \max\{n(\pi(s'|s, a)), u_*^\delta(s')\} \tag{5.10}$$

Tout comme pour l'algorithme d'itération sur les valeurs, l'initialisation de la fonction de valeur ne peut pas être arbitraire (la fonction est initialisée par la fonction d'utilité sur les buts). Une version « optimiste » de l'algorithme d'itération sur les politiques s'obtient de la même manière que précédemment.

5.4.3. *Extensions des MDP possibilistes*

Les MDP possibilistes ont été étendus pour faire face aux limitations similaires à celles affrontées par le cadre des MDP classiques. Plus précisément, les trois extensions suivantes ont été proposées :

– *Apprentissage par renforcement*. Certains problèmes de décision qualitative dans l'incertain mêlent à la fois une représentation qualitative des préférences (un préordre sur les préférences) et une représentation incomplète de l'incertain, seulement accessible via la simulation de transitions, ou leur expérimentation. Des méthodes de type *apprentissage par renforcement* ont été proposées pour traiter ces problèmes.

– POMDP *possibilistes*. L'hypothèse d'observabilité complète ou partielle de l'état du monde n'est pas liée au cadre utilisé pour représenter l'incertain. Les MDP possibilistes ont donc été étendus pour prendre en compte l'observabilité partielle inhérente à certains problèmes.

– *diagrammes d'influences possibilistes*. Le cadre possibiliste de représentation de l'incertain est naturellement mieux adapté que le cadre probabiliste au raisonnement sur des connaissances structurées, du fait des opérateurs ($\min$ et $\max$) impliqués dans les outils de raisonnement. Il était donc naturel d'étendre les MDP possibilistes à des représentations structurées des connaissances et des préférences. Récemment, une contrepartie possibiliste des *diagrammes d'influence* a donc été proposée, incluant des outils algorithmiques pour la résolution de ces problèmes.

Dans cette section, nous décrivons brièvement les résultats obtenus sur ces trois points.

5.4.3.1. *Apprentissage par renforcement possibiliste*

[SAB 01b] a proposé des versions possibilistes des algorithmes d'apprentissage par renforcement dits *indirects* : *équivalent certain* et *prioritized sweeping*. Les propriétés mathématiques des opérateurs de type utilité qualitative ne permettent pas de définir des algorithmes d'apprentissage par renforcement *directs* (TD-lambda, Q-learning) aussi, seules des méthodes indirectes ont été développées.

Le problème de l'apprentissage par renforcement possibiliste est de définir un estimateur $\hat{\pi}_t(s'|s,a)$ de $\pi_t(s'|s,a)$ où π et $\hat{\pi}$ appartiennent à une échelle ordinale finie L. Il existe dans la littérature de nombreux opérateurs de transformation entre probabilités et possibilités. Ces opérateurs peuvent être classés en deux catégories :

– La première famille [DAR 94, HEN 99] est basée sur une interprétation des degrés de possibilité en termes de « probabilités infinitésimales ».

– La seconde famille [GIA 99, DUB 93] est basée sur le principe de transformations entre probabilités et possibilités *cohérentes*. Une transformation est cohérente dès lors que $\forall A, B \subseteq S, P(A) \leq P(B) \Rightarrow \Pi(A) \leq \Pi(B)$[2].

La méthode d'apprentissage de politiques possibilistes optimales la plus simple (et la moins efficace) est la méthode de *l'équivalent certain*, qui consiste à apprendre $\hat{\pi}$ et $\hat{\mu}$ par exploration exhaustive de $S \times A$ avant d'appliquer un algorithme d'itération sur les valeurs ou sur les politiques possibiliste. Cette méthode est inefficace car, comme dans le cas stochastique, elle consacre le même effort à tout l'espace d'état, alors que certains états peu plausibles ont peu d'influence sur la valeur globale d'une politique et que certains couples (état, action) peuvent très vite être considérés comme « mauvais ». Elle peut être améliorée en alternant des phases d'apprentissage de modèle et de mise à jour de la fonction de valeur possibiliste, constituant ainsi une forme d'algorithme de type *prioritized sweeping possibiliste*.

L'algorithme *prioritized sweeping possibiliste* (PSP) est similaire à l'algorithme stochastique, à la différence que (comme pour l'algorithme itération sur les valeurs possibiliste) la politique courante est sauvegardée en mémoire en plus de la fonction de valeur courante. A chaque fois qu'une action a est appliquée en un état s et que des changements de $\hat{\pi}$ et $\hat{\mu}$, suffisants pour changer la valeur courante de $\hat{Q}(s,a)$, sont observés, ces changements sont propagés vers les *prédécesseurs* de s. Si la valeur des prédécesseurs est modifiée, ces modifications sont également propagées etc. La propagation est effectuée grâce à une file d'attente de type FIFO contenant les prédécesseurs à modifier. La taille de la file d'attente est bornée, ainsi que le nombre de mises à jour par transition observée.

---

2. Remarquons que les transformations basées sur des probabilités infinitésimales ne sont pas forcément cohérentes, sauf lorsque $\epsilon \rightarrow 0$, auquel cas les distributions de possibilité obtenues ont tendance à être des distributions « tout ou rien ».

Malheureusement, la politique retournée par l'algorithme PSP n'est pas toujours optimale. Ceci est lié au processus d'allocation d'actions : une nouvelle action $\hat{a}^*(s_{loc})$ est associée par l'algorithme à l'état courant s_{loc} à chaque fois que la valeur courante $\hat{u}^*(s_{loc})$ est modifiée par une expérience. Or il se peut au cours de l'apprentissage que la valeur courante $\hat{u}^*(s_{loc})$ devienne égale à la valeur optimale alors que le modèle courant $\hat{\pi}, \hat{\mu}$ n'est pas encore correct, ce qui implique que l'action courante $\hat{a}^*(s_{loc})$ n'est pas forcément optimale. Si, par la suite, la fonction de valeur courante ne change plus alors que le modèle continue à changer, l'action courante ne pourra plus être modifiée. Le moyen utilisé par [SAB 01b] pour résoudre ce problème consiste à utiliser l'algorithme PSP pour calculer une politique sous-optimale, puis à lancer un algorithme d'itération sur les politiques possibiliste à partir de cette politique, en utilisant les estimations $\hat{\pi}$ et $\hat{\mu}$ courantes. Ceci permet de restaurer l'optimalité des politiques, lorsque le nombre d'essais alloués à PSP augmente. En pratique, on constate que la politique calculée par PSP est « presque » optimale et qu'un très petit nombre d'itérations sont ensuite nécessaires pour obtenir une politique optimale. PSP+Itération sur les politiques permet de calculer une politique optimale plus rapidement que l'algorithme d'*équivalent certain* possibiliste.

5.4.3.2. *MDP possibiliste partiellement observable*

La notion de conditionnement a été étudiée dans le cadre de la théorie des possibilités (voir [DUB 94] pour une présentation complète). Le conditionnement par rapport à un événement prend une forme similaire à celle du conditionnement Bayésien :

$$\forall A, B, \Pi(A \cap B) = \min\{\Pi(B|A), \Pi(A)\}. \tag{5.11}$$

Contrairement au cas du conditionnement Bayésien, l'équation 5.11 ne possède pas une solution $\Pi(B|A)$ unique. Aussi, en général, on choisit la solution de l'équation 5.11 *la moins spécifique*[3] :

$$\Pi(B|A) = 1_L \text{ si } \Pi(A \cap B) = \Pi(A) > 0_L \text{ et } \Pi(B|A) = \Pi(A \cap B) \text{ sinon.} \tag{5.12}$$

Une fois ce choix effectué pour le conditionnement de la mesure de possibilité, le conditionnement $\pi(\cdot|o)$ d'une distribution de possibilité par une observation $o \in \Omega$ se définit immédiatement par :

$$\pi(s|o) = 1_L \text{ si } \pi(s, o) = \Pi(o) \text{ et } \pi(s|o) = \pi(s, o) \text{ sinon.} \tag{5.13}$$

Où $\Pi(o) = max_s \pi(s, o)$ et $\pi(\cdot, \cdot)$ est la distribution de possibilité jointe sur $S \times \Omega$.

MDP **possibiliste partiellement observable**

A partir de cette définition du conditionnement possibiliste, il est facile de définir un POMDP possibiliste [SAB 99], de la même manière que dans le cadre stochastique.

3. Si $\Pi(A \cap B) = \Pi(A) < 1_L$ alors, $\forall \alpha \geq \Pi(A \cap B)$, $\Pi(B|A) = \alpha$ satisfait l'équation 5.11 (lorsque $* = min$).

Dans le cadre possibiliste, un POMDP possibiliste (Π-POMDP) peut être transformé en un PDM possibiliste, tout comme dans le cas stochastique. Toutefois, dans ce cas, l'espace d'état *reste fini*, ce qui permet d'appliquer les algorithmes itératifs décrits précédemment : un *état de croyance possibiliste* β est une distribution de possibilité sur l'espace d'états S. Contrairement au cas stochastique, l'ensemble des états de croyance possibilistes est fini dès lors que l'échelle L utilisée pour préciser les degrés de possibilité est finie. Le cardinal de B, l'ensemble des états de croyance possibilistes est majoré par $|L|^{|S|}$.

Supposons maintenant comme dans le cas probabiliste que les possibilités de transition $\pi(s'|s,a)$ sont données, de même que les possibilités des observations, $\pi(o|s,a)$. Alors on peut définir $\beta_a(s')$, la possibilité d'atteindre s' en partant d'une connaissance sur l'état initial définie par β et en appliquant l'action a :

$$\beta_a(s') = \max_{s\in S} \min\{\pi(s'|s,a), \beta(s)\}. \tag{5.14}$$

On calcule ensuite la possibilité d'observer $o \in \Omega$ après avoir appliqué a en β :

$$\beta_a(o) = \max_{s\in S} \min\{\pi(o|s,a), \beta_a(s)\}. \tag{5.15}$$

Maintenant, β_a^o est l'état de croyance possibiliste, révisé après avoir appliqué a en β et observé o :

$$\begin{aligned} \beta_a^o(s) &= 0_L \text{ si } \pi(o|s,a) = 0_L, \\ \beta_a^o(s) &= 1_L \text{ si } \pi(o|s,a) = \beta_a(o) > 0_L, \\ \beta_a^o(s) &= \beta_a(s) \text{ dans les autres cas.} \end{aligned} \tag{5.16}$$

Tous les éléments du nouveau MDP possibiliste sur l'espace des états de croyance sont définis dans les équations 5.14, 5.15 et 5.16. Intuitivement, l'évolution du système se définit par : si le système est dans l'état β, alors appliquer l'action a peut mener dans l'un des $|\Omega|$ états successeurs possibles β_a^o, la possibilité de rejoindre l'état β_a^o étant $\beta_a(o) = \pi(o|\beta,a)$.

A partir de là, les équations de Bellman possibilistes peuvent être étendues au cas partiellement observable (ici dans le cas pessimiste)[4] :

$$u_*^t(\beta) = \max_{a\in A_s} \min\{\mu(\beta), \min_{o\in O} \max\{n(\beta_a(o)), u_*^{t+1}(\beta_a^o)\}\},$$

où $\mu(\beta) = \min_{s\in S} \max\{n(\beta(s)), \mu(s)\}$ et $u_*^0(\beta)$ est initialisé à $\mu(\beta)$.

4. Par souci de simplification des notations, on se limite ici à une fonction d'utilité μ sur les états et non sur les transitions. Évidemment, l'équation **??** peut être étendue pour prendre en compte des préférences sur les transitions.

5.4.3.3. *Diagrammes d'influence possibilistes (DIP)*

Le cadre des *Diagrammes d'Influence Possibilistes* (DIP), contrepartie possibiliste des *diagrammes d'influence*, a été défini récemment [GAR 06]. La partie graphique d'un DIP est exactement la même que celle d'un diagramme d'influence usuel mais la sémantique diffère. Les vraisemblances des transitions sont exprimées par des distributions de possibilité et les récompenses sont considérées ici comme des degrés de satisfaction attachés à des buts partiels. L'utilité espérée est alors remplacée par l'un des deux critères d'utilité qualitative possibiliste présentés précédemment.

Les algorithmes de programmation dynamique possibiliste (*recherche arrière*, puisque l'horizon est fini) sont applicables pour résoudre un problème exprimé sous la forme d'un DIP. Cependant, ils nécessitent des ressources en temps exponentielles pour calculer l'utilité d'une stratégie possibiliste optimale[5]. [GAR 07, GAR 08] ont montré que le calcul de l'utilité d'une stratégie optimale pour un DIP était NP-complet dans le cas optimiste[6] et PSPACE-complet dans le cas pessimiste. Ils ont proposé deux classes d'algorithmes, respectivement basés sur l'exploration d'un arbre de décision ou sur l'élimination de variables, permettant de résoudre des problèmes exprimés sous la forme de DIP. Indépendamment, un modèle algébrique plus général pour la décision (structurée) dans l'incertain a été proposé [PRA 06], présentant également des algorithmes de la même famille, pour une classe de problèmes plus vaste.

5.5. MDP algébriques

Nous présentons maintenant un cadre étendant à la fois celui des MDP multicritères et celui des MDP possibilistes. Dans le but d'étudier ces problèmes de planification utilisant une représentation de l'incertain non probabiliste et/ou une représentation non classique des préférences sur les actions (récompenses non nécessairement réelles scalaires additives), nous introduisons le cadre général des MDP algébriques proposé par [PER 05, WEN 06b].

Avant de présenter ce formalisme, nous faisons un bref rappel des outils utilisés : semi-anneaux, mesure de plausibilité et utilité espérée généralisée. Nous présentons ensuite formellement les MDP algébriques. Sous certaines conditions que nous détaillons, il est possible d'utiliser un algorithme d'induction arrière pour déterminer les politiques non dominées. Cette étude préliminaire est restreinte au cas de l'horizon fini (nombre fini d'étapes de décisions).

---

5. Et un espace exponentiel pour la représenter.

6. En fait, c'est le problème de *décision* associé à ce problème d'*optimisation*, qui est NP-complet.

5.5.1. *Rappels*

5.5.1.1. *Semi-anneaux*

Pour la définition d'un MDP algébrique (AMDP), nous introduisons deux échelles de valuation V et P, pour mesurer respectivement les récompenses et l'incertain. Elles sont supposées munies d'une structure de *semi-anneau* (cf. [GON 01] pour un exposé plus complet).

Définition 12 *Un* semi-anneau $(X, \oplus_X, \otimes_X, 0_X, 1_X)$ *est un ensemble* X *muni de deux lois* $\oplus_X$ *et* $\otimes_X$ *qui vérifient les conditions suivantes :*

– $(X, \oplus_X, 0_X)$ *est un monoïde commutatif avec* 0_X *comme élément neutre,* i.e. *:*

$$\begin{aligned} a \oplus_X b &= b \oplus_X a \\ (a \oplus_X b) \oplus_X c &= a \oplus_X (b \oplus_X c) \\ a \oplus_X 0_X &= a \end{aligned}$$

– $(X, \otimes_X, 1_X)$ *est un monoïde avec* 1_X *comme élément neutre et* 0_X *comme élément absorbant,* i.e. *:*

$$\begin{aligned} (a \otimes_X b) \otimes_X c &= a \otimes_X (b \otimes_X c) \\ 1_X \otimes_X a = a \otimes_X 1_X &= a \\ 0_X \otimes_X a = a \otimes_X 0_X &= 0_X \end{aligned}$$

– $\otimes_X$ *est distributif sur* $\oplus_X$ i.e. *:*

$$(a \oplus_X b) \otimes_X c = (a \otimes_X c) \oplus_X (b \otimes_X c) \tag{5.17}$$

$$a \otimes_X (b \oplus_X c) = (a \otimes_X b) \oplus_X (a \otimes_X c) \tag{5.18}$$

L'opération $\oplus_X$ permet de définir un préordre (non nécessairement complet) $\geq_X$ appelé *canonique* comme suit :

$$\forall x, y \in X, x \geq_X y \Leftrightarrow \exists z \in X, x = z \oplus_X y.$$

Le semi-anneau est dit *idempotent* lorsque $\oplus_X$ est idempotent (i.e. $\forall x \in X, x \oplus_X x = x$). Dans ce cas, le préordre canonique $\geq_X$ associé à X est un ordre.

Ainsi, l'échelle de valuation des récompenses V est supposée munie de la structure de semi-anneau idempotent $(V, \oplus_\mathcal{V}, \otimes_\mathcal{V}, 0_\mathcal{V}, 1_\mathcal{V})$. Intuitivement, la loi $\oplus_\mathcal{V}$ est une opération pour la sélection des éléments préférés et la loi $\otimes_\mathcal{V}$ permet la combinaison des éléments. L'échelle de valuation de l'incertain P est supposée munie de la structure de semi-anneau $(P, \oplus, \otimes, 0_P, 1_P)$. L'interprétation des opérateurs $\oplus$ et $\otimes$ est donnée dans la section suivante. Nous supposons de plus pour simplifier que le préordre canonique sur P est un ordre. À titre illustratif, dans les MDP classiques, on a $(V, \oplus_\mathcal{V}, \otimes_\mathcal{V}, 0_\mathcal{V}, 1_\mathcal{V}) = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$ et $(P, \oplus, \otimes, 0_P, 1_P) = ([0, +\infty[, +, \times, 0, 1)$.

5.5.1.2. *Mesures de plausibilité*

Pour modéliser l'incertain lié aux conséquences d'une action, nous faisons appel aux *mesures de plausibilité*[7] proposées par [FRI 95] généralisant la plupart des représentations de l'incertain.

Définition 13 *Soit X un ensemble fini. Une* mesure de plausibilité Pl *sur* 2^X *est une application de* 2^X *dans* P *vérifiant :*

– $Pl(\emptyset) = 0_P$

– $Pl(X) = 1_P$

– $\forall A, B \in X, A \subseteq B \Rightarrow Pl(A) \leq Pl(B)$

On peut interpréter de manière simple ces trois conditions. La dernière condition affirme une certaine cohérence dans les plausibilités : un évènement est toujours plus plausible (au sens large) que tout évènement le composant. Elle implique avec la première condition que l'évènement impossible est le moins plausible des évènements et avec la seconde condition que l'évènement certain est l'évènement le plus plausible.

Une mesure de plausibilité est dite *décomposable* si $Pl(A \cup B) = Pl(A) \oplus Pl(B)$ pour toute paire A, B d'événements disjoints et $Pl(A \cap B) = Pl(A) \otimes Pl(B)$ pour toute paire A, B d'événements indépendants au sens des plausibilités ([HAL 01]).

La restriction d'une mesure de plausibilité décomposable sur les singletons de 2^X est appelée *distribution de plausibilité*. Celle-ci détermine complètement la mesure de plausibilité décomposable. Dans les AMDP, on suppose que l'incertain est représenté par des distributions de plausibilité.

Ainsi, on constate que les deux lois du semi-anneau $(P, \oplus, \otimes, 0_P, 1_P)$ permettent respectivement la combinaison des événements disjoints et la combinaison des événements indépendants. Notons de plus que l'hypothèse que $(P, \oplus, \otimes, 0_P, 1_P)$ est un semi-anneau n'est pas très restrictive car [DAR 92], qui utilise des propriétés similaires pour définir les probabilités symboliques, ont montré que ces propriétés sont vérifiées par de nombreuses représentations de l'incertain, telle la théorie des probabilités, la théorie des possibilités et d'autres systèmes de calcul utilisés en intelligence artificielle.

5.5.1.3. *Utilité espérée généralisée*

Nous présentons maintenant le formalisme des *utilités espérées généralisées* (GEU) proposé par [CHU 03] pour offrir un cadre général pour l'étude de critères décisionnels. Dans ce cadre, on suppose que les utilités sont mesurées sur une échelle V et l'incertain concernant les conséquences d'une action est représenté par une mesure de

7. À ne pas confondre avec les fonctions de plausibilité de Dempster et Shafer [DEM 67, SHA 76]

plausibilité à valeur dans une échelle P. Le lecteur intéressé par une justification axiomatique de ce critère quand les mesures de plausibilité sont supposées décomposables pourra se référer à [WEN 06a].

Comme dans les critères classiques (total, total pondéré, moyenne), GEU combine les plausibilités et les utilités pour définir un critère de décision. Dans ce but, on introduit les opérations $\oplus^g : V \times V \to V$ et $\otimes^g : P \times V \to V$ qui sont les analogues de $+$ et $\times$ sur les réels utilisés par les critères classiques. On suppose que ces deux opérations satisfont trois prérequis :

GEU1 $(x \oplus^g y) \oplus^g z = x \oplus^g (y \oplus^g z)$

GEU2 $x \oplus^g y = y \oplus^g x$

GEU3 $1_P \otimes^g x = x$

Le critère GEU se définit alors par :

$$GEU(Pl) = \bigoplus_{x \in V}^{g} Pl(x) \otimes^g x$$

où Pl est une mesure de plausibilité.

Finalement, on dira que Pl est préféré à Pl' si et seulement si $GEU(Pl) \geq_{\mathcal{V}} GEU(Pl')$.

5.5.2. *Définition d'un* MDP *algébrique*

Un MDP algébrique (AMDP) est alors décrit comme un quintuplet $(\mathcal{S}, \mathcal{A}, p, r, T)$, où p et r sont redéfinis comme suit :

– p: $\mathcal{S} \times \mathcal{A} \to \mathbf{Pl}(\mathcal{S})$ est une fonction de transition, où $\mathbf{Pl}(\mathcal{S})$ est l'ensemble des distributions de plausibilité sur $\mathcal{S}$, valuées dans P,

– r: $\mathcal{S} \times \mathcal{A} \to V$ est une fonction de récompense donnant la récompense immédiate d'une action, valuée dans V.

De manière cohérente avec l'hypothèse de Markov, l'état suivant et la récompense ne dépendent que de l'état courant et de l'action choisie. En particulier, les distributions de plausibilité de type $p(s, a)$ sont indépendantes (de manière plausibiliste) des états et des actions passés.

EXEMPLE.– La plupart des MDP introduits précédemment dans la littérature sont des instances de AMDP. Dans les MDP standards (section 1.2.1, p. 18), l'incertain est probabiliste. Ainsi, la structure algébrique sous-jacente utilisée pour les plausibilités est $(P, \oplus, \otimes, 0_P, 1_P) = ([0, +\infty[, +, \times, 0, 1)$. Le critère d'évaluation de l'application d'une politique dans un état repose sur le modèle de l'utilité espérée, ce qui indique que les opérations $\oplus^g$ et $\otimes^g$ sont respectivement $+$ et $\times$. Quand les récompenses

sont définies sur $(V, \oplus_{\mathcal{V}}, \otimes_{\mathcal{V}}, 0_{\mathcal{V}}, 1_{\mathcal{V}}) = (\overline{\mathbb{R}}, \max, +, -\infty, 0)$ où $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$, nous reconnaissons le critère total (p. 22). Avec $(\overline{\mathbb{R}}, \max, +_\gamma, -\infty, 0)$ (où $x +_\gamma y = x + \gamma y$), nous reconnaissons le critère total pondéré. Avec $(\overline{\mathbb{R}}, \max, +_h, -\infty, 0)$ où $a +_h b = \frac{1}{\gamma}a + b$, nous reconnaissons le critère moyen.

Les MDP possibilistes (présentés en section 5.4, p. 175) introduits par [SAB 98] sont aussi des AMDP dans lesquels l'incertain est valué sur une échelle qualitative L munie de la structure de semi-anneau $(L, \vee, \wedge, O_L, 1_L)$ où $\vee$ et $\wedge$ sont respectivement les opérateurs maximum et minimum sur L. Quand l'utilité optimiste est utilisée (section 5.4.1, p. 175), les récompenses sont valuées sur la même échelle munie de la structure $(L, \vee, *, O_L, e^*)$ où $*$ est la loi de composition sur les récompenses ($\wedge$ par exemple), d'élément neutre e^*. Les opérations $\oplus^g$ et $\otimes^g$ sont respectivement $\vee$ et $\wedge$. En revanche, quand l'utilité pessimiste est utilisée, il faut inverser l'échelle d'évaluation L et minimiser ses valeurs car on a :

$$U^-(\pi) = n(\bigvee_{x \in X} \big(\pi(x) \wedge n(u(x))\big))$$

où n est l'opérateur involutif d'inversion d'ordre sur L.

Les MDP qualitatifs, introduits par [BON 02], sont des AMDP où les mesures de plausibilité sont définies sur le semi-anneau des séries formelles $(\Sigma(\epsilon), +, \times, 0, 1)$ où $\Sigma(\epsilon)$ est défini comme l'ensemble des séries formelles infinies convergentes en ϵ :

$$\Sigma(\epsilon) = \{ \sum_{k=-\infty}^{\infty} a_k \epsilon^k : \forall k, a_k \in \mathbb{R}, \sum_{k=-\infty}^{\infty} |a_k| \epsilon^k < \infty \}$$

Les opérations $+$ et $\times$ sont l'addition et la multiplication sur les séries. Elles sont bien définies car les séries sont convergentes. Les récompenses sont, quant à elles, définies sur le semi-anneau $(\Sigma(\epsilon) \cup \{-\infty\}, \max, +, -\infty, 0)$. Enfin, les opérations pour le calcul des espérances sont simplement $+$ et $\times$.

Les MDP multicritères (présentés en section 5.2.2, p. 168) sont également une instance de AMDP. L'incertain est probabiliste comme dans le cas standard. Nous ne présentons pas ici le semi-anneau des récompenses utilisé. Le lecteur intéressé pourra se référer à [WEN 06b] pour plus de détails.

En dehors de ces instances de AMDP déjà proposées, nous présentons d'autres exemples qui n'ont pas encore été étudiés à notre connaissance en section 5.5.5 pour justifier de l'intérêt de notre approche algébrique.

5.5.3. *Fonctions de valeur d'une politique*

Nous pouvons maintenant procéder de la même manière que dans les MDP classiques et définir une fonction de valeur pour les politiques. À cette fin, nous définissons la valeur d'un historique $\gamma_t = (s_t, a_t, s_{t-1}, \ldots, a_1, s_0)$ par :

$$r(\gamma_t) = \bigotimes_{i=1}^{t}{}_{\mathcal{V}}\, r(s_i, a_i).$$

Pour un état initial s, une politique $\pi = (\delta_t, \ldots, \delta_1)$ induit une distribution de plausibilité $Pl_t^\pi(s, \cdot)$ sur les historiques. Ainsi, la plausibilité que l'application de la politique π dans l'état s génère un historique γ_t est donnée par $Pl_t^\pi(s, \gamma_t)$. La fonction de valeur d'une politique π, calculée grâce à cette distribution de plausibilité en utilisant GEU, s'écrit :

$$\forall s \in \mathcal{S}, v_t^\pi(s) = \bigoplus_{\gamma \in \Gamma_t(s)}^{g} Pl_t^\pi(s, \gamma) \otimes^g r(\gamma)$$

Cette fonction de valeur peut être vue comme un vecteur de V^n où n est le nombre d'états. Les politiques peuvent donc être comparées en utilisant la relation de dominance $\succeq_{V^n}$ entre vecteurs de V^n :

$$x \succeq_{V^n} y \iff (\forall i = 1, \ldots, n, \;\; x_i \geq_{\mathcal{V}} y_i) \tag{5.19}$$

pour tout $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n) \in V^n$. En effet, à un horizon t donné, une politique π est préférée à une politique π' si et seulement si, dans chaque état, la valeur de la politique π est meilleure que celle de π', ce qui s'écrit :

$$\pi \succsim \pi' \Leftrightarrow \forall s \in \mathcal{S}, v_t^\pi(s) \geq_{\mathcal{V}} v_t^{\pi'}(s).$$

De manière plus compacte, en utilisant la relation de dominance $\succeq_{V^n}$, on peut alors écrire :

$$\pi \succsim \pi' \Leftrightarrow v_t^\pi \succeq_{V^n} v_t^{\pi'}.$$

Une politique *non dominée* est une politique pour laquelle il n'existe pas de politique qui lui soit préférée.

La résolution d'un AMDP revient donc à déterminer les politiques non dominées. Ces dernières peuvent être obtenues par le calcul des équations suivantes :

$$\begin{array}{ll} \forall s \in \mathcal{S} & v_0^*(s) = 1_{\mathcal{V}} \\ \forall s \in \mathcal{S}, \forall t = 1 \ldots h & v_t^*(s) = \bigoplus_{\pi \in \Pi_t} v_t^\pi(s) \end{array} \tag{5.20}$$

5.5.4. *Conditions*

Les équations (5.20) sont difficilement applicables directement quand l'horizon est élevé. Pour cette raison, nous introduisons un ensemble de conditions sur les opérateurs algébriques qui garantit que ces équations peuvent être calculées itérativement :

AMDP1 $p \otimes^g (x \oplus_{\mathcal{V}} y) = (p \otimes^g x) \oplus_{\mathcal{V}} (p \otimes^g y)$

AMDP2 $x \oplus^g (y \oplus_{\mathcal{V}} z) = (x \oplus^g y) \oplus_{\mathcal{V}} (x \oplus^g z)$

AMDP3 $p \otimes^g (q \otimes^g x) = (p \otimes q) \otimes^g x$

AMDP4 $\bigoplus_i^g p_i \otimes^g (x \otimes_{\mathcal{V}} y_i) = x \otimes_{\mathcal{V}} (\bigoplus_i^g p_i \otimes^g y_i)$

AMDP5 $p \otimes^g (x \oplus^g y) = (p \otimes^g x) \oplus^g (p \otimes^g y)$

pour tout $p, q, p_i \in P, x, y, z, y_i \in V$.

Les conditions AMDP1 et AMDP2 sont deux propriétés de distributivité impliquant une certaine forme d'additivité de $\geq_{\mathcal{V}}$ par rapport à $\otimes^g$ et $\oplus^g$ (c'est-à-dire, $x \geq_{\mathcal{V}} y \Rightarrow (z * x \geq_{\mathcal{V}} z * y)$ pour $* \in \{\otimes^g, \oplus^g\}$). La condition AMDP3 permet la réduction des loteries. La condition AMDP4 rend possible l'isolation d'un gain certain dans une loterie. Remarquons qu'elle est similaire à un axiome de distributivité introduit par [LUC 03], qui affirme qu'une loterie probabiliste l est équivalente à recevoir de manière conjointe un gain certain x et une autre loterie probabiliste obtenue à partir de l en retranchant x à toutes ses conséquences. Enfin, la condition AMDP5 est une condition de distributivité analogue à celle rencontrée dans l'espérance classique.

Ces conditions étant vérifiées, on peut écrire une version algébrique des équations de Bellman (équations (1.1), p. 28) :

$$\begin{array}{ll} \forall s \in \mathcal{S} & v_0^*(s) = 1_{\mathcal{V}} \\ \forall s \in \mathcal{S}, \forall t = 1 \ldots h & v_t^*(s) = \bigoplus_{a \in \mathcal{A}} r(s,a) \otimes_{\mathcal{V}} \bigoplus_{s' \in \mathcal{S}}^{g} p(s,a,s') \otimes^g v_{t-1}^*(s') \end{array} \tag{5.21}$$

[PER 05] ont démontré la proposition suivante qui indique que les équations (5.20) et (5.21) sont équivalentes.

PROPOSITION 5.1 [PER 05].– *Si les conditions AMDP1 à AMDP5 sont vérifiées, alors les politiques obtenues par les équations de Bellman (5.21) sont des politiques non dominées.*

La proposition 5.1 justifie donc l'emploi d'une version algébrique de l'algorithme d'induction arrière 5.3 et factorise en un résultat différents travaux sur les MDP classiques, sur les MDP multicritères, sur les MDP possibilistes,... De part leur généralité, ces résultats permettent donc d'expliquer un certain nombre de travaux connus développés dans des contextes différents, mais aussi de justifier par avance l'algorithme d'induction arrière dans des contextes qui n'ont pas encore été étudiés. Nous présenterons en section 5.5.5 quelques instances originales et potentiellement utiles de AMDP, qui n'ont pas encore été investiguées à notre connaissance.

5.5.5. *Exemples de* AMDP

Pour montrer la généralité de l'approche algébrique, nous fournissons quelques exemples de AMDP non encore étudiées à notre connaissance : un premier exemple avec une structure de préférence incomplète, un deuxième exemple avec une représentation qualitative de l'incertain, et enfin, un dernier exemple où les récompenses sont des fonctions croissantes sur un semi-anneau de valuation.

Algorithme 5.3 : Version algébrique de l'algorithme d'induction arrière pour les AMDP

$v_0 \leftarrow \mathbf{1}$
$t \leftarrow 0$
répéter
  $t \leftarrow t+1$
  pour $i = 1 \dots n$ **faire**
    pour $j = 1 \dots m$ **faire**
      $q_t(s^i, a^j) \leftarrow r(s^i, a^j) \otimes_{\mathcal{V}} \bigoplus_{k=1\dots n}^{g} p(s^i, a^j, s^k) \otimes^g v_{t-1}(s^k)$
    $v_t(s^i) \leftarrow q_t(s^i, a^1) \oplus_{\mathcal{V}} \dots \oplus_{\mathcal{V}} q_t(s^i, a^m)$
jusqu'à $t = h$

5.5.5.1. AMDP *multicritères probabilistes*

Dans les MDP multicritères probabilistes, on utilise la Pareto-dominance pour discriminer entre les vecteurs de récompenses. L'ordre induit par la Pareto-dominance est partiel et il arrive qu'il ne soit pas assez discriminant et qu'on obtienne un nombre trop important de politiques non dominées. Nous proposons de raffiner l'ordre induit par la Pareto-dominance en introduisant une priorité dans la comparaison des critères.

Soit Q l'ensemble de ces critères ($|Q|$ critères au total) et $\succeq_Q$ une relation d'ordre (qui peut être partielle) sur Q (reflétant l'importance des critères). Suivant [GRO 91] et [JUN 02], nous utilisons une relation d'ordre strict $\succ_G$ entre les vecteurs, qui est caractérisée par, pour tout $x = (x_1, \dots, x_{|Q|})$ et $y = (y_1, \dots, y_{|Q|})$ dans $\mathbb{R}^{|Q|}$:

$$x \succ_G y \Leftrightarrow \begin{cases} \exists i = 1 \dots |Q|, x_i \neq y_i \text{ et} \\ \forall i : x_i \neq y_i, \big((x_i > y_i) \text{ ou } (\exists j \succ_Q i,\ x_j > y_j)\big) \end{cases}$$

De manière naturelle, on définit $\succsim_G$, pour tout x, y dans $\mathbb{R}^{|Q|}$, par :

$$x \succsim_G y \Leftrightarrow x = y \text{ ou } x \succ_G y.$$

La relation de Pareto-dominance est un cas particulier de $\succeq_G$ quand $\succeq_Q$ est la relation vide, c'est-à-dire quand tous les critères ont la même importance. Quand $\succeq_Q$ est linéaire, $\succeq_G$ devient la relation d'ordre lexicographique. La relation de préférence $\succeq_G$ permet donc de raffiner la relation de Pareto-dominance en introduisant une priorité sur la prise en compte des critères.

[PER 05] montre que cet exemple est une instance de AMDP et qu'il est possible de calculer les politiques non dominées par induction arrière.

5.5.5.2. AMDP *multicritères possibilistes*

Dans l'incertain possibiliste, l'utilisation des utilités optimiste et pessimiste (section 5.4.1, p. 175) a été étendue dans la prise de décision séquentielle dans le cadre

des MDP possibilistes (section 5.4, p. 175) par [SAB 98]. Nous montrons que l'emploi de l'utilité binaire possibiliste [GIA 01] qui est une unification des deux critères précédents, est également envisageable.

Avant de rappeler la définition de l'utilité binaire possibiliste, présentons le cadre de travail. L'incertain est mesuré sur un ensemble totalement ordonné $(P, \wedge, \vee, 0_P, 1_P)$ où $\wedge$ et $\vee$ sont respectivement les opérations minimum et maximum sur P. Nous définissons $(\hat{P_2} = \{\langle \lambda, \mu \rangle : \lambda, \mu \in P\}, \oplus_{\hat{P_2}}, \otimes_{\hat{P_2}}, \langle 0_P, 1_P \rangle, \langle 1_P, 0_P \rangle)$ où pour tout $\langle \alpha, \beta \rangle, \langle \lambda, \mu \rangle$ dans $\hat{P_2}$, on a :

$$\langle \alpha, \beta \rangle \oplus_{\hat{P_2}} \langle \lambda, \mu \rangle = \langle \alpha \vee \lambda, \beta \wedge \mu \rangle$$

et

$$\langle \alpha, \beta \rangle \otimes_{\hat{P_2}} \langle \lambda, \mu \rangle = \langle \alpha \wedge \lambda, \beta \vee \mu \rangle.$$

Ces deux structures sont des semi-anneaux grâce aux propriétés de $\vee$ et $\wedge$ (notamment leurs distributivités l'un sur l'autre). La loi $\oplus_{\hat{P_2}}$ induit un ordre partiel $\geq_{\hat{P_2}}$ sur $\hat{P_2}$:

$$\forall \langle \lambda, \mu \rangle, \langle \lambda', \mu' \rangle \in \hat{P_2}, \langle \lambda, \mu \rangle \geq_{\hat{P_2}} \langle \lambda', \mu' \rangle \Leftrightarrow \lambda \geq \lambda' \text{ et } \mu \leq \mu'.$$

Les récompenses sont mesurées sur $P_2 = \{\langle \lambda, \mu \rangle \in \hat{P_2} : \lambda \vee \mu = 1_P\}$, qui est un sous-ensemble de $\hat{P_2}$. Remarquons que la relation $\geq_{\hat{P_2}}$ est complète quand elle est restreinte sur P_2 et les lois $\oplus\hat{P_2}$ et $\otimes\hat{P_2}$ sont respectivement les opérations maximum et minimum sur P_2.

Rappelons que l'utilité possibiliste binaire est définie par :

$$PU(\pi) = \bigvee_{x \in X} (\pi(x) \wedge u(x)) = \langle \bigvee_{x \in X} (\pi(x) \wedge u_1(x)), \bigvee_{x \in X} (\pi(x) \wedge u_2(x)) \rangle$$

où $u : X \rightarrow P_2$ est la fonction d'utilité à valeurs dans P_2 et $\forall x \in X, u(x) = \langle u_1(x), u_2(x) \rangle$. Elle est donc une espérance généralisée avec l'opération $\oplus^g$ définie comme l'opération $\vee$ sur chaque composante et l'opération $\otimes^g$ comme $\wedge$ sur chaque composante. Remarquons que ce critère prend ses valeurs dans P_2 également.

Grâce aux propriétés de $\vee$ et $\wedge$, si les politiques sont évaluées avec ce critère binaire, alors la version algébrique de l'algorithme de Jacobi génère les politiques optimales.

Comme dans l'exemple précédent, supposons maintenant que les actions et donc les politiques sont évaluées par un vecteur d'éléments de P_2. De plus, sur l'ensemble des critères Q, supposons qu'une relation $\succeq_Q$ est définie. Alors, l'ordre strict $\succ_G$ est maintenant caractérisé par pour tout $x = (x_1, \ldots, x_{|Q|})$ et tout $y = (y_1, \ldots, y_{|Q|})$ dans ${P_2}^{|Q|}$:

$$x \succ_G y \Leftrightarrow \begin{cases} \exists i = 1 \ldots |Q|, x_i \neq y_i \\ \forall i : x_i \neq y_i, \big((x_i >_{\hat{P_2}} y_i) \text{ ou } (\exists j \succ_Q i,\ x_j >_{\hat{P_2}} y_j)\big) \end{cases}$$

De même que dans l'exemple précédent, [PER 05] ont montré que l'algorithme d'induction arrière 5.3 permet le calcul de politiques non dominées.

5.5.5.3. AMDP *dont les récompenses sont des fonctions croissantes*

Soit une échelle de valuation V possédant une structure de semi-anneau $(V, \oplus_{\mathcal{V}}, \otimes_{\mathcal{V}}, 0_{\mathcal{V}}, 1_{\mathcal{V}})$. La loi idempotente $\oplus_{\mathcal{V}}$ sert à maximiser et la loi $\otimes_{\mathcal{V}}$ sert à combiner les valeurs de l'échelle. L'ensemble H des fonctions croissantes (au sens de $\geq_{\mathcal{V}}$) sur V peut être muni de la structure de semi-anneau suivante $(H, \oplus_{\mathcal{V}}, *, \mathbf{0}, Id)$ ([MIN 77]) où

– $\forall x \in V, \forall f, g \in H, (f \oplus_{\mathcal{V}} g)(x) = f(x) \oplus_{\mathcal{V}} f(x)$

– $\forall f, g \in H, f * g = g \circ f$ (composition des fonctions)

– $\mathbf{0}$ est la fonction constante valant $0_{\mathcal{V}}$ partout

– Id est la fonction identité

Il est alors possible de construire un AMDP dont les récompenses seraient ces fonctions croissantes, c'est-à-dire que la fonction de récompense serait définie par :

– $r : \mathcal{S} \times \mathcal{A} \rightarrow H$

Dans un tel AMDP, la récompense peut donc varier. La valeur d'un historique $\gamma_t = (s_t, a_t, s_{t-1}, \ldots, a_1, s_0)$ est alors définie par :

$$r(\gamma_t) = r(s_t, a_t)\big(r(\gamma_{t-1})\big)$$

où $\gamma_{t-1} = (s_{t-1}, a_{t-1}, \ldots, a_1, s_0)$ et $r(\gamma_0)$ est une valeur fixée dépendant du problème à résoudre.

Les opérations $\oplus^g$ et $\otimes^g$ sont étendues sur H par :

$$\forall x \in V, (h \oplus^g g)(x) = h(x) \oplus^g g(x) \text{ et } (p \otimes^g h)(x) = p \otimes^g h(x)$$

pour tout $h, g \in H$ et tout $p \in P$. La proposition suivante indique que les conditions AMDP1 à AMDP5 sont vérifiées.

PROPOSITION 5.2 [WEN 06B].– *Si les conditions AMDP1 à AMDP5 sont vérifiées pour la structure V, alors ces conditions sont également vérifiées pour la structure H.*

Les conditions AMDP1 à AMDP5 étant vérifiées, nos résultats sur les AMDP peuvent être appliqués. Notamment, l'algorithme généralisé d'induction arrière 5.3 peut être exploité pour la recherche des politiques non dominées.

Nous présentons maintenant un exemple illustratif où il est naturel de modéliser les récompenses par des fonctions croissantes.

EXEMPLE.– L'exemple illustratif que nous développons dans cette section est inspiré du problème de transport de produits dangereux présenté dans [ERK 98] et [SER 06]. Supposons qu'un robot doive transporter un produit dangereux d'un point A à un point But. Les états de ce problème sont les positions que peut occuper le robot. Supposons

que l'environnement soit matérialisé par une grille $n \times m$ (voir la partie gauche de la figure 5.10). On a alors $\mathcal{S} = \{1, 2, \ldots, n\} \times \{1, 2, \ldots, m\}$. Certains états (murs,

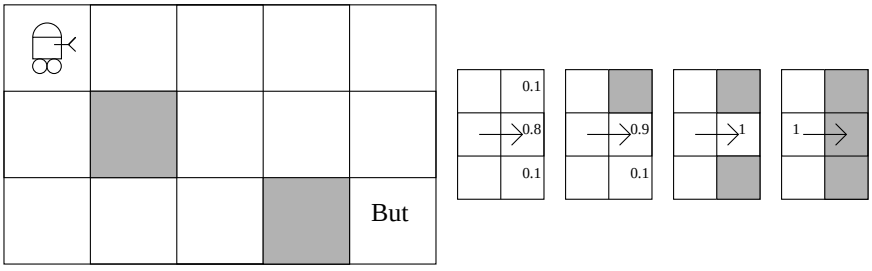


Figure 5.10. *Navigation d'un agent autonome*

obstacles,...) peuvent ne pas être accessibles. Ces informations seront intégrées dans la fonction de transition. Les actions sont constituées par les mouvements que peut effectuer le robot. Par exemple, les actions peuvent être haut, bas, gauche ou droite. Les effets des actions sont modélisés par une fonction de transition probabiliste (voir la partie droite de la figure 5.10). La réussite d'une action n'est pas assurée pour diverses raisons : le robot contrôle imparfaitement ses moteurs, le sol est glissant ou encore à cause d'événements imprévus. Les récompenses sont ici des coûts. Elles sont donc à minimiser et elles modélisent les probabilités d'un accident et son coût. À chaque déplacement, le robot risque d'avoir un accident imputant un coût c avec une probabilité p. Les coûts et les probabilités peuvent dépendre de la position et de l'action effectuée (certains endroits étant plus difficiles d'accès par exemple). Ils sont donc notés $c(s, a)$ et $p(s, a)$ pour tout $s \in \mathcal{S}$ et $a \in \mathcal{A}$.

L'approche classique dans le traitement de ce problème serait de chercher la politique qui minimiserait l'espérance de la somme des coûts sans prendre en compte l'information à disposition des probabilités d'occurrence d'un accident. Le critère serait à l'horizon h :

$$\forall s \in \mathcal{S}, v_h^\pi(s) = E_s^\pi(\sum_{t=1}^{h} C_t)$$

où E_h^π est l'espérance induite par l'application par la politique π dans l'état s et C_t est le coût encouru à l'étape t. Cette approche est discutable puisque l'on prend en compte le coût d'un accident même s'il n'a pas lieu.

Une autre approche serait de chercher la politique maximisant la probabilité de ne pas avoir d'accident. Le critère considéré à l'horizon h serait :

$$\forall s \in \mathcal{S}, v_h^\pi(s) = E_s^\pi\left(\Pi_{t=1}^{h}(1 - P_t)\right)$$

où P_t est la probabilité d'avoir un accident à l'étape t. Dans cette approche, on ne prend pas en compte les coûts des accidents. On peut vouloir faire des compromis entre coût et probabilité d'accident.

Ainsi, quand on a les deux informations (coûts et probabilités d'accident), il est possible d'adopter une meilleure approche en définissant le *risque* d'un historique.

Définition 14 *Le* risque *d'un historique* $\gamma_t = (s_t, a_t, s_{t-1}, a_{t-1}, \ldots, a_1, s_0)$ *est défini récursivement par :*

$$r(\gamma_t) = p_t c_t + (1 - p_t) r(\gamma_{t-1})$$

où $c_t = c(s_t, a_t)$, $p_t = p(s_t, a_t)$ *et* $\gamma_{t-1} = (s_{t-1}, a_{t-1}, \ldots, a_1, s_0)$ *en posant le risque d'un historique vide à* 0.

La fonction de valeur des politiques est alors définie à l'horizon h par :

$$\forall s \in \mathcal{S}, v_h^\pi(s) = \sum_{\gamma \in \Gamma_h(s)} r(\gamma) Pr_s^\pi(\gamma)$$

où Pr_s^π est la distribution de probabilité sur les historiques induite par l'application de la politique π dans l'état s.

Le problème de recherche d'une politique minimisant l'espérance du risque peut se modéliser dans les AMDP dont la fonction récompense est définie comme suit :

– $r : \mathcal{S} \times \mathcal{A} \to H$.

L'ensemble H est l'ensemble des fonctions croissantes définies sur le semi-anneau $(\mathbb{R} \cup \{+\infty\}, \min, +\infty)$.

Pour notre problème de transport de produits dangereux, on peut définir pour tout $s \in \mathcal{S}$ et pour tout $a \in \mathcal{A}$, $r(s,a)(x) = p(s,a)c(s,a) + (1 - p(s,a))x$. On constate que le risque d'un historique $\gamma_t = (s_t, a_t, s_{t-1}, a_{t-1}, \ldots, a_1, s_0)$ est alors défini par :

$$r(\gamma_t) = r(s_t, a_t)(r(\gamma_{t-1})) \quad \text{où } \gamma_{t-1} = (s_{t-1}, a_{t-1}, \ldots, a_1, s_0)$$

Le risque d'un historique est obtenu en composant successivement les récompenses (qui sont des fonctions). Dans cette section, nous avons montré que l'algorithme d'induction arrière pouvait être utilisé pour chercher les politiques minimisant l'espérance du risque.

5.6. Conclusion

Ce chapitre n'a donné qu'un bref aperçu, non exhaustif et forcément biaisé, des travaux existant sur la prise en compte de critères non-classiques dans les MDP. Néanmoins, il a montré que cette voie de recherche est vaste (de l'intégration de critères multiples à l'utilisation de critères de décision non classiques) et active.

Citons pour mémoire, [**?**], qui propose de prendre en compte plusieurs critères dans les MDP dans le cadre des MDP *à contraintes*. Un MDP à contraintes contient plusieurs fonctions de coût[8], $c_0, c_1, \ldots, c_k$. Il se donne également un ensemble de *seuils de valeur*, $C_1, \ldots, C_k$ et un ensemble de degrés de probabilité, $\varepsilon_1, \ldots, \varepsilon_k$. Résoudre un MDP à contrainte consiste à trouver, pour un état initial s_0 fixé, une politique stationnaire *non déterministe* minimisant la fonction de valeur du MDP définie par rapport à c_0, sous les contraintes que la probabilité que la valeur en s_0 de chacune des fonctions de valeurs définies à partir des fonctions de coût $c_i, i \geq 1$ soit supérieure à C_i n'excède pas ε_i.

Les utilités qualitatives possibilistes ne sont pas les seuls critères d'utilité non classiques à avoir été étendus à la décision séquentielle. De manière plus générale, un certain nombre de travaux [**?**, **?**] visent à exploiter dans le cadre de la prise de décision séquentielle des modèles plus riches en termes de pouvoir descriptif. Ces modèles décisionnels, tels que le critère RDU [**?**], l'intégrale de Choquet [**?**] ou l'intégrale de Sugeno [**?**], développés en théorie de la décision ne peuvent s'exprimer sous la forme d'une espérance généralisée et ne rentrent donc pas dans le formalisme des AMDP. En effet, ils sont connus pour ne pas être dynamiquement cohérents [**?**]. L'utilisation de ces modèles décisionnels en décision séquentielle représente ainsi un problème difficile car le principe d'optimalité de Bellman n'est plus vérifiée.

8. Un MDP à contrainte peut être défini de manière équivalente avec des fonctions de récompense mais ce cadre a été défini par un automaticien et reprend la formulation "minimisation" de coût, habituelle dans ce domaine.

DEUXIÈME PARTIE

Exemples d'application des (PO)MDP

Chapitre 6

Apprentissage en ligne de la manipulation de micro-objets

6.1. Introduction

Ce chapitre présente l'application d'un algorithme d'apprentissage par renforcement à l'apprentissage *en ligne* de la manipulation de micro-objets [ADD 05]. L'originalité de cette application tient au fait que la politique d'action a été apprise non pas sur une simulation mais au travers du pilotage du *processus réel*. Ces travaux ont été effectuées au Laboratoire d'Automatique de Besançon dont une des spécialités est la *microrobotique*. D'autres travaux sur l'usage de l'apprentissage par renforcement en microrobotique sont décrits dans [LAU 02].

La microrobotique a pour objectif général de concevoir, réaliser et commander des systèmes robotiques compacts destinés à manipuler des objets de dimensions typiquement comprises entre un millimètre et un micromètre pour diverses applications (instrumentation, applications industrielles, biomédicales).

Compte tenu des dimensions et des précisions recherchées, la microrobotique se heurte à des difficultés de mise en œuvre différentes de celles de la robotique classique :

– au niveau des actionneurs : la microrobotique fait appel à de nouveaux principes d'actionnement plus compacts, notamment fondés sur l'utilisation de matériaux actifs ; de tels actionneurs sont bien souvent fortement non linéaires ;

– au niveau des capteurs : le volume réduit des applications rend difficile la mise en place de capteurs en nombre suffisant ; l'emploi d'un système de vision (via un

---

Chapitre rédigé par Guillaume LAURENT.

microscope) est souvent le principal moyen d'observation et de mesure ;

– au niveau des interactions entre le robot et les objets qui l'entourent : à cette échelle, les forces de surface deviennent prépondérantes sur les forces volumiques ; l'inertie des objets est très faible, la friction entre objets engendre des frottements secs importants et difficiles à quantifier, en dessous de 100 µm les forces d'adhésion (capillarité) et de Van Der Waals font que les objets «collent» les uns aux autres ; ces phénomènes rendent les manipulations extrêmement délicates et hasardeuses.

La non-linéarité des actionneurs, l'imprécision des capteurs, la complexité des forces de surface, rendent les processus difficiles à modéliser. En l'absence de modèle précis, la synthèse de contrôleurs par les approches traditionnelles de l'automatique est difficile. A contrario, l'apprentissage par renforcement permet de s'affranchir de tout modèle du processus contrôlé et de prendre en compte le caractère hasardeux de ce processus via une approche stochastique. Ces méthodes de contrôle sont donc adaptées à la microrobotique.

Ce chapitre est structuré en trois sections. La première présente le contexte de la micro-manipulation par poussée ainsi que le dispositif à piloter. La deuxième décrit l'algorithme d'apprentissage par renforcement employé pour le contrôle du manipulateur. Enfin, les résultats expérimentaux sont présentés dans la dernière section.

6.2. Dispositif de manipulation

6.2.1. *Objectif : le micro-positionnement par poussée*

Dans l'industrie, le positionnement est une fonction essentielle pour l'usinage ou l'assemblage de pièces. Dans le domaine de la microrobotique, les solutions classiques comme la « prise-dépose » ne sont pas transposables directement. Dans ces conditions, il est souvent plus aisé de pousser un micro-objet que de le tenir dans une pince. Ainsi, de nombreux travaux utilisent cette approche de micro-positionnement par poussée [BAU 98, ZES 98].

Si ces manipulateurs sont plus simples à concevoir, il n'en est pas de même pour la commande. En effet, le problème qui consiste à prévoir le mouvement d'un objet poussé en un point donné est déjà complexe à l'échelle macroscopique [PES 88]. En dessous du millimètre, les équations classiques de frottement fondées sur le poids des objets (loi de Coulomb) ne s'appliquent plus du fait de l'importance prépondérante des forces de surfaces. Le mouvement d'un objet poussé dépend alors de paramètres divers comme l'état de surface du support et de l'objet, l'humidité de l'air, la répartition des charges électrostatiques, etc. Dans ces conditions, le mouvement de l'objet est impossible à prédire et c'est ainsi qu'une approche de pilotage par apprentissage par renforcement a été envisagé. L'objectif consiste donc en la synthèse par apprentissage de politiques de contrôle performantes d'un manipulateur réalisant des tâches de micro-positionnement par poussée.

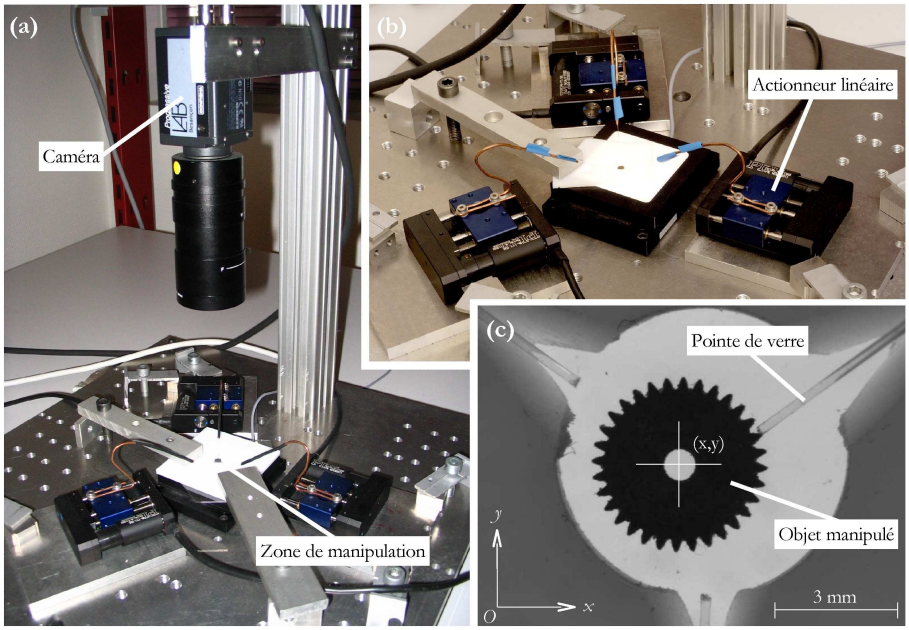


Figure 6.1. *Dispositif de manipulation : (a) vue d'ensemble, (b) zoom sur la zone de manipulation entourée des trois actionneurs de poussée, (c) détail de la zone de manipulation (image de la caméra). La position de l'objet est repérée par le système de vision.*

6.2.2. *Dispositif de manipulation*

Le dispositif de manipulation piloté est inspiré de dispositifs de manipulation existants qui permettent de pousser et tirer des nano-particules une à une à l'aide de pointes fines ou du levier d'un microscope à force atomique [Zyv 06, RES 00, HAN 98c]. En revanche, l'échelle du dispositif décrit ici est tout autre puisqu'il s'agit de positionner des objets de taille millimétrique comme des pignons de montres. Ce dispositif est un banc d'essais ayant pour objectif de démontrer la faisabilité de la commande par apprentissage par renforcement pour le micro-positionnement par poussée.

Le manipulateur est équipé de trois pointes de verre montées sur des actionneurs linéaires asservis en position avec une précision de l'ordre du micromètre (cf. figure 6.1b). L'objet à manipuler est posé sur une lame de verre entre les trois pointes (cf. figure 6.1c). Chaque pointe peut venir en contact de l'objet puis exercer sur lui une poussée sur une distance donnée. L'objectif est de déplacer cet objet vers une position donnée par une séquence adéquate de poussées.

Les axes de déplacements des pointes sont concourants. La zone de manipulation est entourée d'une paroi circulaire empêchant l'objet manipulé de s'échapper. Cette

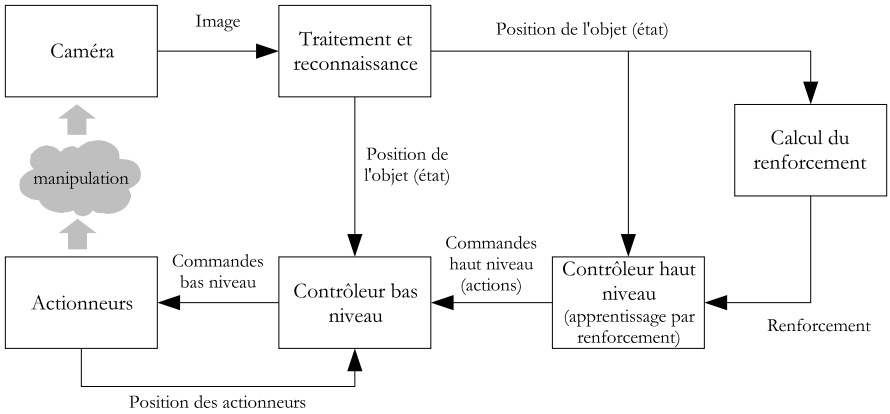


Figure 6.2. *Boucle sensori-motrice.*

disposition permet *a priori* de déplacer un objet cylindrique (de révolution) dans n'importe quelle position (voire dans n'importe quelle orientation mais cela n'est pas encore traité).

La position de l'objet est mesurée via un système de vision (cf. figure 6.1a). L'objet est repéré par son abscisse et son ordonnée dans l'image vidéo. La résolution de la caméra étant limitée, la précision de localisation est de l'ordre de 23 µm.

6.2.3. *Boucle de commande*

Pour le contrôle du manipulateur, deux niveaux de commande sont utilisés : un bas niveau et un haut niveau (cf. figure 6.2). Le bas niveau gère l'asservissement en position des pointes (par des méthodes traditionnelles d'automatique). Il permet d'amener une pointe en contact de l'objet puis de pousser celui-ci sur une distance spécifiée par la commande haut niveau. Le haut niveau est l'organe de décision qui planifie les poussées sur le long terme pour amener l'objet à une position donnée.

La durée d'une action de haut niveau (i.e. d'une poussée) est variable : de une à trois secondes. A la fin d'une poussée, l'objet s'arrête immédiatement car son inertie est très faible. Ainsi, seul le comportement statique de l'objet est pris en compte et son comportement dynamique est négligé.

6.2.4. *Représentation du système de manipulation sous la forme d'un* MDP

6.2.4.1. *Définition de l'espace d'état*

Le système étant considéré comme statique, son état s est simplement défini par la position (x, y) du centre de l'objet dans l'image vidéo.

La zone de manipulation ayant un diamètre de 7 mm et l'objet un diamètre de 4 mm, la position du centre de l'objet peut évoluer dans un disque de diamètre 3 mm. Etant donné la faible résolution du capteur vidéo, la caméra permet de localiser le centre de l'objet dans une zone circulaire de diamètre 131 pixels soit dans environ 13 500 positions différentes ($\mathrm{card}\,\mathcal{S} = 13\,500$).

6.2.4.2. *Définition de l'espace d'action*

Dans les expériences menées, seules 6 actions distinctes ont été utilisées. Chacune des trois pointes peut pousser l'objet sur deux distances déterminées : une poussée « longue » de 1 mm utile pour les grands déplacements ou une poussée « courte » de 100 µm indispensable au positionnement fin. On a donc : $\mathrm{card}\,\mathcal{A} = 6$.

Cet ensemble de 6 actions est le résultat d'un compromis entre qualité des trajectoires obtenues et temps d'apprentissage. Plus de variété dans les actions permettrait sans doute une manipulation plus rapide mais augmenterait considérablement les possibilités d'exploration et donc le temps d'apprentissage global.

6.2.4.3. *Définition de la fonction de renforcement*

L'objectif de la manipulation est d'amener l'objet dans une position déterminée avec une précision donnée. Dans les expériences réalisées, le but était de positionner l'objet au centre de la zone de manipulation à 140 µm près (soit 6 pixels caméra). L'ensemble des états se situant à une distance de la position cible inférieure à la précision requise est noté $\mathcal{S}_{cible}$.

Ainsi, le but est de conduire le processus d'un état quelconque vers un des états de $\mathcal{S}_{cible}$. La fonction de renforcement est donc définie de la manière suivante :

$$r(s, a, s') = \begin{cases} 1 \text{ si } s' \in \mathcal{S}_{cible} \\ 0 \text{ sinon} \end{cases} \tag{6.1}$$

6.2.4.4. *Définition d'un épisode*

Avant chaque manipulation, l'objet est posé dans une position quelconque de la zone de manipulation (i.e. l'état initial est aléatoire). Le manipulateur passe alors sous contrôle de l'algorithme de commande de haut niveau, en l'occurrence un algorithme d'apprentissage par renforcement et un épisode d'apprentissage commence. L'épisode se termine quand l'objet a atteint un état cible.

6.3. Choix de l'algorithme d'apprentissage par renforcement

6.3.1. *Caractéristiques du* MDP

Le système de manipulation étant particulièrement lent (une poussée toute les une à trois secondes), il est impératif de réduire au maximum le nombre d'épisodes nécessaires à l'obtention d'une bonne politique d'action. En outre, il est possible d'effectuer de nombreux traitements hors ligne entre deux poussées.

Le système de localisation par vision fournit une observation discrète de l'état du système. Vu la faible résolution du capteur vidéo, il y a une imprécision notable sur

la mesure de l'état du processus : il n'est pas complètement observable. Malgré tout, l'observation est suffisante pour considérer que le processus est discret, complètement observable et légèrement stochastique (bruit de mesure).

Pour ces raisons, une méthode discrète et indirecte (cf. chapitre 2) a été utilisée afin d'exploiter au mieux, pendant le temps d'exécution des poussées, toutes les interactions passées entre l'algorithme et le processus.

L'algorithme indirect le plus classique est *Dyna-Q* [SUT 90a] (cf. chapitre 2). Dans sa version originale, *Dyna-Q* gère des processus déterministes, ce qui signifie que la même action dans le même état produit toujours le même état suivant. Par conséquent, dans un cas déterministe, le modèle des transitions peut être représenté sous la forme d'une matrice de $|\mathcal{S}|$ lignes par $|\mathcal{A}|$ colonnes, chaque case contenant l'état suivant. Une extension de *Dyna-Q* aux processus stochastiques est immédiate, mais on a alors pour chaque couple $s \times a$ une distribution de probabilité sur tous les états suivants, donc il faut une matrice de taille $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ pour stocker les probabilités. Dans notre cas, avec 13 500 états et 6 actions, une telle représentation n'est pas gérable. De même, *Prioritized Sweeping* permet l'apprentissage d'une politique avec un processus stochastique mais nécessite la mémorisation des relations d'antécédence sous la forme d'une matrice de $|\mathcal{S}| \times |\mathcal{A}|$ lignes par $|\mathcal{S}|$ colonnes. Chaque état ayant souvent de nombreux antécédents même pour un système déterministe, la matrice est généralement bien remplie. Vu le nombre d'états du processus à contrôler, cette méthode, pourtant efficace, n'est pas applicable non plus.

Pour ces raisons, un algorithme plus économe en mémoire est proposé et utilisé pour le contrôle du manipulateur.

6.3.2. *Un algorithme adapté :* STM-Q

L'algorithme utilisé, appelé *STM-Q* (Short-Term Model-based Q-Learning), est une extension de *Dyna-Q* aux systèmes stochastiques qui utilise une représentation de taille mémoire intermédiaire entre le cas déterministe et le cas stochastique exhaustif (cf. algorithme 32) [ADD 05]. C'est un algorithme indirect qui, comme *Dyna-Q*, recherche une politique optimale vis-à-vis du critère γ-pondéré.

En tant que méthode indirecte, *STM-Q* construit au fur et à mesure de ses interactions avec le processus un modèle des transitions et des renforcements. Ce modèle est constitué d'un tableau de files (FIFO) qui stockent, pour chaque couple état-action visité, les différents résultats observés par le passé : à chaque couple $s \times a$ est associé une file $M(s,a)$ dont la taille maximale ne peut dépasser un nombre déterminé avant l'apprentissage, noté n_{max}. Chaque élément de $M(s,a)$ contient un couple constitué de l'état suivant et de la récompense reçue en faisant l'action a dans l'état s. Par exemple, si $n_{max} = 4$ et que le couple $s \times a$ a été visité au moins 4 fois, on peut avoir une file de la forme :

$$M(s,a) = \left\{(s'_{t_1}, r_{t_1}), (s'_{t_2}, r_{t_2}), (s'_{t_3}, r_{t_3}), (s'_{t_4}, r_{t_4})\right\} \tag{6.2}$$

Algorithme 6.1 : STM-Q (Short-Term Model-based Q-Learning).

initialisation
  pour chaque $(s, a) \in \mathcal{S} \times \mathcal{A}$ **faire**
    $Q(s, a) \leftarrow Q_0$
    $M(s, a) \leftarrow \emptyset$
    `/* M(s,a) est la file des couples état-renforcement observés lors des visites passées du couple (s,a) */`

pour chaque *épisode* **faire**
  $s \leftarrow$`ChoixEtat`
  tant que s *n'est pas un état terminal* **faire**
    $a \leftarrow$`ChoixAction(`Q, s`)`
    Effectuer l'action a, observer le nouvel état s' et le renforcement r

    si $\operatorname{card} M(s, a) < n_{max}$ **alors**
      Ajouter l'observation (s', r) à la file $M(s, a)$
    sinon
      Remplacer l'observation la plus ancienne de la file $M(s, a)$ par l'observation (s', r)

    $s \leftarrow s'$

    répéter N **fois**
      `/* cette partie peut être effectuée hors ligne */`
      choisir au hasard un couple (s, a) déjà visité
      $Q(s, a) \leftarrow \sum_{\forall (y,r) \in M(s,a)} \frac{1}{\operatorname{card} M(s,a)} \left[r + \gamma \max_{v \in \mathcal{A}} Q(y, v) \right]$

avec $t_1 < t_2 < t_3 < t_4$. Ces couples ont évidemment des valeurs différentes si le processus n'est pas déterministe.

Le modèle ainsi constitué permet de calculer une estimation des probabilités de transition du processus (maximum de vraisemblance) :

$$\widehat{p}(s'|s, a) = \sum_{\forall (y,r) \in M(s,a) | y = s'} \frac{1}{\operatorname{card} M(s, a)} \tag{6.3}$$

Cette estimation est utilisée pour optimiser la fonction de valeur $Q(s, a)$ selon une procédure similaire à un algorithme d'itération sur les valeurs (cf. chapitre 1). L'équation de mise à jour est alors celle qui apparaît à la dernière ligne de l'algorithme 32.

La taille n_{max} de la file d'attente permet d'adapter l'algorithme au « degré de non déterminisme » du processus. Par souci d'efficacité, n_{max} doit rester faible (de 10 à 50 environ), ainsi *STM-Q* est plutôt adapté au contrôle de processus faiblement stochastiques comme notamment les systèmes physiques vus au travers de capteurs numériques et ayant un espace d'état de dimension faible (2 à 3).

6.4. Résultats expérimentaux

6.4.1. *Mise en œuvre*

Comme expliqué dans la section , l'objectif est de sélectionner les poussées permettant d'amener l'objet à une position donnée le plus rapidement possible. L'algorithme *STM-Q* a donc été implanté dans le contrôleur de haut niveau (cf. figure 6.2). Il reçoit l'état du processus *via* le système de vision et envoie ses commandes au contrôleur bas niveau (choix de la pointe et longueur de poussée).

L'expérience consiste à apprendre à positionner un objet vers le centre de la zone de manipulation à partir d'une position quelconque. Au début de chaque épisode, l'objet est placé au hasard dans la zone et l'épisode se termine quand l'objet est correctement positionné.

La méthode d'exploration retenue pour `ChoixAction(`Q, s`)` dans l'algorithme est la méthode ϵ-greedy (cf. chapitre 2).

Avant le premier épisode, la fonction de valeur est initialisée à $Q_0 = 0$. Durant l'expérience, les valeurs des paramètres de l'algorithme sont :

– $\epsilon = 0.1$,

– $\gamma = 0.9$,

– $n_{max} = 10$,

– N égal au nombre de couples état-action distincts précédemment visités (sur tous les épisodes), c'est-à-dire :

$$N = \sum_{\forall (s,a) \in \mathcal{S} \times \mathcal{A} | M(s,a) \neq \emptyset} 1 \tag{6.4}$$

6.4.2. *Résultats obtenus*

L'expérience a duré un peu plus de 24 heures et totalise 34 134 poussées réparties sur 100 épisodes. La courbe de la figure 6.3 représente le nombre de poussées (i.e. d'actions) effectuées à chaque épisode.

Au début de l'apprentissage, entre 400 et 500 poussées en moyenne sont nécessaires pour que l'objet soit positionné. L'écart-type est très important (environ 500). La politique consiste principalement à explorer l'espace d'état.

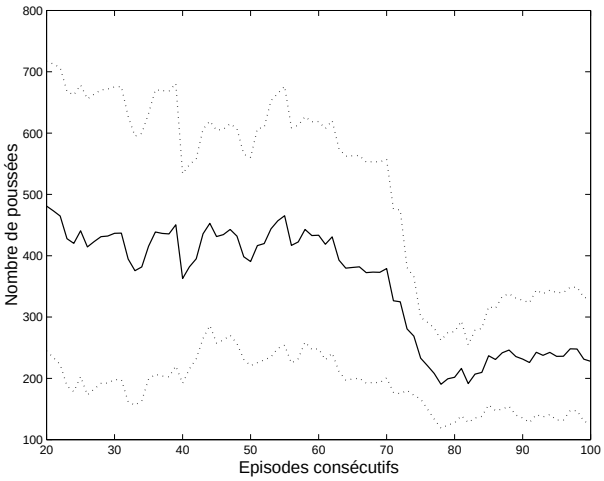


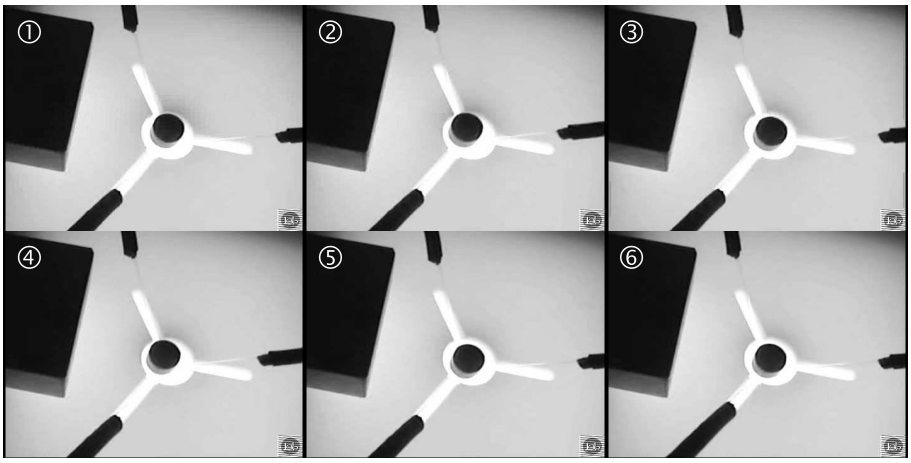
Figure 6.3. *Résultat de l'apprentissage en ligne : en trait plein, le nombre moyen de poussées par épisode (moyenne glissante sur les 20 derniers épisodes consécutifs), en traits pointillés, l'écart-type centré sur la moyenne.*

Au bout de 80 épisodes, on note une baisse significative de la durée des épisodes. Il faut en moyenne 250 poussées pour positionner l'objet, mais l'écart-type reste élevé (environ 200). On peut néanmoins obtenir des épisodes très courts comme en témoigne la figure 6.4b.

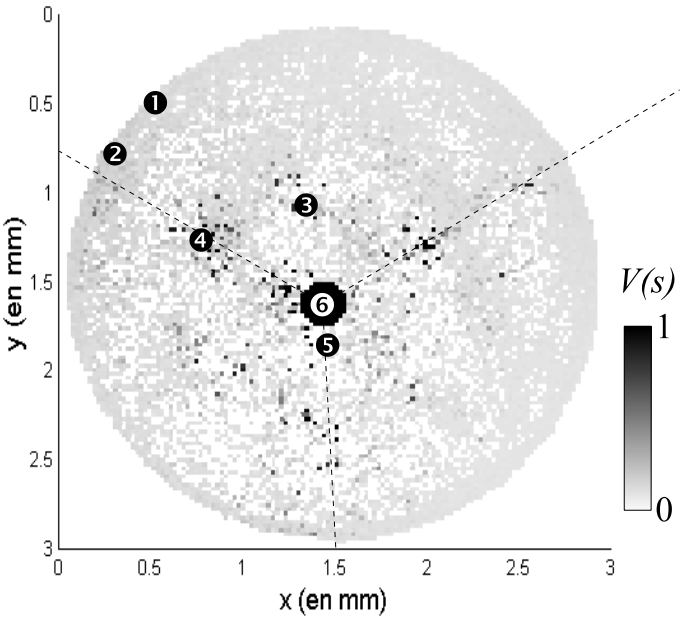
L'observation de séquences plus longues montrent que le pilotage est particulièrement délicat pour deux raisons principales. D'une part, quand l'objet est entre deux pointes contre la paroi, il est difficile de le déloger. La figure 6.4b montre que la fonction de valeur est bien évaluée sur le pourtour par rapport à l'intérieur. D'autre part, pour atteindre l'objectif, il ne suffit pas de rapprocher l'objet du centre : il faut rester dans l'axe d'une pointe et, si l'objectif est manqué de peu, il est souvent nécessaire de recommencer la manipulation « à zéro » en repoussant l'objet contre la paroi. Ces deux observations se traduisent dans la fonction de valeur par des pics situés sur les axes des pointes à environ une poussée de la cible et par des valeurs faibles sur une couronne proche de l'objectif.

6.5. Conclusion

Cette expérience montre qu'il est possible d'utiliser un algorithme d'apprentissage par renforcement indirect pour apprendre en ligne à contrôler un processus réel. Néanmoins, en ce qui concerne le manipulateur étudié, les résultats obtenus sont encore loin d'une automatisation fiable et efficace. Certes, en raisons du contexte « micro », la tâche de manipulation est ardue même pour un opérateur humain entraîné. Mais ces



(a) Séquence de manipulation : ① état initial, ② poussée longue de la pointe de droite, ③ poussée longue de la pointe du haut, ④ poussée longue de la pointe de droite, ⑤ poussée longue de la pointe du haut, ⑥ poussée courte de la pointe du bas. Les pointes sont peu visibles car elles sont fines et transparentes (fibres optiques).



(b) Représentation de la fonction de valeur V et des états successivement atteints lors de la séquence de manipulation. Les points blancs représentent les états non visités.

Figure 6.4. *Exemple d'épisode de manipulation obtenue à l'issu de l'apprentissage.*

	Au total	Visité
Nombre d'états	13 500	8 748
Nombre de couples état-action	81 000	10 700
Nombre d'actions par état	6	1 action dans 7 138 états 2 actions dans 1 138 états plus de 3 actions dans 375 états

Tableau 6.1. *Statistiques de visite des états et des actions à l'issue l'apprentissage en ligne (qui totalise 34 134 actions).*

résultats mitigés confirment aussi un besoin classique en apprentissage : la généralisation.

L'algorithme employé, *STM-Q*, construit un modèle du processus afin de mettre à jour la fonction de valeur le plus rapidement possible, mais ne généralise pas la valeur d'un état à un autre état proche comme le montre l'aspect très parsemé des valeurs obtenues (cf. figure 6.4b).

De plus, les statistiques de visite des états (cf. tableau 6.1) indiquent qu'à peine plus d'un huitième des couples état-action ont été visités lors de l'expérience. Comme il n'est pas raisonnable de prolonger la durée, déjà importante, de ce genre d'apprentissage, un mécanisme de généralisation serait nécessaire pour réutiliser l'expérience acquise pour les couples non visités. Ainsi, au-delà de cette application de manipulation, l'apprentissage en ligne du pilotage de processus réel doit faire face à un double enjeu : utiliser au maximum l'expérience passée via des approches indirectes et généraliser l'expérience acquise pour répondre de la meilleure façon possible dans un état inconnu.

Chapitre 7

Conservation de la biodiversité

7.1. Introduction

La biodiversité terrestre et marine est de plus en plus menacée par la pression grandissante de l'activité humaine. Croissance de la population mondiale, urbanisation, industrialisation des pays en voie de développement et exploitation non raisonnée des ressources naturelles sont autant de causes de disparition d'espèces vivantes constituant la biodiversité de la planète. Face à ces changements environnementaux (déforestation, érosion, pollution), les espèces survivantes sont condamnées à s'adapter rapidement ou bien à disparaître. La biologie de la conservation est un domaine de l'écologie qui se donne pour objectif la protection de la biodiversité. Jadis expérimentales, les recherches actuelles en biologie de la conservation se tournent vers l'étude de la gestion optimale des efforts de conservation. L'augmentation du nombre d'espèces menacées [ANO 07] et les faibles crédits disponibles pour les protéger sont autant d'arguments forts pour optimiser les décisions de conservation et améliorer les actions de sauvegarde de la biodiversité [POS 01]. Dans ce contexte, les processus décisionnels de Markov permettent une formulation claire de ces problèmes d'optimisation.

Dans ce chapitre, on propose d'étudier deux applications des méthodes d'optimisation MDP. Le premier problème concerne la conservation d'espèces menacées difficilement observables. Au fur et à mesure qu'une population diminue, les individus menacés deviennent de plus en plus difficiles à détecter. Les gestionnaires de réserves abritant des espèces menacées font face au dilemme suivant : si je ne suis pas certain que l'espèce est présente, dois-je continuer à protéger cette espèce, ou dois-je investir

---

Chapitre rédigé par Iadine CHADÈS.

mes ressources limitées dans la surveillance de cette espèce ? Nous[1] avons étudié le cas particulier du tigre de Sumatra (*Panthera tigris sumatrae*). Ce travail constitue la première application de POMDP dans le domaine de la conservation de la biologie.

La deuxième application concerne la conservation de deux espèces menacées en interaction (proie-prédateur). Les abalones du Nord (*Haliotis kamtschatkana*) constituent le régime alimentaire préféré des loutres de mer (*Enhydra lutris*). A l'heure actuelle, les stratégies de conservation de ces espèces sont gérées de manière indépendante et ne prennent pas en compte leurs interactions. Nous[2] avons soulevé ce problème et étudié l'optimisation de la gestion des décisions de conservation de ces deux espèces en utilisant deux algorithmes d'apprentissage par renforcement à horizon fini.

7.2. Protéger, surveiller ou abandonner : gestion optimale d'espèces secrètes et menacées

7.2.1. *Surveillance et gestion du tigre de Sumatra*

Le tigre de Sumatra, comme toutes les espèces de tigre, souffre d'un déclin dramatique de sa population, conséquence de l'appauvrissement de son habitat en proies, de la destruction de son habitat et de l'activité de braconnage [LIN 06]. Dans la région de Kerinci Seblat, Linkie et al. ont conduit une étude sur les conséquences des ressources investies dans la mise en place de patrouilles anti-braconnage sur la probabilité d'extinction de la population de tigre de Sumatra [LIN 06]. Les efforts actuels de conservation de cette espèce incluent la réduction de l'activité de braconnage par des gardes et l'analyse du statut de la population par la surveillance. Actuellement, 30 000 dollars sont dépensés chaque année pour effectuer ces deux actions avec approximativement 2/3 de ce budget consacré aux patrouilles (c_m) et le tiers restant investi dans la surveillance (c_s). Nous avons estimé le coût potentiel de l'échec de la conservation d'une population viable de tigres à 175 134 dollars par année (V) sur la base des fonds récoltés par le programme de protection du tigre de Sumatra. Nous avons déterminé la probabilité d'extinction locale de la population de tigres par année lorsque le parc est protégé ($p_m = 0,058$) et lorsqu'il n'est pas protégé ($p_o = 0,1$). De manière similaire, nous avons déterminé la probabilité de détection de tigres vivants dans cette réserve à $0,782$ (d) lorsque l'on conduit une surveillance et $0,001$ sinon.

Nous avons modélisé ce problème à l'aide d'un processus décisionnel de Markov partiellement observable (POMDP).

---

1. E. McDonald-Madden, M. McCarthy, B. Wintle, M. Linkie et H. Possingham sont les co-auteurs de ces travaux voir [CHA 08]

2. T. Martin, J. Curtis, C. Barreto sont les co-auteurs de ces travaux voir [**?**]

7.2.2. *Modèle*

Posons $\mathcal{S} = \{E, V\}$ l'ensemble fini des états de notre système qui qualifie l'état de notre population de tigres comme *éteinte* ou *viable*. L'ensemble fini des actions $\mathcal{A}$ comporte trois actions $\mathcal{A} = \{S, P, N\}$ qui définit nos décisions de conservation, respectivement *surveiller*, *protéger* et *abandonner* (ne rien faire). A chaque action est associée une matrice de transition T_a qui définit pour chaque paire état-action une distribution de probabilité sur $\mathcal{S}$. Dans notre problème nous faisons l'hypothèse que la population de tigres, une fois éteinte, ne peut être recolonisée, ainsi l'extinction de la population est définitive. La fonction de récompense prend en compte le coût des actions (c_m, c_s ou 0) et les bénéfices des états (V ou 0). Enfin, nous prenons en compte l'observabilité partielle en représentant la probabilité de détection d'un tigre en fonction de l'état et de l'action choisie à l'aide d'un ensemble de matrices d'observation O. On dénote $\Omega = \{A, Pr\}$ l'ensemble fini des observations du système, respectivement *absent* et *présent*.

Le critère d'optimisation choisi est la maximisation de l'espérance des gains à horizon fini comme défini par l'équation (3.13) du chapitre 3. Nous avons utilisé l'algorithme élagage incrémental (*Incremental Pruning*) [CAS 98] pour déterminer la stratégie de conservation optimale. Cet algorithme est disponible dans la boîte à outils POMDP de Cassandra [CAS 05].

7.2.3. *Résultats*

La politique optimale est représentée ici sous deux formes : avec un graphe de décision (figure 7.1) et comme une fonction de deux variables représentant la politique optimale directement (figure 7.2). Il est optimal de *protéger* le tigre de Sumatra pendant 12 ans s'il n'est pas observé. Puis, si le tigre reste non observé, l'action *surveiller* est optimale pendant 3 ans, avant d'envisager d'autres actions de conservation (*abandonner*).

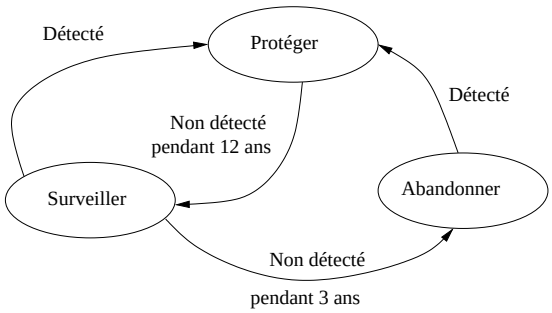


Figure 7.1. *Graphe de décision*

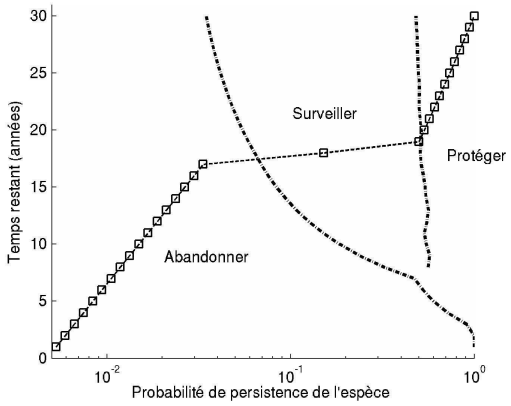


Figure 7.2. *Représentation de la politique optimale en fonction de la probabilité de croyance de persistence de l'espèce. La ligne de carrés représente une simulation de la stratégie optimale lorsque le tigre de Sumatra demeure non observée.*

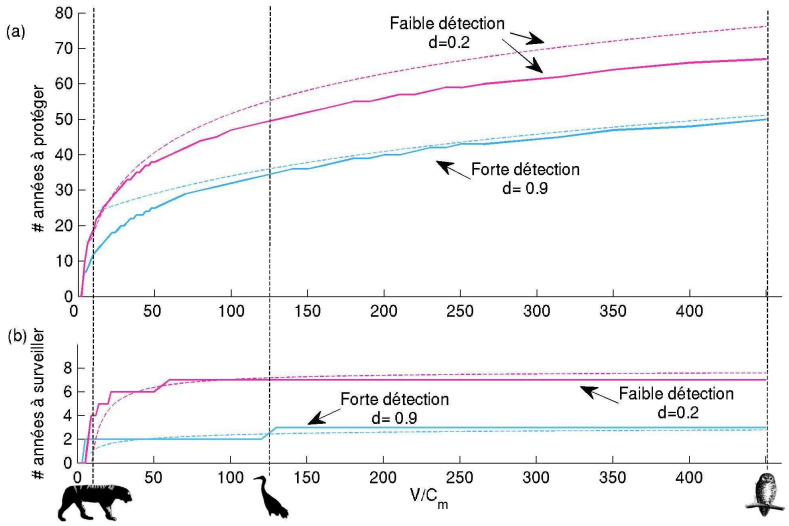


Figure 7.3. *Influence de la probabilité de détection (d) sur la stratégie optimale et le rapport valeur économique de l'espèce/coût de protection (V/c_m). Les courbes continues illustrent les stratégies optimales numériques, les courbes pointillées représentent les stratégies calculées par approximation analytique.*

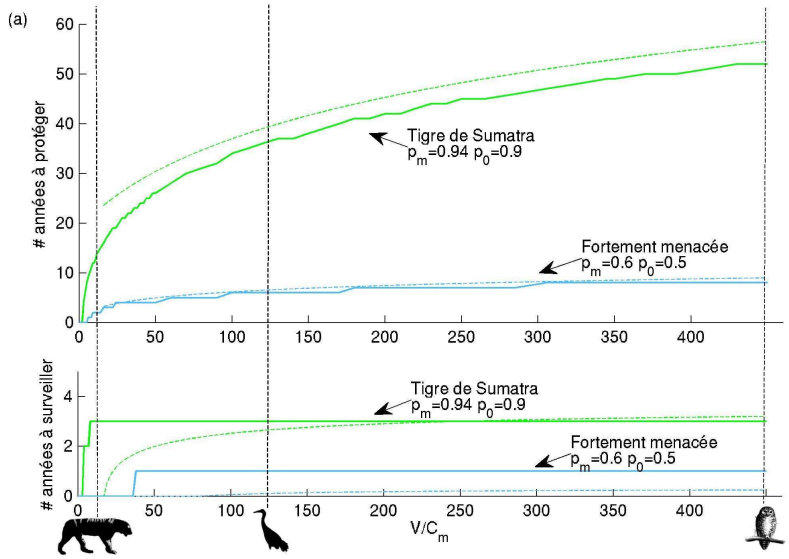


Figure 7.4. *Influence des probabilités d'extinction (p_m et p_0) sur la stratégie optimale et le rapport valeur économique de l'espèce/coût de protection. Les courbes continues illustrent les stratégies optimales numériques, les courbes pointillées représentent les stratégies calculées par approximation analytique.*

Sur la figure 7.2 nous avons représenté la politique optimale pour un horizon de 30 ans. On distingue 3 régions, chacune d'elle nous informe sur la décision optimale à prendre en fonction du temps restant et de l'état de croyance courant. A titre d'exemple, nous avons simulé un scénario (ligne de carrés) qui illustre la décroissance de la valeur de l'état de croyance que le tigre persiste s'il n'est pas observé pendant 30 ans en réalisant l'action optimale à chaque pas de temps. Nous avons conduit une étude de sensibilité intensive des paramètres importants de notre modèle : la fonction de coût et de récompense (figures 7.3 et 7.4), l'influence des paramètres de détection (figure 7.3), ainsi que la probabilité d'extinction locale (figure 7.4). Cette étude nous permet de généraliser nos résultats à d'autres espèces menacées comme par exemple la grue blanche (*Grus americana*) ou encore la chouette tachetée Mexicaine (*Strix occidentalis lucida*).

Les politiques optimales obtenues sont caractéristiques et suivent le même modèle, seules les durées de protection et de surveillance varient. Nous pouvons en déduire des règles de gestion de réserves naturelles. Le résultat principal s'énonce simplement : plus l'espèce est menacée, plus il est optimal d'investir dans la protection active de l'espèce, au point que l'action de surveillance peut ne pas apparaître comme une décision optimale (figure 7.4). Cela s'explique par l'extrême urgence de la situation :

les valeurs économiques ont moins d'influence sur la stratégie optimale ; le temps devient la ressource critique pour la sauvegarde de l'espèce. Enfin, à la lecture des graphes solutions, il apparaît qu'il est possible d'approcher la solution analytique de ce problème. Les valeurs de croyance pour lesquelles il devient optimal de changer de décision (*protéger* vers *surveiller*, et *surveiller* vers *abandonner*) peuvent s'exprimer en fonction des paramètres de notre problème. Ils nous permettent également d'approcher les périodes de temps correspondant à chaque phase. Le lecteur intéressé pourra se reporter à l'article [CHA 08]. L'intérêt d'approcher analytiquement la solution optimale est double. Cette approximation permet de mieux comprendre les effets de chacun des paramètres de notre modèle mais surtout les gestionnaires de réserves naturelles disposent maintenant d'un outil simple pour déterminer la politique optimale qu'ils peuvent suivre sans avoir à résoudre un nouveau POMDP. La qualité de cette approximation est représentée par les lignes discontinues des figures 7.3 et 7.4.

Nous avons formulé en termes simples le problème de décider quand on doit *protéger*, *surveiller* et arrêter la gestion d'une réserve naturelle pour une espèce secrète menacée. Nous avons résolu ce problème de manière exacte en utilisant un algorithme de résolution de POMDP et de manière approchée en développant la solution analytique. L'étude intensive des paramètres de ce problème nous permet d'établir une conduite à suivre pour les gestionnaires de parcs et de réserves. De manière générale, ce travail illustre comment les POMDP, bien que de complexité algorithmique souvent rédhibitoire, peuvent être appliqués dans un problème d'optimisation du compromis recherche d'information/exploitation de l'information.

7.2.4. *Extension à plusieurs populations*

L'extension à plusieurs populations est naturelle si l'on considère la réalité du terrain. Plusieurs réserves fragmentent l'île de Sumatra et les ressources sont limitées. Nous avons donc étudié quelle serait la stratégie de conservation optimale pour deux populations. Cette fois-ci, l'efficacité des actions est proportionnelle à la charge de travail : la protection simultanée des deux populations est moins efficace d'un point de vue individuel que dans le cas où une population bénéficie de l'ensemble des ressources. Nous avons étudié le critère d'optimisation qui maximise le nombre de populations de tigres viables, et nous avons étudié comment la qualité de l'habitat influençait la performance des actions. Certaines réserves plus vastes ont une probabilité de persistance plus élevée qu'une réserve plus petite. Par conséquent il nous faut trouver les stratégies optimales adaptées à ces conditions.

Nous avons de nouveau étudié ce problème en utilisant un POMDP multi-agent en distinguant l'état de la population a et de la population b : $\mathcal{S} = \mathcal{S}_a \times \mathcal{S}_b$. L'ensemble fini des actions $\mathcal{A}$ comporte 5 actions $\mathcal{A} = \{P_a, P_b, P_{ab}, P_aS_b, S_aP_b\}$ qui définit nos décisions de conservation (respectivement protéger a, protéger b, protéger a et b, protéger a et surveiller b, surveiller a et protéger b). La probabilité de transition associée à chaque action est représentée par un ensemble de matrices de transition T définissant pour chaque paire état-action une distribution de probabilité sur $\mathcal{S}$. La fonction de

récompense prend en compte les bénéfices des états (1 point par population viable). Enfin, nous prenons en compte l'observabilité partielle en représentant la probabilité de détection d'un tigre en fonction de l'état et de l'action choisie à l'aide d'un ensemble de matrices d'observation O. On dénote $\Omega = \Omega_a \times \Omega_b$ l'ensemble fini des observations du système.

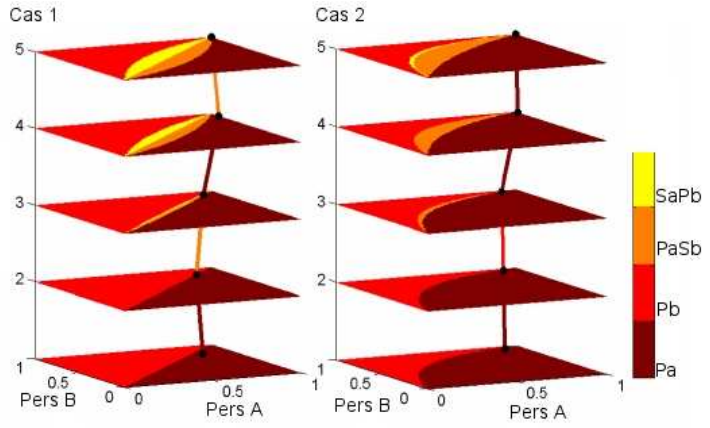


Figure 7.5. *Influence de la probabilité d'extinction sur la stratégie optimale dans deux cas de figure sur un horizon de 5 années. Pers A et Pers B représentent la croyance de persistence des populations A et B.*

L'étude des solutions comme celle présentée figure 7.5 nous permet d'énoncer des règles simples de gestion de réserves multiples. L'un des résultats importants est qu'il est plus efficace d'alterner la protection des réserves pour le tigre de Sumatra plutôt que d'essayer de protéger les deux réserves simultanément de manière moins efficace. Sur la figure 7.5, dans le cas 1, les populations bénéficient de la même qualité d'habitat et ont les mêmes probabilités d'extinction : l'alternance de protection est optimale. Dans le cas 2, la population B a une probabilité d'extinction plus forte que la population a : la stratégie optimale favorise la protection de la population a au détriment de la population b plus faible. Les lecteurs intéressés par cette étude pourront se rapporter à [MCD 08].

7.3. Les loutres et les abalones peuvent-ils co-exister ?

7.3.1. *Les abalones et les loutres, deux espèces menacées*

Les loutres et les abalones de Colombie-Britannique sont protégées [CIT 07] et disposent toutes deux d'un plan de protection et de réhabilitation. La loutre a été exterminée au début du 20ème siècle : chassée pour sa fourrure, l'espèce s'est éteinte en Colombie-Britannique en 1905. Seules quelques colonies ont survécu au nord de

l'Alaska. Dans les années 80, un plan de réintroduction a été mis en place. La population de loutres s'étend à présent sur les côtes de l'île de Vancouver et l'on compte un peu plus de 3500 individus. La loutre de Colombie-Britannique est principalement menacée par la pollution maritime et les phénomènes de marée noire qui peuvent être particulièrement dévastateurs pour une petite population.

L'abalone, aussi appelé ormeaux, est un coquillage qui vit sur les côtes de la Colombie-Britannique dans les forêts d'algues (*kelp*). L'abalone constitue le régime alimentaire préféré des loutres. Après la disparition de ces dernières, la population d'abalones a explosé. Dans les années 70, une activité de pêche prolifique s'est ainsi développée. L'abalone est un coquillage précieux, apprécié pour sa chair et la nacre de sa coquille. Devant la diminution des stocks, cette pêche génératrice de revenus importants a dans un premier temps été contrôlée par la mise en place de quotas dans les années 80, quotas qui se sont révélés insuffisants pour la régénération des stocks. La pêche d'abalones a donc été suspendue, puis interdite depuis 1993. La densité d'abalones est ainsi passée de plus de $1,2 - 2\,abalone/m^2$ à moins de $0,1 - 0,3\,abalone/m^2$. Bien que la pêche d'abalones soit interdite, l'espèce est en réalité toujours victime de braconnage et du marché noir.

La diminution de la population d'abalones et l'extermination des loutres a pour conséquence un autre désastre écologique : l'explosion de la population d'oursins dévoreurs d'algues précieuses à l'équilibre de la biodiversité de ces régions. Les forêts d'algues font place à des déserts marins. C'est tout un écosystème qui a été bouleversé par l'activité humaine. Aujourd'hui, bien que protégée, la population d'abalones n'augmente pas et reste à un niveau préoccupant. Plusieurs hypothèses biologiques ont été formulées afin d'expliquer ce phénomène. Tout d'abord le mode de reproduction de ce coquillage requiert une densité forte pour être réussi, la forte proportion d'oursins empêche l'installation de nouvelles abalones, ces deux espèces sont en compétition pour la nourriture et l'habitat. Enfin, l'activité parallèle de pêche illégale est très importante, elle est estimée à plus de 40 tonnes par an, soit l'équivalent du dernier quota de pêche autorisée en 1993. Dans ce contexte où la population de loutres est en expansion, nous soulevons la question de la co-existence de ces deux espèces à des niveaux de subsistance suffisants.

Pour répondre à cette question nous avons choisi de modéliser la dynamique de population de ces deux espèces ainsi que leurs interactions. En utilisant un simulateur, nous avons reproduit les effets des décisions de conservation sur chaque population. Nous avons utilisé deux algorithmes d'apprentissage par renforcement adapté à l'horizon fini : le QH-learning et le RH-learning.

7.3.2. *Modèles*

7.3.2.1. *Dynamique de population des abalones*

Nous avons choisi un modèle de dynamique des populations démographiques présenté dans Bardos et al ([BAR 06]). Ce modèle, aussi appelé modèle structuré de matrice de population, constitue un bon compromis entre une modélisation du comportement individuel de type systèmes multi-agent réactifs et les modèles déterministes simplifiés. Les éléments de la matrice expriment les différentes probabilités de croissance, de naissance et de mortalité pour un individu dans une phase donnée aussi appelées probabilités de transition. Ces valeurs sont utilisées lors de simulations déterministes pour lesquelles les probabilités pour les individus définissent des fractions ou taux pour les populations. C'est une approximation commune dans la communauté écologique pour laquelle les fluctuations sont ignorées. Cette approximation est vérifiée aux limites de grandes populations [MAY 73].

Plus formellement, l'état de la population à l'instant n est défini par le vecteur $\vec{x}(n) = (x_1(n), ..., x_7(n))$ de dimension 7 identifiant autant de classes d'âge. L'évolution de la population d'abalones à l'instant $n+1$ est définie par la relation $\vec{x}(n+1) = G.\vec{x}(n)$, avec G la matrice de transition :

$$\begin{pmatrix} 0 & 0 & 0 & 0 & f_5 s_5 & f_6 s_6 & f_7 s_7 \\ g_{2,1} s_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ g_{3,1} s_1 & g_{3,2} s_2 & 0 & 0 & 0 & 0 & 0 \\ g_{4,1} s_1 & g_{4,2} s_2 & g_{4,3} s_3 & g_{4,4} s_4 & 0 & 0 & 0 \\ 0 & g_{5,2} s_2 & g_{5,3} s_3 & g_{5,4} s_4 & g_{5,5} s_5 & 0 & 0 \\ 0 & 0 & g_{6,3} s_3 & g_{6,4} s_4 & g_{6,5} s_5 & g_{6,6} s_6 & 0 \\ 0 & 0 & 0 & g_{7,4} s_4 & g_{7,5} s_5 & g_{7,6} s_6 & g_{7,7} s_7 \end{pmatrix}$$

Les g_{ij} définissent la matrice de croissance g, c'est à dire la probabilité de transition d'un individu de la classe j à la classe i. Les s_j définissent la probabilité de survie pour un individu en phase j. Les f_i représentent la fécondité à la classe i multipliée par la probabilité de fertilisation et la probabilité de survie d'une larve. En d'autres termes, f_i est le nombre de post-larves produites par un individu dans la classe i à chaque pas de temps. Les f_i sont multipliés par les probabilités de survie et sont dépendant de la densité d'adultes (classes 5 à 7). En l'absence de menace, la population d'abalones se stabilise à $1 abalone/m^2$ (25 premières années de la figure 7.6).

La classe 1 correspond à l'état post-larvaire et les classes 2 à 4 sont les classes d'âge des juvéniles, enfin les classes 5 à 7 identifient des abalones adultes. Les abalones, bien que protégés, sont victimes de la pêche illégale. Il est estimé que le taux de pêche actuel diminue le taux de survie de la 7ème classe de 90% . La simulation de la pêche illégale fait descendre la densité d'abalones au m^2 à $0,31$. Nous modélisons ce phénomène par un processus stochastique en faisant varier ce taux de braconnage entre 90% (probabilité $0,75$) et 70% (probabilité $0,25$).

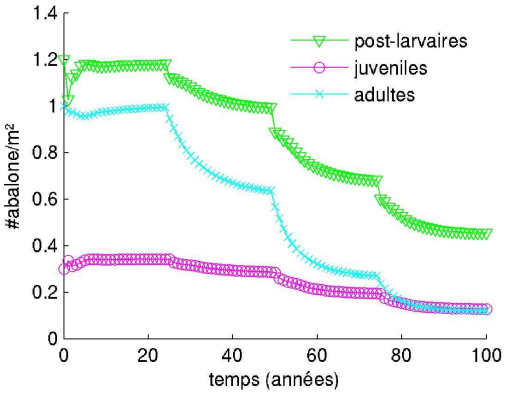


Figure 7.6. *Evolution de la population d'abalones selon différents taux de prédation (t=0 aucun t=25 faible, t=50 moyen et t=75 fort).*

7.3.2.2. *Dynamique de population des loutres*

La dynamique de population des loutres est représentée par le modèle Beverton-Holt décrit dans [GER 04]. Ce modèle inclut une relation asymptotique entre la densité et la natalité :

$$N_{t+1} = \frac{e^r K N_t}{e^r N_t - N_t + K}$$

avec K la taille maximale de la population, N_t la taille de la population à t et r le taux de croissance intrinsèque. Les paramètres sont issus de l'étude d'une population de loutres vivant dans l'état de Washington avec $K = 612$ et $r = 0,26$ [GER 04]. Bien que les prédateurs comme les requins, les orques et les aigles aient un impact sur les loutres, les marées noires sont de loin la cause majeure de mortalité. A partir des données recueillies sur le terrain, nous avons introduit dans notre modèle l'occurrence de marées noires comme un processus stochastique qui se produit en moyenne tous les 10 ans et dont l'intensité varie, pouvant réduire la population de loutres de 20% à 43%. Ce taux est relativement faible par rapport aux observations réalisées ces dernières années. La figure 7.7 illustre la dynamique de la population de loutres soumise à l'occurrence de marées noires.

7.3.2.3. *Etats*

On définit $\mathcal{S}_a$ l'ensemble fini des états de la population d'abalones. $\mathcal{S}_a$ partitionne l'ensemble des valeurs possibles de la population adulte (classes 5 à 7) en 20 états distincts. Chaque état représente un intervalle de densité $0,05$. On attribut également à chaque état un statut qualifiant la population selon son niveau d'alerte (voir tableau 7.1).

On définit $\mathcal{S}_l$ l'ensemble fini des états de la population de loutres. $\mathcal{S}_l$ partitionne l'ensemble des valeurs possibles de la population en 10 états distincts. Chaque état représente un intervalle de 10% de la capacité K. On attribue également à chaque état

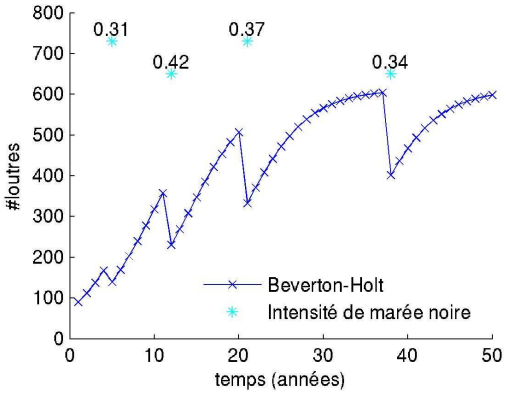


Figure 7.7. *Simulation des effets de marées noires sur une population de loutres.*

Densité d'abalones (m^{-2})		< 0.1	< 0.3	< 0.5	≥ 0.5
Status		En danger	Menacé	Concerné	Pas à risque
Quantité de loutres $(\%K)$	0	≤ 30	≤ 40	≤ 60	≥ 60
Status	Extirpé	En danger	Menacé	Concerné	Pas à risque

Tableau 7.1. *Classification des états en status selon son niveau d'alerte.*

un statut qualifiant la population selon son niveau d'alerte (voir tableau 7.1). Enfin, on définit $\mathcal{S}$ l'ensemble fini des états de notre problème $\mathcal{S} = \mathcal{S}_a \times \mathcal{S}_l$ avec $|\mathcal{S}| = 200$. L'état initial de notre problème suppose que les loutres ne sont pas encore réintroduites et qu'une activité de braconnage est présente.

7.3.2.4. *Décisions*

Nous considérons cinq actions de conservation définissant $\mathcal{A}$: ne rien faire (N), réintroduire les loutres (RI), augmenter la force anti-braconnage (AB), contrôler la population de loutres (CL) et l'action combinée contrôler la population de loutres et force anti-braconnage (ABCL).

La mise en place d'une force anti-braconnage permet de réduire l'activité de pêche illégale de 90% à 10% avec une probabilité $0,75$ ou à 30% avec une probabilité $0,25$.

La décision de contrôle de la population de loutres ne se réalise que si le statut de sa population est à "*pas à risque*". Cette action réduit la population de $0,3K$ chaque année.

7.3.2.5. *Interactions et probabilités de transition*

En l'absence de modèle mathématique décrivant les interactions entre les loutres et les abalones nous avons décidé d'étudier trois formes d'interactions inspirées de la littérature :

– La première forme d'interaction suppose que la pression prédatrice des loutres sur les abalones ne dépend que de la densité d'abalones (F1) : le taux de prédation croît avec le nombre d'abalones (figure 7.8).

– De manière symétrique, la deuxième forme suppose que la pression prédatrice des loutres sur les abalones ne dépend que de la densité de loutres (F2) : le taux de prédation croît avec le nombre de loutres.

– Enfin, la forme 3 est influencée par l'abondance des loutres et des abalones. La prédation augmente avec la densité en proie et diminue jusqu'à satiété lorsque la densité en proie est forte (F3).

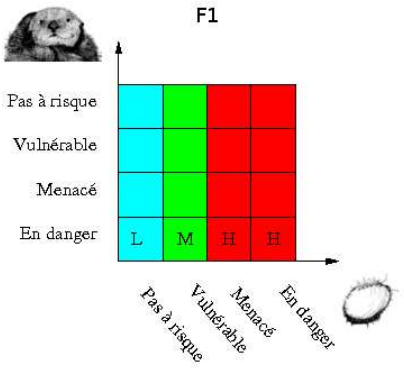


Figure 7.8. *Représentation de la fonction d'interaction F1 entre les loutres et les abalones (L pour faible prédation, M pour prédation moyenne, H pour prédation forte).*

Le taux de survie des classes 3 à 7 est réduit de 5% (L) pour une prédation faible, de 15% (M) pour une prédation moyenne et 25% (H) pour prédation forte (voir figure 7.6 et 7.8) .

7.3.2.6. *Objectif multicritère et fonction de récompense*

Pour déterminer une politique de gestion optimale, nous devons définir notre objectif. Ici, nous avons distingué deux critères : maximiser l'occurrence d'avoir les deux espèces à "*pas à risque*" ou "*vulnérable*" simultanément (R1), ou bien de manière indépendante (R2). La figure 7.9 illustre les deux fonctions de récompenses que nous avons considérées.

7.3.3. *Méthodes*

Nous avons choisi d'utiliser deux algorithmes d'apprentissage par renforcement pour résoudre ce problème à horizon fini : le QH-learning et le RH-learning adapté par Garcia et al. [GAR 98]. Il n'existe pas de preuve de convergence théorique de

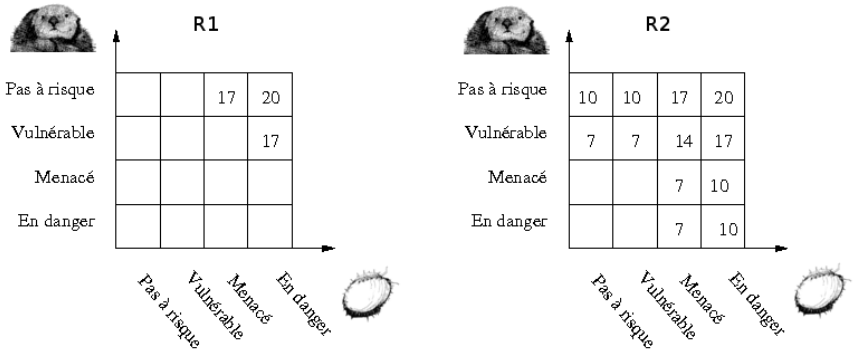


Figure 7.9. *R1 identifie une récompense jointe et R2 une récompense individuelle.*

Niveau de braconnage	0%	10%	90%
vulnérable → menacé	27.2%	25.9%	19.7%
menacé → en danger	13.4%	11.1%	0.6%
pas à risque → vulnérable	7.4%	4.3%	0.1%

Tableau 7.2. *Niveau de prédation suffisant pour que la population d'abalone change de statut dans différents cas de braconnage.*

ces algorithmes à horizon fini. Toutefois les résultats expérimentaux montrent leur efficacité à produire de bonnes stratégies.

7.3.4. *Résultats*

Notre objectif est de déterminer les conditions de gestion pour lesquelles les deux populations peuvent co-exister à des niveaux non menacés sur un horizon de 50 années, une décision étant prise tous les 5 ans. Nous avons laissé les deux algorithmes apprendre sur 500 000 simulations. Nous avons procédé par étape tout d'abord sans considérer l'action controversée du contrôle de la population de loutres. Nos résultats montrent que sans contrôle de la population de loutres il n'est pas possible d'atteindre l'objectif souhaité.

7.3.4.1. *Conditions à remplir*

Pour maintenir les deux populations à des niveaux *pas à risque*, l'impact des loutres sur les abalones sous différentes conditions de braconnage doit être relativement faible comme en témoigne le tableau 7.2. Dans les conditions idéales d'absence de braconnage, le taux de prédation des loutres doit être inférieur à $7,4\%$ pour ne pas menacer les abalones. Ce taux de prédation descend à $4,3\%$ en cas de braconnage de faible intensité et est inférieur à $0,1\%$ en cas de braconnage intensif.

7.3.4.2. *Scénario 1 : réintroduction de loutres et force anti-braconnage*

L'état initial considéré dans notre problème identifie l'absence de loutre et une activité de braconnage.

Nous avons dans un premier temps étudié les décisions utilisées actuellement : la réintroduction de loutres et la mise en place d'une force anti-braconnage.

Les stratégies optimales pour l'interaction de type F1 et les récompenses R1 et R2 réintroduisent les loutres dès le premier pas de temps. La force anti-braconnage est par la suite optimale. Le tableau 7.3, colonnes 2 et 3, lignes QL1 et RL1, représente les performances moyennes observées pour ce scénario. La population de loutres se stabilise autour des états *pas à risque* et *vulnérable* (suivant l'occurrence de marées noires) tandis que la population d'abalones oscille entre *menacé* et *vulnérable*. Cette dernière oscillation s'explique par la définition du niveau de prédation de la fonction d'interaction F1 de faible à moyen (figure 7.8).

Les stratégies optimales pour l'interaction de type F2 et les récompenses R1 (figure 7.10) et R2 s'efforcent dans un premier temps d'augmenter la population d'abalones jusqu'au niveau « *pas à risque* » en préconisant la force anti-braconnage. Puis les loutres sont réintroduites. La fonction d'interaction F2 rend encore plus difficile la co-existence des deux espèces à des niveaux *vulnérable* ou *pas à risque*. La population d'abalones suit une évolution inverse de la population de loutres. Les performances observées tableau 7.3 colonnes 4 et 5 représentent les faibles récompenses accumulées pour l'interaction F2.

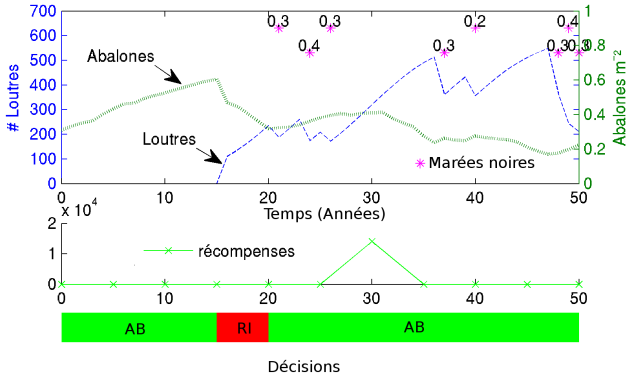


Figure 7.10. *Une simulation de la meilleure stratégie pour le cas du scénario 1 (fonction d'interaction F2 et récompenses R1).*

Les stratégies optimales pour l'interaction de type F3 et les récompenses R1 et R2 réintroduisent les loutres dès le premier pas de temps. La force anti-braconnage est par la suite optimale. La densité des abalones décroit avec l'augmentation du nombre de loutres jusqu'à atteindre un seuil où la pression est faible (*menacé* ou *en danger*).

Les performances observées tableau 7.3 colonnes 6 et 7 reflètent les récompenses accumulées pour l'interaction F3 et ne nous permettent pas de conclure à la co-existence des deux espèces à des niveaux non menacés.

7.3.4.3. *Scénario 2 : contrôle des loutres*

A la lumière de ces résultats nous avons décidé d'introduire l'action de contrôle de la population de loutres. Cette action est controversée car la loutre est protégée par la loi, il est donc illégal de procéder à cette action à ce jour.

Cette action ne permet pas d'améliorer les performances (lignes QL2, RL2 tableau 7.3).

7.3.4.4. *Scénario 3 : actions combinées contrôle des loutres et force anti-braconnage*

Nous avons donc décidé de combiner cette action de contrôle des loutres avec l'action anti-braconnage. Cette option fait l'hypothèse que nous disposons de suffisamment d'argent pour effectuer ces deux actions parallèlement et en garantissant la même efficacité. Dans ce troisième scénario, les performances des stratégies optimales sont aussi bonnes pour la fonction d'interaction F1 où le taux de prédation des abalones ne dépend pas de la quantité de loutres. Les performances sont meilleures pour les fonctions d'interaction F2 et F3. Toutefois cela reste insuffisant : les populations ne se stabilisent pas à des niveaux *pas à risque* (figure 7.11).

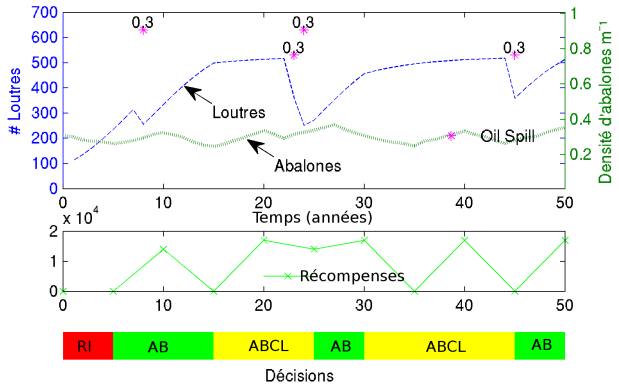


Figure 7.11. *Une simulation de la meilleure stratégie pour le cas du scénario 3 (fonction d'interaction F3 et récompenses R2).*

7.3.5. *Discussion*

Nous n'avons pas réussi à trouver une stratégie qui permette de stabiliser les deux espèces à des niveaux *pas à risque*. Nous pouvons évoquer quelques éléments explicatifs.

	F1 R1	F1 R2	F2 R1	F2 R2	F3 R1	F3 R2
QL1	66.06	111.85	19.90	98.96	53.63	108.28
RL1	66.11	111.44	20.50	98.55	53.94	108.33
QL2	66.29	111.36	19.47	99.11	53.07	108.72
RL2	66.10	111.78	19.48	99.12	54.92	108.65
QL3	66.17	111.56	25.39	100.06	84.67	119.95
RL3	66.05	111.44	24.73	99.90	84.53	119.64

Tableau 7.3. *Performances comparées pour chaque scénario et algorithme exprimé sous la forme de récompenses cumulées sur un total de 160 points.*

Premièrement, nos modèles et nos hypothèses peuvent ne pas être réalistes. En l'absence d'information publiée sur les interactions entre les loutres et les abalones, nous avons supposé 3 formes d'interactions possibles et complémentaires. Il est possible que la vraie fonction d'interaction soit un mélange de ces 3 fonctions. La littérature sur le sujet étant pauvre, une étude approfondie de la sensibilité des paramètres d'interaction nous permettra de répondre à cette question. Nous procédons également à l'amélioration du modèle de population des abalones. En effet les abalones du Nord ont des caractéristiques de croissance différentes du modèle de population générale présenté dans [BAR 06].

Une seconde possibilité qui pourrait expliquer pourquoi nous n'avons pas été en mesure de montrer la co-existence des deux espèces à des niveaux *pas à risque* est une définition non réaliste des niveaux d'alerte des abalones en présence des loutres. Si cela est le cas, alors les équipes de rétablissement sont susceptibles d'investir des ressources significatives afin d'atteindre un objectif non réalisable. Car, même avec notre définition des niveaux d'alerte d'abalones optimistes (*pas à risque* 0.5 au lieu de 1), la co-existence des deux espèces à ces niveaux n'a pas été montrée.

Dans la littérature, Watson [WAT 00] argumente que le rétablissement des loutres et de la pêche d'abalones sont mutuellement exclusives, corroborant ainsi nos conclusions. Nous avons toutefois trouvé qu'il était possible d'obtenir des configurations *pas à risque* pour les loutres et *vulnérable* pour les abalones en implémentant les actions combinées de contrôle des loutres et de force anti-braconnage. Le rétablissement d'une pêche traditionnelle par les populations indigènes est une possibilité pour le contrôle de la population de loutres.

Ce travail met en valeur l'intérêt des modèles et méthodes des MDP pour l'aide à la décision de la conservation de deux espèces menacées en interaction. De nombreuses perspectives sont envisagées et incluent une généralisation à une étude spatiale, l'incorporation de fonctions de coûts ainsi que l'enrichissement du choix des décisions possibles.

7.4. Conclusion

Dans ce chapitre nous avons illustré comment les modèles décisionnels de Markov et leurs méthodes d'optimisation pouvaient s'appliquer dans le domaine de la conservation de la biologie. De manière générale, nous identifions un besoin grandissant de développement de nouvelles méthodes prennant en compte la spatialité des problèmes, l'observabilité partielle et l'intérêt des approches multi-critères. De tels algorithmes permettraient de résoudre des problèmes d'allocation de ressources et de protection de la biodiversité comme présenté dans [WIL 06] qui cherche à identifier les lieux prioritaires parmi l'ensemble des "hot spot" de biodiversité dans le monde.

TROISIÈME PARTIE

Extensions

Chapitre 8

DEC-MDP/POMDP

8.1. Introduction générale

Les MDP et les POMDP sont deux modèles mathématiques rigoureux qui ont été utilisés avec succès dans la formalisation des processus décisionnels séquentiels sous incertitude. Ils ont été utilisés pour le contrôle d'un agent évoluant dans des environnements partiellement ou complètement observables et qui cherche à maximiser sa performance en interaction avec cet environnement (et probablement avec d'autres agents) en se fondant sur ses observations. Ce succès amène naturellement les chercheurs à utiliser ce modèle mathématique dans des systèmes complexes composés de plusieurs agents en interaction.

EXEMPLE.– Revenons à l'entretien de notre voiture (voir tome 1, section 1.1). On peut supposer que plusieurs garagistes, qui ne se coordonnent pas toujours, seront sollicités pour cet entretien. On peut même imaginer une situation futuriste où les garages seront équipés d'une flotille de robots spécialisés : un pour les freins, l'autre pour l'huile, un autre pour la rouille, *etc*. Ainsi, plusieurs agents peuvent décider d'agir sur le même objet, parfois avec des informations communes, parfois avec des informations parcellaires. Comment faire pour que ces agents coopèrent pour que l'action globale résultant de chacune des actions individuelles soit optimale ? Plusieurs extensions des MDP permettent d'apporter des réponses à ce problème.

Dans ce chapitre, nous présentons, comme pour les MDP, les POMDP mono-agent, les divers formalismes d'extension des MDP et POMDP aux systèmes multi-agents. Ces extensions sont diverses et se fondent sur différentes hypothèses qu'on peut classer en : i) chaque agent a une connaissance complète de l'état du monde, ii) chaque agent

---

Chapitre rédigé par Aurélie BEYNIER et François CHARPILLET et Daniel SZER et Abdel-Illah MOUADDIB.

a une connaissance partielle (en général propre) de l'état du monde, iii) les agents peuvent communiquer, iv) les agents ne peuvent pas (ou partiellement) communiquer, ...

Ces différentes hypothèses ont donné lieu à différents formalismes que nous passons en revue pour les plus notables d'entre eux, comme MMDP, Dec-MDP, Dec-POMDP, Dec-MDP-Com, MTDP, COM-MTDP, R-MTDP, E-MTDP, EMT, I-POMDP, POSG, POIPSG, ND-POMDP, TI-Dec-MDP, OC-Dec-MDP, EOC-Dec-MDP. Ce chapitre montrera aussi la difficulté de construire des solutions optimales avec ces formalismes à cause de la très haute complexité et qui construit un obstacle à leur utilisation dans des problèmes réels complexes. Ce chapitre discutera ce problème et montrera que, lorsqu'on exploite une certaine structure du problème comme la localité des interactions, la décomposabilité des récompenses et certaines formes d'indépendances entre les agents, des algorithmes approchés sont possibles et, pour certains, convergent vers des optima locaux et atteignent certaines formes d'équilibre.

8.2. Observabilité

L'observabilité d'un environnement caractérise l'ensemble des informations qui sont accessibles à un agent. Les modèles utilisés afin de formaliser les problèmes de décision multi-agent sous incertitude varient selon le degré d'observabilité des agents. Nous commençons donc, dans cette section, par définir les différents types d'observabilité. Nous montrerons dans la suite du chapitre que le degré d'observabilité influence également la complexité de la prise de décision.

Comme cela a été indiqué au chapitre 3, nous parlerons d'**observabilité partielle** quand toutes les informations nécessaires à la prise d'une décision par un agent ou un groupe d'agents ne sont pas accessibles instantanément. Les agents seront alors contraints de faire face à ce manque d'information et devront décider au mieux comment agir en fonction des informations accessibles et des éventuelles connaissances complémentaires dont ils disposent.

Dans le cadre multi-agents, suivant les informations auxquelles les agents ont accès, nous pouvons distinguer différents types d'observabilité. Nous distinguerons tout d'abord l'observabilité de l'état du système et l'observabilité de l'état de l'agent. Le terme « état global » désignera l'état du système multi-agents (agents + environnement). L'état d'un agent sera défini selon son degré de délibération —la complexité de ses raisonnements. Il pourra correspondre aux perceptions courantes de l'agent ou bien à une représentation interne de ses connaissances.

L'état d'un agent peut être partiellement ou complètement observable. Dans le premier cas, on parle d'état **localement partiellement observable**. L'agent n'a alors pas accès à toutes les informations nécessaires pour savoir dans quel état il se trouve. Dans le cas contraire, l'état est **localement totalement observable** : l'agent peut, à partir de ses observations, déduire avec certitude son état.

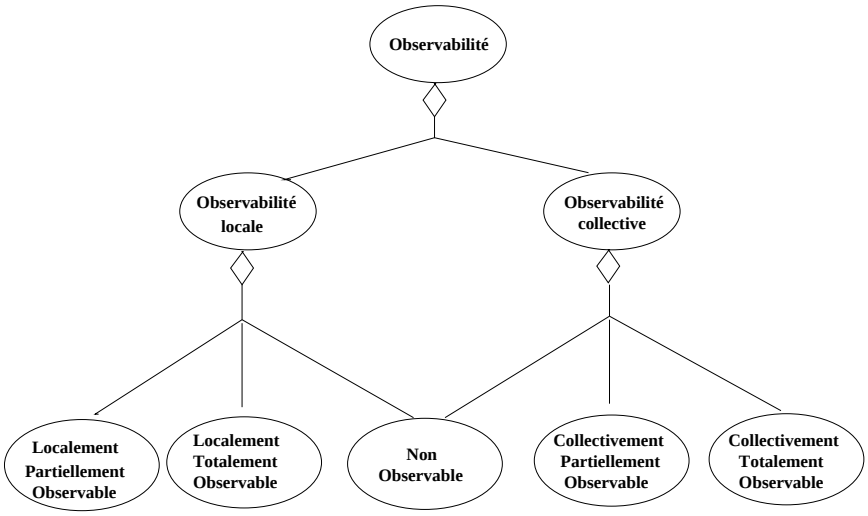


Figure 8.1. *Différents types d'observations*

En ce qui concerne l'état global du système, on distingue **observabilité collective** et **observabilité individuelle**. L'observabilité collective résulte de l'agrégation des observations de tous les agents. L'état global du système peut être **collectivement partiellement observable** ou bien **collectivement totalement observable**. Dans le cas de l'observabilité collective partielle, il n'est pas possible de déduire l'état du système même si on a accès à toutes les observations de tous les agents. Certaines propriétés de l'état global ne sont donc observées par aucun des agents. Dans le cas contraire, on peut déduire l'état global du système à partir des observations locales et l'état global est dit **collectivement totalement observable**.

Par ailleurs, l'état global du système est dit **individuellement observable** si chaque agent connaît l'état global du système à partir de ses observations. Un état global individuellement observable est par conséquent collectivement totalement observable. Dans le cadre mono-agent, l'observabilité individuelle de l'état du système différencie les MDP des POMDP (cf. chapitre 3).

Enfin, lorsque les agents n'ont aucune observabilité, on parle de **non-observabilité** (locale et collective).

Remarquons que l'observabilité ne concerne pas seulement l'état du système et des autres agents mais également leur comportement. En effet, chaque agent peut, dans certains cas, connaître les stratégies des autres agents. Ainsi, il est possible de déduire comment s'adapter au mieux à leur comportement.

8.3. Processus décisionnels de Markov multi-agents

Afin d'adapter le formalisme des MDP aux systèmes multi-agent coopératifs, Boutilier [BOU 96a, BOU 99c, BOU 99a] a défini les processus décisionnels de Markov multi-agent ou MMDP (*Multiagent Markov Decision Processes*). Ces derniers permettent de formaliser des problèmes de décision séquentielle dans des systèmes multi-agents coopératifs. Ce formalisme est très proche de celui des jeux de Markov [SHA 53] présenté au chapitre 4. Cependant, les MMDP modélisent uniquement des systèmes coopératifs. Il s'agit d'un jeu de Markov dans lequel la fonction de récompense est partagée par tous les joueurs, alors que, dans le cadre général des jeux de Markov, il y a une fonction de récompense propre à chaque agent.

8.3.1. *Formalisme*

Un MMDP est défini par un tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ tout comme les processus décisionnels de Markov classiques. Cependant, une action élémentaire, appelée « action jointe » est décrite par l'ensemble des actions individuelles des agents. En plus du tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, on ajoute une variable α qui correspond au nombre d'agents du système.

Définition 15 ***Processus décisionnels de Markov multi-agents***

Un MMDP *est défini par un tuple $\langle n, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ tel que :*

– n est le nombre d'agents $\mathcal{A}g_i$ du système, $i \in \{1, ..., n\}$.

– $\mathcal{S}$ correspond à l'ensemble des états s du système.

– $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ définit l'ensemble des actions jointes des agents, $\mathcal{A}_i$ est l'ensemble des actions locales de l'agent $\mathcal{A}g_i$.

– $\mathcal{T}$ est une fonction de transition. Elle donne la probabilité $\mathcal{T}(s, a, s')$ que le système passe dans un état s' quand les agents exécutent l'action jointe $a \in \mathcal{A}$ à partir de l'état s.

– $\mathcal{R}$ définit la fonction de récompense. $\mathcal{R}(s, a, s')$ est la récompense obtenue par le système lorsqu'il passe d'un état s à un état s' en exécutant l'action a.

Un MMDP peut être vu comme un MDP ayant un grand espace d'états et d'actions. L'ensemble des agents est alors considéré comme un seul agent dont le but est de calculer une politique optimale pour le MDP joint (cf. chapitre 1). Un MMDP peut également être considéré comme un jeu stochastique à α joueurs dans lequel la fonction de récompense est la même pour tous les joueurs. Le formalisme des MMDP correspond donc à une généralisation des MDP au cas multi-agent et à une spécialisation des jeux stochastiques à α joueurs.

Résoudre un MMDP consiste à calculer une politique jointe $\pi = \langle \pi_1, \cdots, \pi_n \rangle$ où π_i correspond à la politique locale de l'agent $\mathcal{A}g_i$. Elle définit une fonction $\pi_i : \mathcal{S} \rightarrow \mathcal{A}_i$ qui fait correspondre à tout état du système une action a_i de l'agent $\mathcal{A}g_i$. Une telle

politique jointe peut être calculée par un algorithme classique comme l'algorithme d'itération sur les valeurs (cf. chapitre 1).

La politique d'un MMDP est une politique centralisée. Il est nécessaire pour l'exécuter que chaque agent ait accès à l'état global du système. Dans les systèmes multi-agents cette hypothèse est rarement vérifiée. Afin d'exécuter la politique du MMDP, il est alors nécessaire que le contrôle soit centralisé ou bien que les agents puissent communiquer.

8.3.2. *Contrôle centralisé*

Les systèmes multi-agents à contrôle centralisé sont conçus de telle façon qu'un contrôleur central est chargé de calculer la politique optimale jointe et de prendre les décisions pour tous les agents. Dans de tels systèmes, la politique d'un MMDP peut être facilement exécutée. L'entité centrale connaît l'état global du système et peut dicter à chaque agent quelle action réaliser.

Un tel type de contrôle n'est généralement pas envisageable dans des applications pratiques. En effet, les agents sont souvent répartis dans l'espace et aucun n'a assez d'information pour jouer le rôle d'entité centrale.

8.3.3. *Contrôle décentralisé*

Appliquer la politique d'un MMDP de manière décentralisée est cependant possible si les agents peuvent communiquer à volonté et gratuitement, c'est-à-dire si la communication n'est pas limitée par des contraintes physiques telles que la taille de la bande passante et si elle n'a aucune influence sur l'utilité des agents. Ces derniers peuvent alors échanger leurs informations sur leurs états afin de déterminer l'état global du système. Il est malgré tout nécessaire que l'état du système soit collectivement totalement observable. Sinon, les agents ne pourront pas déduire l'état du système même s'ils ont échangé toutes leurs informations.

Dans les systèmes multi-agents où le contrôle est décentralisé, le calcul de la politique peut être réalisé de manière centralisée ou décentralisée. Dans le premier cas, un contrôleur central est chargé de calculer la politique optimale jointe $\pi = \langle \pi_1, \cdots, \pi_n \rangle$. Les politiques locales π_i constituant cette politique optimale jointe sont ensuite envoyées aux agents chargés de les exécuter.

Un autre modèle de conception, complètement décentralisé, peut être utilisé. Chaque agent calcule alors de manière individuelle la politique optimale jointe. Si les agents suivent le même raisonnement pour effectuer ce calcul, ils obtiendront tous la même politique optimale. Cependant, s'il existe plusieurs politiques optimales, il est nécessaire que les agents coordonnent le choix de la politique optimale à suivre, faute de quoi des problèmes de coordination risquent de survenir [CLA 98]. En effet, si les agents sélectionnent différentes politiques et font ainsi des choix non-coordonnés, le comportement global peut ne pas être optimal.

La **coordination** est définie par Malone [MAL 88] comme « l'ensemble des activités supplémentaires qu'il est nécessaire d'accomplir dans un environnement multi-agents et qu'un seul agent poursuivant les même buts n'accomplirait pas ». Ces tâches de coordination permettent d'améliorer le comportement des agents. Ainsi, les agents peuvent faire face à des situations où leurs actions sont susceptibles d'interférer entre elles. De même, la coordination permet de garantir une certaine qualité de résultat même si un agent manque de compétences, de ressources ou d'information. Selon Ferber [FER 95] ; « *la coordination des actions, dans le cadre de la coopération, peut donc être définie comme l'articulation des actions individuelles accomplies par chacun des agents de manière à ce que l'ensemble aboutisse à un tout cohérent et performant. (...) l'action du groupe est améliorée soit par une augmentation des performances, soit par une diminution des conflits. La coordination des actions est donc l'une des principales méthodes pour assurer la coopération entre agents autonomes.* ».

En décrivant les MMDP, Boutilier a proposé un premier formalisme intégrant systèmes multi-agents coopératifs et processus décisionnels de Markov. L'utilisation des processus décisionnels de Markov multi-agents (MMDP) suppose l'existence d'un contrôleur central ayant une vue globale du système ou bien nécessite la possibilité de communiquer à volonté et gratuitement. Cependant, dans de nombreux systèmes multi-agents, chaque agent n'a qu'une vue locale de l'état global du système sur laquelle il base ses décisions et la communication possède un coût. Les MMDP peuvent alors difficilement être utilisés.

8.4. Contrôle décentralisé et processus décisionnels de Markov

Les processus décisionnels de Markov décentralisés partiellement observables (DEC-POMDP) et les processus décisionnels de Markov décentralisés (DEC-MDP) constituent des extensions des POMDP et des MDP pour des domaines où le contrôle est décentralisé.

8.4.1. *Les processus décisionnels de Markov décentralisés*

Dans le cas d'une prise de décision décentralisée, l'état de chaque agent peut ne pas être directement observable par l'agent. Celui-ci reçoit alors des observations qui peuvent s'avérer insuffisantes afin de déduire précisément son état. On parle d'état localement partiellement observable. L'agent doit alors fonder sa décision sur ses observations et non sur son état. De ce fait, les DEC-MDP et les DEC-POMDP définissent un ensemble d'observations Ω constitué des observations locales Ω_i des agents. Une fonction d'observation $\mathcal{O}$ est également définie, qui permet d'obtenir la probabilité qu'un agent $\mathcal{A}g_i$ observe o_i étant donnés un état de départ s, une action jointe a et un état d'arrivée s'.

Définition 16 *Un* DEC-POMDP *est défini par un tuple* $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ *tel que :*

– $\mathcal{S}$ définit l'état global du système.

– $\mathcal{A} = \langle A_1, \cdots, A_n \rangle$ est l'ensemble des actions jointes et A_i définit l'ensemble des actions a_i de l'agent $\mathcal{A}g_i$.

– $\mathcal{T} = S \times A \times S \rightarrow \mathcal{R}$ définit la fonction de transition. $T(s, a, s')$ correspond à la probabilité que le système passe d'un état s à un état s' lorsque l'action jointe a est exécutée.

– $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$ est l'ensemble des observations des agents et Ω_i est l'ensemble des observations de l'agent $\mathcal{A}g_i$.

– $\mathcal{O} = S \times A \times S \times \Omega \rightarrow \mathcal{R}$ définit la fonction d'observation. $\mathcal{O}(s, a, s', o = \langle o_1, \cdots, o_n \rangle)$ correspond à la probabilité que chaque agent $\mathcal{A}g_i$ observe o_i lorsque les agents exécutent l'action jointe a à partir de l'état s et que le système arrive dans l'état s'.

– $\mathcal{R}$ définit la fonction de récompense. $R(\langle s_1, \cdots, s_n \rangle, \langle a_1, \cdots, a_n \rangle, \langle s'_1, \cdots, s'_n \rangle)$ est la récompense obtenue par le système lorsque les agents exécutent l'action jointe $\langle a_1, \cdots, a_n \rangle$ à partir de l'état $\langle s_1, \cdots, s_n \rangle$ et arrivent dans l'état $\langle s'_1, \cdots, s'_n \rangle$.

La fonction de transition ainsi que la fonction de récompense dépendent de l'action jointe exécutée par tous les agents. Comme les POMDP, les DEC-POMDP sont définis sur un horizon T correspondant au nombre d'étapes de décision. Celui-ci peut être fini ou infini. A chaque exécution d'une action jointe, la fonction $\mathcal{R}$ détermine la récompense octroyée au système. Le but est de trouver le comportement (politique) de chaque agent tel que ces derniers maximisent globalement leur mesure de performance. Notons qu'un DEC-POMDP à un seul agent est équivalent à un POMDP.

Si l'état du système est collectivement totalement observable —on peut le déduire des observations locales des agents—, le DEC-POMDP devient un DEC-MDP. Cette propriété peut se traduire plus formellement ainsi : si $\mathcal{O}(s, a, s', o = \langle o_1, \cdots, o_n \rangle) > 0$, alors $P(s' | \langle o_1, \cdots, o_n \rangle) = 1$. Remarquons que cette propriété n'implique pas nécessairement que chaque agent observe totalement son état.

Dans le cas où les états des agents sont localement totalement observables, l'ensemble des observations Ω et la fonction d'observation $\mathcal{O}$ peuvent être omis afin d'éviter les redondances d'information. Le DEC-MDP est alors défini par l'ensemble $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$. Les relations entre DEC-POMDP, DEC-MDP, POMDP et MDP peuvent être résumées par le diagramme de la figure 8.2. Dans le cadre du contrôle centralisé, observabilité locale et observabilité collective sont équivalentes.

8.4.2. *Multiagent Team Decision Problem*

Pynadath et Tambe [PYN 02] ont décrit un formalisme similaire aux DEC-POMDP : le *Multiagent Team Decision Problem* (MTDP). Tout comme les DEC-POMDP, les MTDP permettent de formaliser des problèmes de contrôle décentralisé sous incertitude. Un MTDP reprend les mêmes composantes qu'un DEC-POMDP. Afin de représenter plus facilement les problèmes de contrôle dans les systèmes multi-agents, un

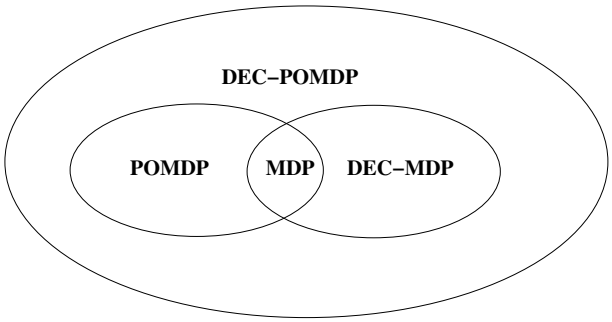


Figure 8.2. *Relations entre les différents types de processus décisionnels de Markov*

ensemble d'états de croyance est ajouté à la définition des DEC-POMDP. L'état de croyance b_i^t d'un agent $\mathcal{A}g_i$ décrit son état mental à l'instant t. Il est ainsi possible de différencier croyances et observations.

Définition 17 *Un* MTDP *est défini par un tuple* $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, B_i, \Omega, \mathcal{O}, \mathcal{R} \rangle$ *tel que :*

– $\mathcal{S}$ *définit l'état global du système.*

– $\mathcal{A} = \langle A_1, \cdots, A_n \rangle$ *est l'ensemble des actions jointes et* A_i *définit l'ensemble des actions* a_i *de l'agent* $\mathcal{A}g_i$*.*

– $\mathcal{T} = S \times A \times S \rightarrow \mathcal{R}$ *définit la fonction de transition.* $\mathcal{T}(s, a, s')$ *correspond à la probabilité que le système passe d'un état* s *à un état* s' *lorsque l'action jointe* a *est exécutée.*

– B_i *désigne l'ensemble des états de croyance de l'agent* $\mathcal{A}g_i$*. Un agent* $\mathcal{A}g_i$ *construit son état de croyance* b_i^t *à l'instant* t *à partir de ses observations jusqu'à l'instant* t*.*

– $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$ *est l'ensemble des observations des agents tel que* Ω_i *est l'ensemble des observations de l'agent* $\mathcal{A}g_i$*.*

– $\mathcal{O} = S \times A \times S \times \Omega \rightarrow \mathcal{R}$ *définit la fonction d'observation.* $\mathcal{O}(s, a, s', o = \langle o_1, \cdots, o_n \rangle)$ *correspond à la probabilité que chaque agent* $\mathcal{A}g_i$ *observe* o_i *lorsque les agents exécutent l'action jointe* a *à partir de l'état* s *et que le système arrive dans l'état* s'*.*

– $\mathcal{R}$ *définit la fonction de récompense.* $R(\langle s_1, \cdots, s_n \rangle, \langle a_1, \cdots, a_n \rangle, \langle s'_1, \cdots, s'_n \rangle)$ *est la récompense obtenue par le système lorsque les agents exécutent l'action* $\langle a_1, \cdots, a_n \rangle$ *à partir de l'état* $\langle s_1, \cdots, s_n \rangle$ *et arrivent dans l'état* $\langle s'_1, \cdots, s'_n \rangle$*.*

Seuken et Zilberstein [SEU 05] ont démontré que les modèles des DEC-POMDP et des MTDP étaient équivalents lorsque chaque agent a accès à toutes les informations qu'il a perçues par le passé. Intuitivement, les états de croyance d'un MTDP peuvent

être définis par l'historique des observations. Il n'existe donc pas de perte d'informations entre ces deux modèles et les politiques locales des agents sont définies de façon similaire.

Les relations entre les MTDP, DEC-POMDP, DEC-MDP, POMDP et MDP sont résumées par le tableau 8.1.

Type de contrôle	Centralisé		Décentralisé	
Etat global collectivement totalement observable	Oui	Non	Oui	Non
Formalisme	MDP	POMDP	DEC-MDP	DEC-POMDP
	MMDP		MTDP	

Tableau 8.1. *Relations entre les* DEC-POMDP*,* DEC-MDP*,* POMDP *et* MDP

8.4.2.1. *Complexité*

Résoudre de manière optimale un DEC-POMDP (un MTDP, ou un DEC-MDP) consiste à trouver une politique optimale jointe $\pi = \langle \pi_1, \cdots, \pi_n \rangle$ composée d'un ensemble de politiques locales π_i. Une politique optimale jointe est une politique qui maximise l'utilité espérée du système. Elle fournit aux agents un comportement optimal, c'est-à-dire que, à chaque étape de décision, chaque agent choisit l'action qui maximise le gain espéré du système. Dans le cadre des DEC-POMDP, la politique locale d'un agent $\mathcal{A}g_i$ associe à chaque historique d'observations $\langle o_{i,1}, \cdots o_{i,t} \rangle$ de l'agent une action a_i. En revanche, dans le cadre des MTDP, la politique locale d'un agent $\mathcal{A}g_i$ correspond à une fonction $\pi_i : B_i \rightarrow \mathcal{A}_i$ qui fait correspondre une action à chaque état de croyance de l'agent $\mathcal{A}g_i$.

En raison du manque d'observabilité de l'état global du système, la résolution optimale de problèmes de contrôle décentralisé est beaucoup plus difficile que dans le cadre du contrôle centralisé. Les performances du système dépendent de son état global et de l'action jointe exécutée. Cependant, lorsque l'état du système est collectivement totalement ou partiellement observable, les agents n'ont pas accès individuellement à l'état global du système et doivent baser leurs décisions sur leurs observations locales. Bien que la résolution optimale des POMDP soit un problème PSPACE-complet[1] [PAP 87], la résolution des DEC-POMDP est bien plus complexe. Bernstein et al. [BER 02] ont démontré que résoudre de façon optimale un DEC-POMDP à 2 agents ou plus est un problème NEXP[2]. Il en est de même pour les DEC-MDP : la résolution optimale d'un MDP est un problème P-complet alors que celle d'un DEC-MDP est NEXP pour 3 agents ou plus.

1. Un problème PSPACE-complet peut être résolu par un algorithme déterministe et polynomial en espace (par rapport à la taille de l'instance du problème).

2. Un problème de complexité NEXP est un problème pouvant être résolu par un algorithme non-déterministe exponentiel. La vérification d'une solution prend donc un temps exponentiel.

Si l'état du système est individuellement partiellement observable, c'est-à-dire si chaque agent connaît l'état global du système, alors le problème peut se réduire à un MDP dont la résolution est P-complète. Enfin, si l'état global du système n'est pas observable, le problème se réduit à un MDP non observable (NOMDP) connu pour être NP-complet. Le tableau 8.2 résume l'influence de l'observabilité de l'état global du système sur la complexité du problème.

Observabilité de l' état du système	Individuellement Observable	Collectivement Observable	Collectivement Partiellement Observable	Non Observable
Complexité	P-complet	NEXP-complet	NEXP-complet	NP-complet
Formalisme	MMDP, DEC-MDP	DEC-MDP	DEC-POMDP	
	MTDP			

Tableau 8.2. *Observabilité et Complexité en temps*

8.4.3. *Gestion de la communication dans les* DEC-POMDP

Dans certains systèmes multi-agents, il est possible que les agents communiquent entre eux lors de l'exécution des tâches. Cette communication peut alors leur permettre d'échanger des informations sur leurs observations, leur état, leurs actions, etc. Ainsi, les agents augmentent leurs connaissances sur les autres agents et sur l'état global du système, et peuvent se coordonner plus facilement.

Deux types de communication peuvent être identifiés : la communication directe et la communication indirecte. La communication directe consiste à envoyer des messages directement aux autres agents. La communication indirecte est généralement réalisée par modification de l'environnement ou par manipulation de connaissances communes. Les agents peuvent laisser des traces dans l'environnement afin d'indiquer aux agents quelle action a été exécutée, quel est leur état, etc. Lorsque les agents ont tous accès à une base de données commune la communication peut s'effectuer par modification des connaissances stockées dans la base. La perception de l'environnement constitue alors un mécanisme permettant de mettre en place un tel type de communication. Cependant, dans le cas de la communication indirecte, il n'est pas certain que tous les agents observeront ces modifications. Par conséquent, l'information peut ne pas être reçue.

Nous nous intéresserons par la suite à l'influence de la communication sur l'observabilité des agents. Nous ne traiterons alors que de la communication directe. En effet, cette dernière constitue le seul type de communication qui permette d'atteindre l'observabilité totale lorsqu'il n'existe pas de caractéristiques incontrôlables communément observables par tous les agents.

Toute communication a généralement un coût dû aux ressources qu'elle consomme (énergie, bande passante, ...) ou bien dû au risque de révéler des informations à d'éventuels agents compétitifs. Lorsqu'un agent a la possibilité de communiquer, il doit tenir compte de l'utilité des informations apportées par la communication et du coût de cette dernière. La décision de communiquer ou non résulte d'un compromis entre cette utilité et ce coût. Nous désignerons par communication gratuite une communication directe instantanée dont le coût est nul pour les agents.

Plusieurs formalismes se sont proposés d'étendre les DEC-POMDP afin de permettre la modélisation de la communication entre les agents durant la phase d'exécution des tâches[3]. A chaque étape de décision, un agent peut décider de communiquer ou non. Ensuite, il détermine quelle action réaliser. Comme le montre la figure 8.3, chaque étape de décision est donc décomposée en deux phases : la phase de communication et la phase d'exécution d'une action standard. Le terme « action standard » désigne toute action ne consistant pas à communiquer des informations avec les autres agents.

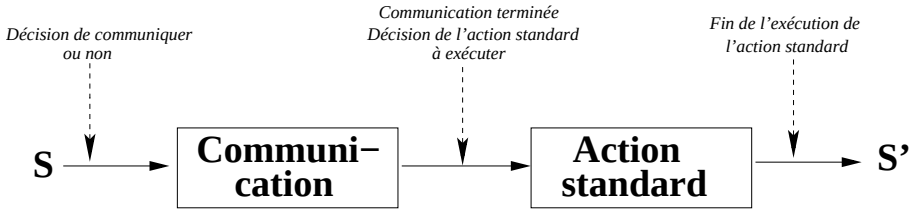


Figure 8.3. *Différentes phases d'une étape de décision*

La politique π_i de chaque agent est alors composée de deux politiques distinctes : une politique de communication π_i^{Σ} et une politique d'exécution des actions standards π_i^a.

Goldman et Zilberstein [GOL 03] ont décrit une extension des DEC-POMDP permettant de modéliser la communication entre les agents. Les DEC-POMDP-COM (processus décisionnels de Markov décentralisé partiellement observables avec communication) sont définis de la même manière que les DEC-POMDP. Deux nouvelles composantes sont cependant ajoutées au formalisme : un langage de communication Σ et une fonction de coût d'envoi des messages C_{Σ}.

Définition 18 *Un* DEC-POMDP-COM *est défini par un tuple* $\langle \mathcal{S}, \mathcal{A}, \Sigma, C_{\Sigma}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ *tel que :*

– $\mathcal{S}$ désigne l'état global du système.

3. La phase d'exécution des tâches est appelée phase « *on-line* ». Elle est opposée à la phase « *off-line* » de calcul des politiques et ayant lieu avant l'exécution des tâches.

– $\mathcal{A} = \langle A_1, \cdots, A_n \rangle$ *est l'ensemble des actions jointes et* A_i *définit l'ensemble des actions* a_i *de l'agent* $\mathcal{A}g_i$.

– Σ *désigne l'alphabet des messages.* $\sigma_i \in \Sigma$ *correspond à un message de l'agent* $\mathcal{A}g_i$.

– C_Σ *associe à tout message un coût. Cette fonction est telle que* $C_\Sigma : \Sigma \rightarrow \mathcal{R}$.

– $\mathcal{T} = S \times A \times S \rightarrow \mathcal{R}$ *définit la fonction de transition.* $\mathcal{T}(s, a, s')$ *correspond à la probabilité que le système passe d'un état* s *à un état* s' *lorsque l'action jointe* a *est exécutée.*

– $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$ *est l'ensemble des observations des agents tel que* Ω_i *est l'ensemble des observations de l'agent* $\mathcal{A}g_i$.

– $\mathcal{O} = S \times A \times S \times \Omega \rightarrow \mathcal{R}$ *définit la fonction d'observation.* $\mathcal{O}(s, a, s', o = \langle o_1, \cdots, o_n \rangle)$ *correspond à la probabilité que chaque agent* $\mathcal{A}g_i$ *observe* o_i *lorsque les agents exécutent l'action jointe* a *à partir de l'état* s *et que le système arrive dans l'état* s'.

– $\mathcal{R}$ *définit la fonction de récompense.* $R(\langle s_1, \cdots, s_n \rangle, \langle a_1, \cdots, a_n \rangle, \langle s'_1, \cdots, s'_n \rangle)$ *est la récompense obtenue par le système lorsque les agents exécutent l'action* $\langle a_1, \cdots, a_n \rangle$ *à partir de l'état* $\langle s_1, \cdots, s_n \rangle$ *et arrivent dans l'état* $\langle s'_1, \cdots, s'_n \rangle$.

Remarquons qu'il est également possible d'ajouter la communication au formalisme des DEC-MDP comme l'ont fait Xuan et Lesser [XUA 01].

Pynadath et Tambe [PYN 02] ont également décrit une extension des MTDP permettant la modélisation de la communication. Un COM-MTDP (*Communicative Multiagent Team Decision Problem*) est défini par un tuple $\langle \mathcal{S}, \mathcal{A}, \Sigma, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{B}, \mathcal{R} \rangle$. La fonction de récompense $\mathcal{R}$ est étendue afin de modéliser le coût des envois de messages.

Comme pour les DEC-POMDP et les MTDP, Seuken et Zilberstein [SEU 05] ont démontré que les modèles des DEC-POMDP-COM et des COM-MTDP étaient équivalents lorsque chaque agent a accès à toutes les informations qu'il a perçues par le passé. Par ailleurs, il a été prouvé que les modèles des DEC-POMDP et des DEC-POMDP-COM étaient équivalents. En effet, les échanges de messages peuvent être considérés comme des actions appartenant à l'ensemble $\mathcal{A}_i$ des actions de chaque agent $\mathcal{A}g_i$ (fusion des ensembles d'actions de communication et des actions standards). En adaptant en conséquence l'espace des états, la fonction de transition et la fonction de récompense, il est alors possible de transformer tout DEC-POMDP-COM en un DEC-POMDP équivalent. Un DEC-POMDP étant, par définition, une classe particulière de DEC-POMDP-COM, on en déduit l'équivalence entre ces deux modèles. A partir de ces différentes équivalences, il peut être déduit que les modèles des DEC-POMDP, DEC-POMDP-COM, MTDP et COM-MTDP sont équivalents lorsque chaque agent a accès à toutes les informations qu'il a perçues par le passé. Les complexités de ces différents problèmes sont par conséquent identiques.

Le coût de la communication et le degré d'observabilité du système influencent toutefois la complexité de ces modèles (tableau 8.3). Lorsque la communication est gratuite, il est possible pour chaque agent de communiquer toutes ses connaissances à tous les autres agents. Si le système est collectivement totalement observable, la communication permet de se ramener à un processus de décision décentralisée complètement observable. Ce dernier peut être représenté par un MMDP [BOU 99a] dont la résolution est P-complète [PAP 87]. Dans le cas où l'état n'est pas collectivement totalement observable, communiquer permet de ramener le problème à un POMDP dont la résolution est PSPACE [PAP 87]. Si les agents n'ont aucune observabilité, alors le problème correspond à un NOMDP (processus décisionnel de Markov non observable) qui est NP-complet.

Lorsque la communication n'est pas gratuite, chaque agent doit trouver un compromis entre le coût de la communication et les bénéfices résultant des informations qu'il aura obtenues ou transmises. Résoudre de manière optimale de tels problèmes consiste à trouver une politique optimale jointe $\pi = \langle \pi_1, \cdots, \pi_n \rangle$ où chaque politique locale comprend deux composantes. Chaque agent doit donc construire une politique $\pi_i = \langle \pi_i^{\Sigma}, \pi_i^a \rangle$ telle que π_i^{Σ} et π_i^a maximisent la récompense jointe.

Lorsque l'état du système est individuellement observable, les agents n'ont aucun besoin de communiquer. Le problème peut se ramener, comme dans le cas de la communication gratuite, à un MMDP. Dans le cas où l'état du système est non observable, les agents sont « aveugles » et n'ont alors aucune information à communiquer. Le problème se ramène à un NOMDP NP-complet.

Observabilité de l' état du système	Individuellement Observable	Collectivement Observable	Collectivement Partiellement Observable	Non Observable
Communication gratuite	P-complet	P-complet	PSPACE-complet	NP-complet
Communication avec un coût	P-complet	NEXP-complet	NEXP-complet	NP-complet

Tableau 8.3. *Observabilité et Complexité en temps*

Le formalisme des COM-MTDP ne pose aucune hypothèse quant à l'observabilité collective de l'état du système et au coût de la communication. Leur complexité varie suivant l'observabilité du système.

8.5. Propriétés et classes particulières de DEC-POMDP

Nous avons vu que la résolution des DEC-POMDP est un problème difficile. Cependant, certaines propriétés, autres que le degré d'observabilité, permettent de diminuer la complexité du problème.

8.5.1. *Transitions, observations et buts*

Parmi ces propriétés, on peut citer l'indépendance des transitions, l'indépendance des observations et l'existence d'un but commun à tous les agents. Dans cette section, nous allons définir ces propriétés et étudier leur influence sur la complexité des DEC-POMDP et DEC-MDP. Ces résultats sont également applicables aux MTDP.

8.5.1.1. *Transition et Observation Indépendantes*

L'indépendance des transitions permet d'identifier des problèmes où les actions des agents sont indépendantes les unes des autres.

Définition 19 *Un* DEC-POMDP *est dit à transitions indépendantes si la fonction de transition peut être factorisée en un produit de probabilités tel que :* $P = \prod_{i=1}^{n} P_i$ *où* $P_i = Pr(s'_i|s_i, a_i)$.

Pour un DEC-POMDP à deux agents, la propriété d'indépendance des transitions se traduit par la formule suivante :

$$\forall s_1, s'_1 \in S_1, \forall s_2, s'_2 \in S_2, \forall a_1 \in A_1, \forall a_2 \in A_2,$$

$$Pr(s'_1|(s_1, s_2), a_1, a_2, s'_2) = Pr(s'_1|s_1, a_1) \text{ et}$$
$$Pr(s'_2|(s_1, s_2), a_2, a_1, s'_1) = Pr(s'_2|s_2, a_2).$$

Lorsqu'un DEC-POMDP est à transitions indépendantes, la probabilité qu'un agent $\mathcal{A}g_i$ passe d'un état s_i à un état s'_i ne dépend que de l'action a_i qu'il a exécutée. Les actions des autres agents n'ont pas d'influence sur la transition de l'agent $\mathcal{A}g_i$.

L'indépendance des observations caractérise les problèmes où les observations de chaque agent sont indépendantes des actions des autres agents.

Définition 20 *Un* DEC-POMDP *est dit à observations indépendantes si la probabilité d'observation* $\mathcal{O}$ *peut être décomposée en* n *probabilités d'observations* $\mathcal{O}_i$ *telles que :*

$$\mathcal{O}_i = Pr(o_i|\langle s_1, \cdots, s_n\rangle, \langle a_1, \cdots, a_n\rangle, \langle s'_1, \cdots, s'_n\rangle, \langle o_1, \cdots, o_{i-1}, o_{i+1}, \cdots, o_n\rangle)$$

Pour un DEC-POMDP à deux agents, la propriété d'indépendance des observations se traduit par la formule suivante :

$$\forall o_1 \in \Omega_1, \forall o_2 \in \Omega_2, \forall s = (s_1, s_2), s' = (s'_1, s'_2) \in \mathcal{S}, \forall a_1 \in A_1, \forall a_{\in} A_2,$$

$$Pr(o_1|(s_1, s_2), a_1, a_2, (s'_1, s'_2), o_2) = Pr(o_1|s_1, a_1, s'_1) \text{ et}$$
$$Pr(o_2|(s_1, s_2), a_1, a_2, (s'_1, s'_2), o_1) = Pr(o_2|s_2, a_2, s'_2) \text{ et}$$
$$\mathcal{O}(o_1, o_2|(s_1, s_2), a_1, a_2, (s'_1, s'_2)) =$$

$$Pr(o_1|(s_1,s_2),a_1,a_2,(s'_1,s'_2),o_2) \times Pr(o_2|(s_1,s_2),a_1,a_2,(s'_1,s'_2),o_1)$$

Un DEC-POMDP à transitions et observations indépendantes modélise un problème où l'influence des actions d'un agent sur les autres est restreinte. En effet, seule la fonction de récompense dépend de l'action jointe exécutée. La conjugaison de ces deux propriétés permet, dans certains cas, de diminuer la complexité.

Lors de la résolution des problèmes de décision décentralisée, les agents doivent faire face au manque d'observabilité de l'état global du système. Afin de calculer une politique globale optimale, chaque agent doit garder en mémoire toute la séquence de ses observations puisque chaque décision dépend de cette séquence. La taille de la politique de l'agent est donc exponentielle en $|\Omega_i|^T$ où $|\Omega_i|$ désigne la taille de l'ensemble des observations de l'agent $\mathcal{A}g_i$ et T est l'horizon du problème. L'évaluation de la politique est par conséquent de complexité exponentielle. De plus, il existe $|A_i|^{|\Omega_i|^T}$ politiques devant être évaluées pour chaque agent, d'où une complexité NEXP.

Goldman et Zilberstein [GOL 04] ont montré que la résolution d'un DEC-MDP à transitions et observations indépendantes nécessite que l'agent ne garde en mémoire que sa dernière observation. Ainsi, la taille des politiques locales des agents diminue. La politique d'un agent $\mathcal{A}g_i$ a alors une taille polynomiale en $|S_i|T$ où $|S_i|$ correspond à la taille de l'espace d'états de l'agent $\mathcal{A}g_i$. L'évaluation d'une telle politique peut être réalisée en temps polynomial. Cependant, il existe un nombre exponentiel $|A_i|^{|S_i|T}$ de politiques. Afin de calculer la politique optimale, il est nécessaire d'évaluer tout l'espace des politiques, c'est pourquoi l'algorithme est NP-complet.

En ce qui concerne les DEC-POMDP, les propriétés d'indépendance des transitions et des observations n'entraînent pas de diminution de la complexité. En effet, chaque agent doit, dans tous les cas, mémoriser toute la séquence de ses observations et la taille des politiques locales reste inchangée.

8.5.1.2. *Agents orientés vers un but*

Un processus orienté vers un but est un processus dans lequel les agents cherchent à atteindre des états globaux spécifiques correspondant à des états où le système a atteint son but. Par exemple, des agents devant transférer des objets d'un point A à un point B auront atteint un état but quand tous les objets seront au point B.

Définition 21 *Un* DEC-POMDP *est dit orienté par des buts si les conditions suivantes sont remplies :*

1) Il existe un sous-ensemble non vide $\mathcal{G}$ de l'ensemble des états $\mathcal{S}$ représentant les états but du système. Au moins un état $g \in \mathcal{G}$ est accessible par une politique jointe.

2) Le problème a un horizon fini T.

3) Toute action a_i d'un agent $\mathcal{A}g_i$ a un coût $C(a_i) < 0$.

4) La fonction de récompense est telle que $R(s, \langle a_1, \cdots, a_n \rangle, s') = \sum_{i=1}^{n} C(a_i)$.

5) Si à l'instant T, le système a atteint un état but $s \in \mathcal{G}$ alors une récompense supplémentaire positive est attribuée au système pour avoir atteint un état but.

Un tel DEC-POMDP est communément noté GO-DEC-POMDP.

Définition 22 *Un GO-*DEC-POMDP *est dit à coût uniforme si toutes les actions ont le même coût.*

La résolution de DEC-MDP à transitions et observations indépendantes, ayant un seul but et à coût uniforme, est un problème P-complet [GOL 04]. Chaque agent doit, dans ce type de problème, suivre une politique qui minimise les coûts. En raison de l'hypothèse d'uniformité des coûts, ceci revient à calculer la politique de plus court chemin, par un algorithme de programmation dynamique P-complet.

En revanche, la résolution d'un DEC-MDP ou d'un DEC-POMDP orienté par un but, à transitions et observations non indépendantes reste un problème NEXP [GOL 04].

8.5.2. DEC-MDP *dirigés par les événements*

Becker et al. [BEC 04a] ont identifié une autre classe de DEC-MDP dont la complexité est moins importante que celles des DEC-MDP classiques : les DEC-MDP dirigés par les événements ou *Event Driven Decentralized Markov Decision Processes* (ED-DEC-MDP).

Les DEC-MDP décrivent des problèmes dans lesquels les interactions entre les agents correspondent à des dépendances entre leurs actions. Ces dernières se traduisent par des contraintes entre les agents, comme par exemple des contraintes temporelles ou bien des contraintes d'ordre entre les actions. Cette classe de DEC-MDP permet de représenter des contraintes sur l'exécution des tâches qui sont souvent rencontrées en pratique et non modélisées dans le formalisme classique des processus décisionnels de Markov décentralisés. Il est ainsi possible de formaliser des problèmes dans lesquels le résultat de l'action d'un agent dépend de l'exécution d'autres actions effectuées par d'autres agents.

Différents types de relations peuvent ainsi être modélisées. Parmi elles, on trouve les relations de précédence : un agent $\mathcal{A}g_i$ ne peut exécuter la tâche t_i que si l'agent $\mathcal{A}g_j$ a fini d'exécuter la tâche t_j. De même, il est possible de représenter des dépendances de qualité : si l'agent $\mathcal{A}g_j$ a terminé la tâche t_j lorsque la tâche t_i est exécutée, alors la qualité d'exécution de t_i sera meilleure. Afin de représenter ces dépendances, le formalisme des ED-DEC-MDP se base sur le langage TAEMS [DEC 93] de description des tâches.

Les DEC-MDP supposent que les états locaux des agents sont localement totalement observables. Enfin, la fonction de récompense $\mathcal{R}$ d'un ED-DEC-MDP est à récompenses indépendantes, c'est-à-dire qu'elle peut être décomposée en une somme de fonctions de récompense locales telles que :

$$\mathcal{R}(\langle s_1, \cdots, s_n\rangle, \langle a_1, \cdots, a_n\rangle, \langle s'_1, \cdots, s'_n\rangle) = \sum_{i=1}^{n} R_i(s_i, a_i, s'_i)$$

La résolution d'un ED-DEC-MDP est un problème NP-complet. En effet, un ED-DEC-MDP a une complexité exponentielle en le nombre d'états et doublement exponentielle en nombre de dépendances. Les DEC-MDP étant localement totalement observables, une politique fait correspondre une action à chaque état (et non à chaque observation). La taille d'une telle politique est alors exponentielle en le nombre d'états. Le nombre d'états étant exponentiel en le nombre de dépendances, le nombre de politiques est doublement exponentiel en le nombre de dépendances.

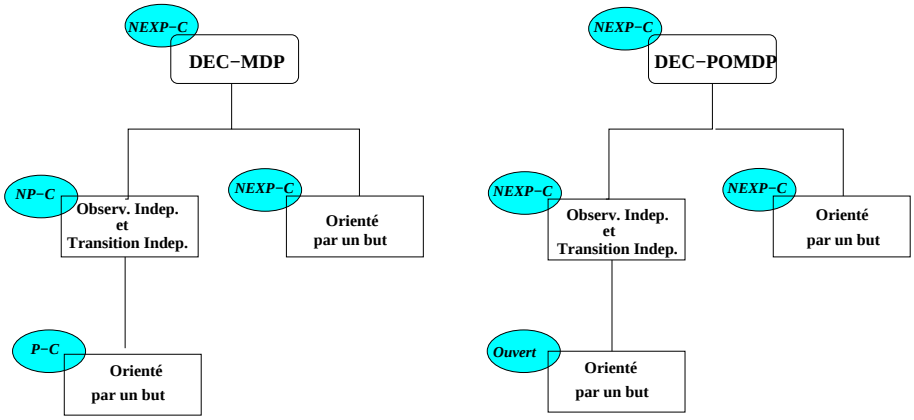


Figure 8.4. *Complexité des problèmes des classes de* DEC-MDP *et* DEC-POMDP

Les propriétés que nous venons de décrire permettent d'identifier des sous-classes de DEC-MDP de complexité moins importante. Ces résultats sont résumés par la figure 8.4. La solution optimale à ces problèmes peut alors être obtenue plus facilement.

8.5.3. *Modélisation de* DEC-MDP *avec contraintes*

Bien que les DEC-MDP permettent la modélisation de problèmes décisionnels multi-agents, leur utilisation pour la résolution de problèmes concrets, par exemple pour la planification de tâches dans des colonies de robots autonomes, peut s'avérer difficile. En effet, les DEC-MDP ne permettent pas la prise en compte de certaines données du problème comme les contraintes sur l'exécution des tâches. De plus, la complexité de leur résolution est telle qu'il est extrêmement difficile de calculer une solution optimale ou une solution approchée pour des problèmes de plus d'une dizaine de tâches et

d'agents. Afin de pallier ces difficultés, Beynier et al. [BEY 04] ont cherché à améliorer la modélisation du temps et des actions dans les DEC-MDP avec le modèle OC-DEC-MDP. A partir de cette formalisation du problème, Beynier et al. [BEY 05, BEY 06] ont ensuite cherché à développer des algorithmes de résolution approchée efficaces qui puissent être utilisés pour résoudre des problèmes de taille importante.

8.5.3.1. *Problèmes envisagés*

Les problèmes considérés dans ce travail sont constitués d'un ensemble d'agents devant réaliser un ensemble de tâches tout en respectant différents types de contraintes.

Définition 23 ***Mission***

*Une **mission** est définie par un couple $\langle \mathcal{A}g, T \rangle$ tel que :*

*– $\mathcal{A}g = \{\mathcal{A}g_1, \cdots, \mathcal{A}g_n\}$ définit l'**ensemble des** n **agents** $\mathcal{A}g_i \in \mathcal{A}g$.*

*– $T = \{t_1, \cdots, t_p\}$ est l'**ensemble des tâches** devant être exécutées par les agents.*

A partir de l'étude d'applications en robotique collective, Beynier et al. ont identifié différentes contraintes devant être respectées lors de l'exécution d'une mission. Les contraintes suivantes ont plus particulièrement été recensées :

– des contraintes temporelles,

– des contraintes de précédence,

– des contraintes de ressources,

– une communication limitée voire impossible,

– des capacités de calcul restreintes.

Les deux derniers types de contraintes répertoriés influencent plus particulièrement le processus délibératif des agents. Ils imposent que les agents prennent des décisions en limitant les calculs et les communications. Les trois premiers types de contraintes portent sur l'exécution des tâches. Pour chaque tâche, ils définissent des conditions au succès de son exécution. Les **contraintes temporelles** définissent, pour chaque tâche, une fenêtre temporelle durant laquelle l'exécution de la tâche doit avoir lieu. Les **contraintes de précédence** traduisent des contraintes d'ordre entre les tâches. Elles permettent la formalisation de pré-conditions telles que « la tâche A doit être terminée pour que la tâche B puisse commencer ». Enfin les **contraintes de ressources** posent les conditions sur les ressources nécessaires à l'exécution d'une tâche.

Le problème de décision auquel chaque agent doit alors faire face consiste à décider quelle tâche exécuter et quand, de façon à maximiser l'espérance de gain du système.

8.5.3.2. *Le modèle* OC-DEC-MDP

Beynier et al. se sont penchés sur la mise en place d'un modèle basé sur les processus décisionnels de Markov décentralisés (DEC-MDP), qui permette la formalisation

des contraintes et autorise une modélisation plus riche du temps et des actions. L'approche élaborée, nommée OC-DEC-MDP (processus décisionnel de Markov décentralisés avec coût occasionné), permet de représenter les contraintes temporelles, de ressources et de précédence entre les tâches. Afin de limiter la taille du DEC-MDP obtenu et d'être en mesure de gérer des problèmes réels de taille importante, le problème de décision multi-agent est décomposé en un ensemble de problèmes plus simples. Le problème initial est ainsi représenté par un ensemble de MDP locaux, où chaque MDP définit le problème décisionnel d'un agent.

Définition 24 *Un* OC-DEC-MDP *pour n agents est constitué d'un ensemble de n* MDP *locaux, un pour chaque agent. Le* MDP *local d'un agent $\mathcal{A}g_i$ est défini par un tuple $\langle \mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i, \mathcal{R}_i \rangle$ tel que :*

– $\mathcal{S}_i$ est l'ensemble fini des états de l'agent $\mathcal{A}g_i$.

– $\mathcal{A}_i$ est l'ensemble fini des actions de l'agent $\mathcal{A}g_i$.

– $\mathcal{T}_i$ définit la fonction de transition de l'agent $\mathcal{A}g_i$.

– $\mathcal{R}_i : T_i \rightarrow \mathbb{R}$ décrit la fonction de récompense de l'agent. $\mathcal{R}_i(t_i)$ est la récompense obtenue par l'agent lorsque la tâche t_i a été exécutée en respectant les contraintes.

Chaque composante des MDP locaux est définie de manière à tenir compte des différentes contraintes du problème. Ainsi, l'espace d'états est composé de trois types d'états :

– les états de **succès** dans lesquels un agent arrive lorsqu'il a exécuté une tâche avec succès ;

– les états d'**échec partiel** correspondant à des situations où les contraintes de précédence entre les tâches n'ont pas été respectées ;

– les états d'**échec total** correspondant à des situations où les contraintes temporelles ou les contraintes de ressources ne sont pas respectées.

De la même façon, la fonction de transition rend compte de trois grands types de transitions : succès, échec partiel et échec total.

Il a été démontré que la complexité en temps d'un OC-DEC-MDP est exponentielle en le nombre d'états. Les contraintes du problème influencent la taille de l'espace d'états et donc la taille et le nombre de politiques. En effet, elles restreignent le nombre d'états et d'actions possibles pour chaque agent. Cependant, ces contraintes n'affectent en rien la complexité au pire cas du problème, le nombre de politiques possibles demeure exponentiel en le nombre d'états. Les contraintes ne réduisent donc pas la classe de complexité du problème.

8.6. La résolution des DEC-POMDP

Dans la suite de ce chapitre, nous allons nous intéresser plus particulièrement à la résolution des problèmes formalisés sous forme de DEC-POMDP. Cette section présente différents algorithmes permettant de résoudre des DEC-POMDP généraux ou se focalisant sur la résolution de certaines classes de DEC-POMDP.

Résoudre un DEC-POMDP revient à trouver un ensemble de politiques, une par agent, dont l'exécution parallèle et synchrone maximise le critère de performance choisi. La fonction de récompense $\mathcal{R}$ du DEC-POMDP est propre au système ; il n'y a pas de récompense individuelle pour chaque agent. Les DEC-POMDP formalisent donc des problèmes de décision pour des systèmes multi-agents purement coopératifs. Les critères de performance s'apparentent à ceux du POMDP mono-agent. Nous allons nous concentrer ici sur le critère fini et le critère γ-pondéré.

Nous allons présenter dans ce qui suit trois types d'algorithmes : des algorithmes optimaux pour les DEC-POMDP à horizon fini, des algorithmes approximatifs pour les DEC-POMDP à horizon fini, et des algorithmes approximatifs pour les DEC-POMDP à horizon infini.

8.6.1. *Algorithmes de résolution optimaux*

Une politique pour un POMDP à horizon fini peut toujours être représentée sous forme d'un arbre de décision [KAE 98] que l'on notera q (cf. chapitre 3). L'action associée à la racine de l'arbre est dénotée $\alpha(q)$ et le sous-arbre associé à une observation o est dénoté $q(o)$. $\lambda(q,i)$ dénote la i^{ime} feuille de l'arbre q. Dans le cas du contrôle décentralisé, une politique optimale pour le système consiste en un ensemble de politiques locales. Si l'on note q_i^T un arbre de profondeur T, le but de la planification pour DEC-POMDP est de calculer une politique jointe $\boldsymbol{q}^T = \langle q_1^T, q_2^T, \ldots, q_n^T \rangle$ de profondeur T maximisant l'espérance

$$E\left(\sum_{t=1}^{T} \mathcal{R}(s_t, \langle a_1, a_2, \ldots, a_n \rangle_t, s_{t+1})\right) \tag{8.1}$$

La valeur d'une politique jointe $\boldsymbol{q}^T$ pour un état initial s peut être déterminée par programmation dynamique avec

$$V(s, \boldsymbol{q}^T) = \sum_{\boldsymbol{o} \in \Omega} P(\boldsymbol{o}|s, \boldsymbol{q}^T) \left[\sum_{s' \in \mathcal{S}} P(s'|s, \boldsymbol{q}^T, \boldsymbol{o}) V(s', \boldsymbol{q}^T(\boldsymbol{o}))\right], \tag{8.2}$$

où $\boldsymbol{o} = \langle o_1, \ldots, o_n \rangle$ dénote une observation jointe et $\boldsymbol{q}^T(\boldsymbol{o})$ la sous-politique jointe à horizon $(T-1)$ des agents après observation de $\boldsymbol{o}$.

Nous précisons qu'une recherche exhaustive dans l'ensemble des politiques possibles devrait considérer un nombre total de

$$\left(|\mathcal{A}|^{\frac{1-|\Omega|^T}{1-|\Omega|}}\right)^n \tag{8.3}$$

politiques. Deux approches de planification ont été proposées récemment pour effectuer ce calcul de manière plus efficace. Nous allons les présenter dans les sections suivantes.

8.6.1.1. *Résolution par programmation dynamique*

L'algorithme de programmation dynamique présenté par Hansen et al. [HAN 04] a été le premier algorithme non-trivial permettant de résoudre de manière générale les DEC-POMDP à horizon fini. Il est basé sur la génération exhaustive de politiques possibles, puis l'élagage itératif de politiques dominées.

Nous allons introduire maintenant l'extension de la notion d'état estimé (cf. chapitre 3) au cas multi-agent décentralisé, c'est-à-dire l'information probabiliste dont dispose un agent $\mathcal{A}g_i$ au moment t de l'exécution. Elle peut être vue comme la composition d'une estimation de l'état sous-jacent s_t, mais aussi du comportement futur des autres agents. Si on note Q_i^t l'ensemble de politiques à horizon t possibles pour l'agent $\mathcal{A}g_i$, et $\boldsymbol{Q}_{-i}^t = \langle Q_1^t, \cdots, Q_{i-1}^t, Q_{i+1}^t, \cdots, Q_n^t \rangle$ les ensembles des politiques possibles pour les agents $\mathcal{A}g_j, j \neq i$, alors l'état estimé multi-agent est une distribution de probabilité sur $\mathcal{S}$ et sur $\boldsymbol{Q}_{-i}^t$.

Définition 25 (Etat estimé multi-agent) *On appelle* état estimé multi-agent b_i *une distribution de probabilité sur la configuration possible du système, à savoir l'état réel sous-jacent et les politiques futures des autres agent participants :* $b_i \in \Delta(\mathcal{S} \times \boldsymbol{Q}_{-i})$.

A la différence de l'état estimé d'un POMDP, la dimension de l'état estimé multi-agent dépend du nombre de politiques possibles pour chaque agent, et donc au pire cas de toutes les politiques envisageables. On peut noter que, pour le cas spécial d'un DEC-POMDP à un seul agent, cette notion d'état estimé se réduit à la définition rencontrée dans le cas POMDP (3).

L'état estimé multi-agent synthétise la vision partielle et subjective que possède un agent $\mathcal{A}g_i$ sur le système. Il permet de définir une fonction de valeur V_i propre à cet agent. Si on note $\boldsymbol{q}_{-i}$ une politique jointe pour tous les agents sauf l'agent $\mathcal{A}g_i$, $\langle \boldsymbol{q}_{-i}, q_i \rangle$ est une politique jointe complète et la valeur de la politique q_i pour l'état estimé $\boldsymbol{b}_i$ peut-être exprimée à l'aide de la fonction de valeur du système :

$$V_i(\boldsymbol{b}_i, q_i) = \sum_{s \in \mathcal{S}} \sum_{\boldsymbol{q}_{-i} \in \boldsymbol{Q}_{-i}} \boldsymbol{b}_i(s, \boldsymbol{q}_{-i}) V(s, \langle \boldsymbol{q}_{-i}, q_i \rangle). \tag{8.4}$$

Il est important de préciser qu'une politique locale ne peut pas être évaluée seule dans un cadre multi-agent, qu'il soit coopératif ou non. La valeur d'une politique dépend en effet toujours du comportement futur du système entier, donc du choix des politiques de tous les agents participants. En ce sens, le contrôle stochastique multi-agent se formalise dans le cadre des jeux de Markov. L'importance de la prise en compte du comportement futur des autres agents est mis en évidence par la notion de *réponse optimale*.

Définition 26 (Politique de réponse optimale) *On appelle $B_i(\boldsymbol{b}_i)$ la* politique de réponse optimale *de l'agent $\mathcal{A}g_i$ pour l'état estimé multi-agent $\mathbf{b}_i$:*

$$B_i(\boldsymbol{b}_i) = \arg\max_{q_i \in Q_i} V_i(\boldsymbol{b}_i, q_i).$$

Il s'agit donc du meilleur comportement de l'agent $\mathcal{A}g_i$ pour compléter le groupe, sous condition que tout agent $\mathcal{A}g_j, j \neq i$ a déjà préalablement choisi sa politique individuelle. Or, le choix de la politique de réponse optimale d'un agent $\mathcal{A}g_j$ nécessite que la politique de l'agent $\mathcal{A}g_i$ lui-même soit déjà déterminée. Déterminer la politique optimale jointe par n calculs de réponses optimales individuelles n'est donc pas envisageable. L'approche de programmation dynamique consiste à faire l'inverse, c'est-à-dire à identifier les politiques qui ne constituent jamais une réponse optimale. Pour cela, nous allons étendre au cas multi-agent le concept de *politique utile*, introduit dans le cas des POMDP (cf. chapitre 3). Une politique est *utile* pour l'agent $\mathcal{A}g_i$ si elle représente une réponse optimale pour au moins un état estimé. Une politique qui n'est utile sous aucune configuration est appelée politique *dominée*.

Définition 27 (Politique dominée) *On appelle* politique dominée *une politique $q_i \in Q_i$ qui est sous-optimale sur l'ensemble de l'espace des états estimés, ce qui veut dire que, pour chaque état estimé $\mathbf{b}_i$ possible, il existe au moins une autre politique $\tilde{q}_i$ qui est au moins aussi performante :*

$$(\forall \boldsymbol{b}_i)(\exists \tilde{q}_i \in Q_i \backslash \{q_i\}) \quad \textit{tel que} \quad V_i(\boldsymbol{b}_i, \tilde{q}_i) \geq V_i(\boldsymbol{b}_i, q_i)$$

Une politique qui n'est pas entièrement dominée est appelée politique utile.

Au lieu de déterminer les politiques de réponse optimale pour chaque agent, l'approche de programmation dynamique de Hansen et al. vise à identifier et ensuite à élaguer toutes les politiques dominées. Identifier une politique dominée revient à résoudre un programme linéaire simple :

$$\begin{aligned}
\text{maximiser} \quad & \epsilon \\
\text{sous contraintes} \quad & V_i(\boldsymbol{b}_i, \tilde{q}_i) + \epsilon \leq V_i(\boldsymbol{b}_i, q_i) && (\forall \tilde{q}_i \neq q_i) \\
& \sum_{s \in S} \sum_{\boldsymbol{q}_{-i} \in \boldsymbol{Q}_{-i}} \boldsymbol{b}_i(s, \boldsymbol{q}_{-i}) = 1 \\
& \boldsymbol{b}_i(s, \boldsymbol{q}_{-i}) \geq 0 && (\forall s \in \mathcal{S})(\forall \boldsymbol{q}_{-i} \in \boldsymbol{Q}_{-i}).
\end{aligned}$$

Si le résultat est inférieur ou égal à zéro ($\epsilon \leq 0$), alors la politique q_i est entièrement dominée et peut être supprimée.

Algorithme 8.1 : ProgDynPourDEC-POMDP

Entrées : Un DEC-POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ et un horizon T
pour *tout agent* $\mathcal{A}g_i$ *:* **faire**
  Initialiser $Q_i^0 \leftarrow \emptyset$
pour $t = 1$ ***à*** $t = T$ **faire**
  pour *tout agent* $\mathcal{A}g_i$ *:* **faire**
    $Q_i^t \leftarrow$ `GénérationExhaustive`(Q_i^{t-1})
  tant que *toutes les politiques entièrement dominées n'ont pas été élaguées* **faire**
    Trouver un agent $\mathcal{A}g_i$ et une politique $q_i \in Q_i^t$ avec

$$\forall \boldsymbol{b}_i,\ \exists \tilde{q}_i \in Q_i^t \backslash \{q_i\} \text{ tel que } V_i(\boldsymbol{b}_i, \tilde{q}_i) \geq V_i(\boldsymbol{b}_i, q_i)$$

    $Q_i^t \leftarrow Q_i^t \setminus \{q_i\}$
Sorties : Un ensemble de politiques utiles pour chaque agent

L'algorithme de programmation dynamique multi-agent consiste en une alternance de deux opérateurs : la génération exhaustive des politiques possibles et l'élagage de politiques dominées. Ce processus est répété pour chaque horizon, en commençant par les politiques à horizon 1. L'algorithme 8.1 résume cette approche par programmation dynamique. Le choix d'une politique jointe optimale pour un état initial s_0 se fait alors par simple maximisation :

$$\boldsymbol{q}_{s_0}^* = \underset{\boldsymbol{q}^T \in Q_1^T \times \cdots \times Q_n^T}{\arg\max} V(s_0, \boldsymbol{q}^T). \tag{8.5}$$

Algorithme 8.2 : GénérationExhaustive

Entrées : Un ensemble de politiques Q_i^t à horizon t
$k = |\Omega_i|$
pour ***toute action*** $a_i \in \mathcal{A}_i$ **faire**
  pour $p = 1$ ***à*** $|Q_i^t|^k$ **faire**
    En dénotant $\langle q_i^{o_1}, \ldots, q_i^{o_k} \rangle_p$ la p-ième sélection de k politiques parmi Q_i^t, créer une nouvelle politique q_i^{t+1} avec :
    pour $j = 1$ ***à*** k **faire**
      $q_i^{t+1}(o_j) := q_i^{o_j}$
    $\alpha(q_i^{t+1}) := a_i$
    $Q_i^{t+1} \leftarrow Q_i^{t+1} \bigcup \{q_i^{t+1}\}$
Sorties : Un ensemble de politiques Q_i^{t+1} à horizon $(t+1)$

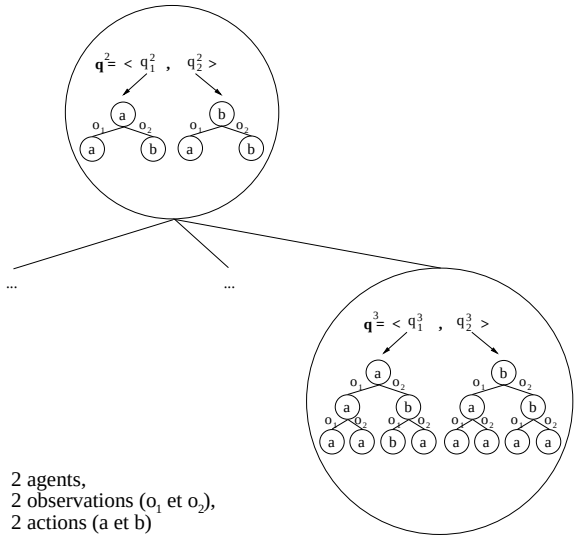


Figure 8.5. *Une partie de l'arbre de recherche A* multi-agent. La figure montre une politique jointe à horizon 2 avec un de ses fils développé, une politique jointe à horizon 3.*

8.6.1.2. *Résolution par recherche heuristique*

Une autre approche de résolution a été proposée par Szer et al. [SZE 05]. Il s'agit de l'extension multi-agent de l'algorithme de recherche heuristique A*, appelée MAA* (A* multi-agent). Tandis que l'approche par programmation dynamique procède de bas en haut, en élaguant pour chaque horizon les politiques dominées, l'algorithme MAA* fonctionne dans le sens inverse et se sert d'une fonction heuristique pour exclure des politiques dominées.

La recherche se fait de manière incrémentale : les feuilles de l'arbre de recherche contiennent des solutions partielles au problème et, à chaque itération, la meilleure solution apparente est choisie pour être développée d'une étape supplémentaire. Une feuille de l'arbre contient une politique jointe à horizon $t < T$. Développer une politique jointe $\boldsymbol{q}^t$ à horizon t signifie construire tous les fils de $\boldsymbol{q}^t$, c'est-à-dire toutes les politiques jointes $\boldsymbol{q}^{t+1}$ à horizon $t+1$ qui coïncident avec $\boldsymbol{q}^t$ pour les t premiers niveaux. Une partie d'un tel arbre de recherche apparaît sur la figure 8.5. La base de tout algorithme de recherche heuristique est ensuite la décomposition de la fonction d'évaluation en une partie exacte pour une solution partiellement construite, et une estimation heuristique pour la partie restante. Dans le cas présent, il s'agit d'évaluer des politiques jointes $\boldsymbol{q}^t$ pour un certain horizon t, et de trouver une heuristique H^{T-t} pour estimer le comportement potentiel du système après l'exécution de $\boldsymbol{q}^t$. Ce comportement futur est appelé le complément d'une politique jointe, et il est noté Δ^{T-t}. Il s'agit d'un ensemble de politiques de profondeur $(T-t)$ qui peuvent être attachés

aux feuilles de $\boldsymbol{q}^t$, tel que $\langle \boldsymbol{q}^t, \Delta^{T-t} \rangle$ constitue une politique jointe complète de profondeur T. La différence entre les deux évaluations $V(s_0, \boldsymbol{q}^t)$ et $V(s_0, \langle \boldsymbol{q}^t, \Delta^{T-t} \rangle)$ détermine la valeur du complément Δ^{T-t} :

$$V(\Delta^{T-t}|s_0, \boldsymbol{q}^t) := V(s_0, \langle \boldsymbol{q}^t, \Delta^{T-t} \rangle) - V(s_0, \boldsymbol{q}^t). \tag{8.6}$$

D'une manière équivalente, la valeur de toute politique jointe peut être décomposée en la valeur d'une racine et la valeur du complément :

$$V(s_0, \langle \boldsymbol{q}^t, \Delta^{T-t} \rangle) = V(s_0, \boldsymbol{q}^t) + V(\Delta^{T-t}|s_0, \boldsymbol{q}^t). \tag{8.7}$$

Pour une politique jointe $\boldsymbol{q}^t$ partiellement construite, une borne supérieure sur son éventuelle valeur à horizon T peut être évaluée en utilisant le meilleur complément possible :

$$\overline{V}^{*T}(s_0, \boldsymbol{q}^t) = V(s_0, \boldsymbol{q}^t) + \max_{\Delta^{T-t}} V(\Delta^{T-t}|s_0, \boldsymbol{q}^t). \tag{8.8}$$

Calculer cette valeur explicitement revient à une recherche exhaustive dans l'espace des politiques. La fonction heuristique H^{T-t} a pour but de surestimer efficacement la valeur du meilleur complément.

Définition 28 (Fonction heuristique multi-agent) *La fonction d'évaluation heuristique pour guider la recherche meilleur-d'abord multi-agent doit garantir la propriété suivante :*

$$H^{T-t}(s_0, \boldsymbol{q}^t) \geq \max_{\Delta^{T-t}} V(\Delta^{T-t}|s_0, \boldsymbol{q}^t). \tag{8.9}$$

Une heuristique vérifiant cette propriété est dite admissible.

Pour toute heuristique admissible H, la fonction d'évaluation de l'algorithme A* multi-agent peut finalement s'écrire :

$$\overline{V}^{T}(s_0, \boldsymbol{q}^t) = V(s_0, \boldsymbol{q}^t) + H^{T-t}(s_0, \boldsymbol{q}^t) \geq \overline{V}^{*T}(s_0, \boldsymbol{q}^t). \tag{8.10}$$

La partie essentielle de l'algorithme de recherche s'avère la définition d'une telle fonction heuristique, c'est-à-dire une borne supérieure à la valeur d'un DEC-POMDP partiel. Il a été constaté par [LIT 95b] et puis par [HAU 00] que la fonction de valeur d'un POMDP peut être facilement surestimée à l'aide du MDP sous-jacent. La fonction de valeur d'un POMDP est définie pour l'espace des états estimés (cf. chapitre 3). La valeur optimale d'un état de croyance dans un POMDP peut ainsi être approximée par

$$V^*_{pomdp}(\boldsymbol{b}) \leq \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V^*_{mdp}(s). \tag{8.11}$$

Une propriété semblable peut être établie dans le cas décentralisé. Si $P(s|s_0, \boldsymbol{q})$ dénote la probabilité que l'état du système soit s après l'exécution de la politique jointe $\boldsymbol{q}$ en

s_0, alors une deuxième heuristique h peut être utilisée pour surestimer la valeur du DEC-POMDP optimal :

$$h^t(s) \geq V^{*t}(s). \tag{8.12}$$

L'heuristique h permet la définition de toute une classe de fonctions heuristiques H comme suit :

$$H^{T-t}(s_0, \boldsymbol{q}^t) := \sum_{s \in \mathcal{S}} P(s|s_0, \boldsymbol{q}^t) h^{T-t}(s). \tag{8.13}$$

Intuitivement, toute heuristique H est admissible, puisqu'elle simule le fait que le vrai état du système est révélé aux agents après l'exécution de la politique jointe $\boldsymbol{q}^t$. Ceci constitue une information supplémentaire que les agents n'ont pas au moment de l'exécution. L'avantage de la fonction H réside dans le fait qu'elle permet d'approximer une infinité de distributions alors que h ne doit être évaluée que pour les $|\mathcal{S}|$ états du système.

Lemme 1 *Si h est admissible, toute fonction heuristique H est admissible.*

PREUVE.– Pour prouver ce lemme, nous avons besoin de détailler ce qui se passe effectivement après l'exécution de la politique jointe $\boldsymbol{q}^t$: chaque agent $\mathcal{A}g_i$ a exécuté son arbre de politique q_i jusqu'à une feuille suite à la séquence d'observations $\theta_i = (o_i^1, \ldots, o_i^t)$. Le complément potentiel $\Delta^{T-t}(\theta_i)$ contient alors un arbre de profondeur $(T-t)$ que l'agent $\mathcal{A}g_i$ doit exécuter pendant les $(T-t)$ pas de temps restants. De la même manière, le vecteur $\boldsymbol{\theta} = \langle \theta_1, \ldots, \theta_n \rangle$ représente les séquences d'observations individuelles pour tous les agents, et on peut noter $\Delta^{T-t}(\boldsymbol{\theta})$ l'ensemble des compléments à horizon $(T-t)$ pour l'équipe d'agents toute entière. Sa valeur dépend évidemment de la distribution sous-jacente des états, correspondant aux séquences d'observations $\boldsymbol{\theta}$. On peut donc écrire pour la valeur de tout complément de politique :

$$\begin{aligned}
V(\Delta^{T-t}|s_0, \boldsymbol{q}^t) &= \sum_{\boldsymbol{\theta} \in \Theta^t} P(\boldsymbol{\theta}|s_0, \boldsymbol{q}^t) V(\Delta^{T-t}(\boldsymbol{\theta})|s_0, \boldsymbol{q}^t) \\
&= \sum_{\boldsymbol{\theta} \in \Theta^t} P(\boldsymbol{\theta}|s_0, \boldsymbol{q}^t) \left[\sum_{s \in \mathcal{S}} P(s|\boldsymbol{\theta}) V(s, \Delta^{T-t}(\boldsymbol{\theta})) \right] \\
&\leq \sum_{\boldsymbol{\theta} \in \Theta^t} P(\boldsymbol{\theta}|s_0, \boldsymbol{q}^t) \left[\sum_{s \in \mathcal{S}} P(s|\boldsymbol{\theta}) V^{*T-t}(s) \right] \\
&= \sum_{s \in \mathcal{S}} \sum_{\boldsymbol{\theta} \in \Theta^t} P(s|\boldsymbol{\theta}) P(\boldsymbol{\theta}|s_0, \boldsymbol{q}^t) V^{*T-t}(s) \\
&= \sum_{s \in \mathcal{S}} P(s|s_0, \boldsymbol{q}^t) V^{*T-t}(s) \\
&\leq \sum_{s \in \mathcal{S}} P(s|s_0, \boldsymbol{q}^t) h^{T-t}(s) = H^{T-t}(s_0, \boldsymbol{q}^t). \qquad \square
\end{aligned}$$

L'intérêt principal de l'heuristique H réside dans la simplicité de son évaluation. D'abord, la valeur de l'heuristique $h^t(s)$ peut être réutilisée dans le calcul de H dans (8.13) pour chaque nœud de l'arbre de recherche qui se trouve à la même profondeur t. Ceci réduit le nombre d'évaluations de $|\Omega^{t^n}|$ à $|\mathcal{S}|$. Ensuite, le calcul de h lui-même peut apporter des bénéfices. Il existe plusieurs façons de déterminer une fonction h admissible, que nous allons lister dans les sections ci-après.

8.6.1.2.1. L'heuristique MDP

Une première approche consiste en l'utilisation du MDP sous-jacent —mono-agent et complètement observable. Cette méthode a déjà été employée par [WAS 96] puis par [GEF 98] pour résoudre des POMDP par recherche heuristique. Le MDP sous-jacent est caractérisé par une d'observation complète, c'est-à-dire que le véritable état du système est dévoilé à chaque moment, ainsi que par le choix centralisé d'une action jointe $a = \langle a_1, \ldots, a_n \rangle$. Ceci implique que le contrôle du MDP peut être plus efficace que celui du DEC-POMDP et que sa fonction de valeur constitue par conséquent une borne supérieure pour celle du DEC-POMDP correspondant :

$$h^{T-t}(s) := V_{mdp}^{T-t}(s). \tag{8.14}$$

Résoudre un MDP peut se faire par des méthodes de recherche ou par programmation dynamique, avec une complexité polynomiale dans le pire cas (cf. chapitre 1).

8.6.1.2.2. L'heuristique POMDP

Une heuristique plus proche de la vraie valeur du DEC-POMDP fait appel au POMDP centralisé correspondant, c'est-à-dire à un processus sous-jacent qui simule un agent capable de connaître à chaque moment les observations de tous les agents :

$$h^{T-t}(s) := V_{pomdp}^{T-t}(s). \tag{8.15}$$

Une telle heuristique reste admissible, puisqu'elle permet de considérer des observations et des actions jointes, donc de coordonner les agents de la meilleure façon, ce qui n'est justement plus possible dans le cas de l'exécution décentralisée. Résoudre un POMDP est en général PSPACE-complet. Par conséquent, l'heuristique POMDP est plus complexe à calculer que l'heuristique MDP, mais le fait qu'elle soit plus proche de la vraie valeur du DEC-POMDP peut permettre l'exclusion d'un nombre plus important de nœuds et ainsi aboutir à une meilleure performance finale de l'algorithme de recherche.

8.6.1.2.3. L'heuristique DEC-POMDP

Un cas particulier de fonction heuristique consiste à utiliser la solution optimale au DEC-POMDP lui-même, calculée par exemple de manière récurrente par MAA* sur l'horizon restant :

$$h^{T-t}(s) := V_{dec-pomdp}^{T-t}(s) = MAA^{*T-t}(s). \tag{8.16}$$

L'intérêt de cette heuristique, contre-intuitive à première vue puisqu'elle signifie résoudre un DEC-POMDP entier sur l'horizon restant, s'explique par le fait que h ne doit

être évalué qu'un nombre limité de fois, mais que sa valeur peut être réutilisée dans les calculs de H dans (8.13).

L'algorithme de recherche heuristique pour les DEC-POMDP à horizon fini est donné dans l'algorithme 8.3. Il est la synthèse d'une recherche dans l'espace des politiques jointes et l'utilisation d'une des trois heuristiques (8.14), (8.15) et (8.16) pour le calcul de H. L'algorithme est à la fois complet et optimal, ce qui peut être souligné par le théorème suivant :

Théorème 1 *(référence) MAA* est complet et optimal.*

PREUVE.– L'algorithme va terminer au pire cas après avoir énuméré tous les vecteurs de politiques possibles et avoir trouvé le meilleur. S'il termine plus tôt, la solution proposée contient une politique jointe dont l'évaluation est supérieure à celle de toutes les feuilles restantes. Puisque l'évaluation de feuilles utilise toujours une heuristique admissible et surestime par conséquent la valeur de toute politique qui n'a pas encore été construite, on peut garantir l'optimalité de la solution retournée. □

Algorithme 8.3 : MAA*

Entrées : Un DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$, un état initial s_0, un horizon T
Initialiser la liste OPEN $D = \times_i \mathcal{A}_i$
$\boldsymbol{q}_{temp} \leftarrow \arg\max_{\boldsymbol{q} \in D} F^T(s_0, \boldsymbol{q})$
répéter
  Choisir $\boldsymbol{q}^* \leftarrow \arg\max_{\boldsymbol{q} \in D} F^T(s_0, \boldsymbol{q})$
  $\boldsymbol{q}^{*'} \leftarrow$ `Développer(`$\boldsymbol{q}^*$`)`
  si $F^T(s_0, \boldsymbol{q}_{temp}) < F^T(s_0, \boldsymbol{q}^{*'})$ **alors**
    $\boldsymbol{q}_{temp} \leftarrow \boldsymbol{q}^{*'}$
    Afficher $\boldsymbol{q}^{*'}$
    pour tous les $\boldsymbol{q} \in D$ **faire**
      si $F^T(s_0, \boldsymbol{q}) \leq F^T(s_0, \boldsymbol{q}^{*'})$ **alors**
        $D \leftarrow D \setminus \{\boldsymbol{q}\}$
  $D \leftarrow D \bigcup \{\boldsymbol{q}^{*'}\}$
  si $\boldsymbol{q}^*$ *complètement développé* **alors**
    $D \leftarrow D \setminus \{\boldsymbol{q}^*\}$
jusqu'à $\exists \boldsymbol{q}^T \in D$ *telle que* $\forall \boldsymbol{q} \in D : F^T(s_0, \boldsymbol{q}) \leq F^T(s_0, \boldsymbol{q}^T) = V(s_0, \boldsymbol{q}^T)$
Sorties : Une politique jointe optimale

8.6.1.3. *Résolution optimale de classes spécifiques de* DEC-POMDP

A partir des travaux de Goldman et Zilberstein sur l'identification de classes de DEC-POMDP de complexité moins importante, Becker et al. ont proposé un algorithme permettant de résoudre certaines de ces classes de problèmes. CSA (*Coverage*

Algorithme 8.4 : Développer

Entrées : Une politique jointe $\boldsymbol{q}^t$ à horizon t
$k_i = |\Omega_i|^t$
Définir une nouvelle politique jointe $\boldsymbol{q}^{t+1} := \boldsymbol{q}^t = \langle q_1, q_2, \ldots, q_n \rangle$
Initialiser l'état de développement de $\boldsymbol{q}^{t+1}$ à zéro
si *p est l'état de développement de la politique jointe $\boldsymbol{q}^t$* **alors**
  $\langle a_1^1, \ldots, a_1^{k_1}, a_2^1, \ldots, a_2^{k_2}, \ldots, a_n^1, \ldots, a_n^{k_n} \rangle_p$ dénote la p'ième sélection d'actions pour les feuilles de $\boldsymbol{q}^{t+1}$
  pour tous les $i = 1$ ***à*** n **faire**
    pour tous les $j = 0$ ***à*** k_i **faire**
      $\lambda(q_i, j) := a_i^j$
  $p \leftarrow p + 1$
Sorties : Une politique jointe $\boldsymbol{q}^{t+1}$ à horizon $(t+1)$

Set Algorithm) permet de résoudre des DEC-MDP à transitions et observations indépendantes [BEC 03, BEC 04b], mais également des DEC-MDP dirigés par les événements [BEC 04a].

Bien que CSA soit présenté dans le cadre de problèmes à deux agents, il est en mesure de traiter des problèmes à n agents ($n \geq 2$). CSA se décompose en trois étapes : la création de MDP augmentés, la recherche d'un ensemble optimal de couverture et la recherche d'une solution. Étant donnés deux agents $\mathcal{A}g_i$ et $\mathcal{A}g_j$, un MDP augmenté représente un problème de décision dans lequel un agent $\mathcal{A}g_i$ doit calculer sa politique optimale, sachant une politique fixe de l'agent $\mathcal{A}g_j$.

La première phase de l'algorithme consiste donc à créer, pour chaque politique de $\mathcal{A}g_j$, un MDP augmenté. La deuxième étape réalisée par CSA calcule la politique optimale de chacun de ces MDP augmentés. Elle permet ainsi d'obtenir l'ensemble des politiques optimales de l'agent $\mathcal{A}g_i$ quelle que soit la politique de l'agent $\mathcal{A}g_j$. L'ensemble de ces politiques est appelé ensemble de couverture optimal. Définir cet ensemble permet de réduire l'espace de recherche des politiques et d'améliorer les performances par rapport à une recherche exhaustive. Enfin, la troisième et dernière étape, réalisée par CSA, cherche la politique optimale jointe dans l'ensemble de couverture optimal. Pour chaque politique de l'agent $\mathcal{A}g_i$ appartenant à cet ensemble, la politique optimale pour $\mathcal{A}g_j$ est recherchée. La politique jointe résultante est évaluée et la meilleure paire trouvée correspond à la politique optimale jointe.

La complexité de cet algorithme est exponentielle en le nombre d'états. Dans le cadre de la résolution des DEC-MDP, la complexité est également doublement exponentielle en nombre de dépendances. Dans le pire cas, CSA a la même complexité que la recherche exhaustive.

Bien que la complexité des sous-classes de problèmes traitées par CSA soit moins importante que celles des DEC-POMDP généraux, il reste encore très difficile de résoudre des problèmes de taille conséquente. Dans le cas des DEC-MDP, CSA ne peut en pratique résoudre que de petits problèmes à 2 agents et très peu de dépendances. A partir du moment où le nombre d'agents ou de dépendances augmente, le temps nécessaire pour trouver la solution devient beaucoup trop important et la solution ne peut être obtenue.

8.6.2. *Algorithmes de résolution approchée*

Les approches que nous venons de présenter recherchent une solution optimale au problème de décision décentralisée. Étant donnée la complexité des DEC-POMDP, plusieurs travaux se sont intéressés à la mise en place d'algorithmes permettant d'obtenir une approximation de la politique optimale. Ainsi, la complexité des algorithmes de résolution a pu être réduite et une plus grande diversité de problèmes ont pu être envisagés.

8.6.2.1. *Limitation de la mémoire*

Afin de pallier les problèmes d'explosion de mémoire dus au nombre exponentiel de politiques à stocker, Bernstein et al. [BER 05] ont proposé un algorithme pour la résolution de DEC-POMDP ne consommant qu'une quantité fixe de mémoire et pouvant traiter des problèmes à horizon fini ou infini. La politique de chaque agent est représentée par un contrôleur stochastique à états finis. Un contrôleur dit de corrélation permet aux agents de coordonner leurs politiques.

L'algorithme fonctionne par amélioration itérative des contrôleurs : à chaque itération un contrôleur est mis à jour de façon à augmenter l'utilité jointe des agents. La convergence vers un optimum global n'est pas garantie. Cependant, il est démontré que la qualité de la solution augmente de façon monotone à chaque itération. L'algorithme atteint par conséquent un optimum local. Les expérimentations montrent que plus le nombre de nœuds de chaque contrôleur augmente, plus la qualité de la solution augmente. En revanche, la quantité de mémoire nécessaire augmente elle aussi.

Chaque itération étant réalisée en un temps polynomial, des problèmes plus grands que ceux traités par les algorithmes de résolution optimale peuvent être envisagés. L'augmentation de la taille des problèmes pouvant être résolus est réalisée au détriment de la qualité de la solution : de plus grands problèmes peuvent être traités mais la solution optimale n'est pas trouvée. La taille des problèmes résolus reste cependant limitée à un petit nombre d'agents puisque chaque itération est exponentielle en fonction du nombre d'agents.

8.6.2.2. *Algorithmes de co-évolution itératifs*

A compléter par Iadine ?

Différents algorithmes itératifs de co-évolution ont été proposés afin de permettre l'obtention d'une politique localement optimale. Le principe de ces algorithmes consiste à améliorer la politique d'un agent en fixant les politiques de tous les autres agents.

Pour un système à n agents, les politiques de $n-1$ agents étant fixées, l'algorithme calcule la politique optimale de l'agent restant. Ce principe est répété itérativement jusqu'à ce qu'il ne soit plus possible d'améliorer aucune des politiques des agents.

Nair et al. [NAI 03] ont décrit un algorithme basé sur ce principe de co-évolution : JESP (*Joint Equilibrium Based Search for Policies*). Ce dernier permet de résoudre des problèmes formalisés sous forme de MTDP et ayant des observations indépendantes.

A chaque itération, la politique d'un agent est modifiée afin d'améliorer la politique jointe. L'algorithme converge vers un optimum local correspondant à un équilibre de Nash. Cet algorithme est en moyenne beaucoup plus rapide que la recherche exhaustive. Dans le pire cas, la complexité est toutefois équivalente à celle de la recherche exhaustive.

Nair et al. ont proposé une amélioration de JESP utilisant les principes de la programmation dynamique : DP-JESP (*Dynamic Programming Joint Equilibrium Based Search for Policies*). DP-JESP est plus rapide que JESP en raison de l'utilisation du principe d'optimalité pour l'évaluation des politiques. Il peut donc traiter des problèmes plus grands que ceux résolus par JESP. Malgré tout, la taille des problèmes traités aussi bien par JESP que par DP-JESP reste limitée. Sur le scénario à deux agents du tigre et du trésor [NAI 03], JESP résout le problème jusqu'à l'horizon 3 et DP-JESP atteint l'horizon 7. Ces problèmes restent assez petits par rapport à des problèmes réels. La complexité de ces algorithmes étant exponentielle, on peut en déduire qu'ils ne sont pas en mesure de traiter des problèmes de grande taille.

L'algorithme LID-JESP (*Locally Interacting Distributed Joint Equilibrium Based Search for Policies*) [NAI 05] utilise l'algorithme JESP et les principes de l'optimisation distribuée de contraintes pour la résolution de DEC-MDP à observations et transitions indépendantes, dans lesquels les interactions entre les agents restent locales. LID-JESP exploite la localité des interactions entre les agents afin d'améliorer les performances de JESP. Chaque agent possède un ensemble de voisins (des agents) avec lesquels il est en interaction. Par définition, tout agent n'appartenant pas au voisinage d'un agent $\mathcal{A}g_i$ n'est pas influencé par les actions de $\mathcal{A}g_i$. La politique de l'agent $\mathcal{A}g_i$ n'influence donc que ses voisins. LID-JESP tire parti de cette topologie des interactions afin de calculer de manière distribuée les politiques des différents voisinages. Les politiques des agents d'un même voisinage sont calculées par un algorithme de co-évolution itératif identique à JESP. LID-JESP permet ainsi un gain de temps dans la résolution de problèmes où chaque agent n'interagit qu'avec un nombre restreint d'autres agents.

Chadès et al. [CHA 02, SCH 02a] ont également décrit un algorithme de co-évolution itératif permettant de résoudre des DEC-POMDP. Contrairement à JESP, aucune contrainte n'est imposée concernant l'indépendance des observations.

Chadès et al. ont tout d'abord proposé deux algorithmes de co-évolution permettant de résoudre les MMDP . Ces algorithmes aboutissent à un optimum local équivalent à un équilibre de Nash.

Une adaptation de ces algorithmes aux DEC-POMDP a ensuite été proposée. Afin de rendre compte de l'observabilité partielle de l'état du système, chaque agent modélise le problème à l'aide d'un MDP-Subjectif. Un MDP-Subjectif est un MDP dans lequel les états sont remplacés par les perceptions locales de l'agent. Ce formalisme est équivalent à un POMDP sans états de croyance, ni historique, et dans lequel on travaillerait directement sur les observations locales des agents. A chaque itération de l'algorithme, un agent, choisi au hasard, reçoit les politiques locales des autres agents. Ces dernières étant supposées fixes, l'agent peut alors construire son MDP-Subjectif et ensuite le résoudre afin de déterminer sa politique optimale.

Un MDP-Subjectif n'étant pas markovien, sa résolution ne permet pas d'obtenir une politique optimale. Dans le cas du contrôle décentralisé, l'algorithme de coévolution proposé par Chadès et al. ne garantit donc pas la convergence vers un équilibre de Nash. La qualité des politiques obtenues n'est garantie que dans le cas du contrôle centralisé ou de l'observabilité totale (on peut alors formaliser le problème par un MMDP).

8.6.2.3. *Apprentissage par descente de gradient*

L'algorithme d'apprentissage par descente de gradient proposé dans [PES 00] s'intéresse également à la résolution de problèmes de décision décentralisée dans des systèmes multi-agents coopératifs. Peshkin et al. proposent un algorithme d'apprentissage distribué permettant à chaque agent d'apprendre individuellement quelle politique exécuter afin de maximiser la récompense globale du système.

Chaque politique est représentée par un contrôleur à états finis. A partir d'un sous-ensemble de politiques jointes, les agents apprennent simultanément leur politique par descente de gradient. Cet apprentissage conduit alors à une solution correspondant à un optimum local.

Contrairement à l'algorithme JESP, cet optimum peut ne pas être un équilibre de Nash. Par ailleurs, les politiques des agents étant représentées par un ensemble de contrôleurs factorisés, certaines politiques jointes ne peuvent pas être représentées et seules les politiques pouvant être factorisées sont envisagées. La solution calculée est donc recherchée dans un sous-ensemble des politiques jointes et correspond à un optimum local dans ce sous-ensemble.

8.6.2.4. *Jeux bayésiens*

Emery-Montemerlo et al. [EME 04] ont proposé un algorithme décentralisé pour la résolution de DEC-POMDP. Cette approche consiste à approcher un DEC-POMDP par une série de jeux bayésiens. La politique du DEC-POMDP est obtenue par concaténation de politiques plus petites résultant de la résolution des jeux bayésiens.

L'algorithme proposé alterne planification et exécution. A chaque étape de planification, un jeu bayésien est résolu par chaque agent afin d'obtenir la politique pour cette étape. Cette politique est ensuite exécutée, puis un nouveau jeu bayésien est résolu et ainsi de suite.

La résolution d'un jeu bayésien nécessite le recours à une heuristique afin de déterminer la fonction d'utilité. Les performances de cette approche dépendent donc du problème traité et de l'heuristique utilisée.

Par ailleurs, lors de la résolution d'un jeu bayésien, il est nécessaire que chaque agent considère toutes les valeurs possibles des variables qu'il ne peut observer (comme par exemple la position des autres agents), ce qui peut conduire à une explosion du nombre des valeurs à prendre en compte.

8.6.2.5. *Approches heuristiques*

Les travaux utilisant des méthodes heuristiques pour le calcul d'une solution approchée se sont essentiellement intéressés à la résolution de DEC-POMDP-COM.

Goldman et Zilberstein [GOL 03] ont présenté une approche de choix glouton pour la résolution du problème de la rencontre d'agents. Deux agents se déplaçant sur une grille doivent se rencontrer le plus tôt possible. Chaque agent ne peut observer la position de l'autre. Les déplacements étant incertains, communiquer permet aux agents d'échanger des informations sur leurs positions respectives et éventuellement de réviser leur point de rencontre en cas de déviation. La communication permet ainsi de limiter le temps nécessaire aux agents pour se rencontrer.

La communication ayant un coût, le problème consiste à déterminer une politique de communication pour chaque agent, lui indiquant quand communiquer. Si la communication est prohibitive, la politique optimale consiste à ne jamais communiquer. Si la communication est gratuite, la politique optimale est de communiquer tout le temps. Dans les autres cas, la politique optimale se situe entre ces deux extrêmes.

Goldman et Zilberstein ont proposé une méthode permettant de calculer une approximation de la politique de communication optimale. Elle correspond à un choix myope des agents qui ne perçoivent pas leur possibilité de communication future : à chaque étape, chaque agent détermine sa politique de communication en supposant qu'il ne lui reste qu'une seule possibilité de communication. Cette méthode contraint les agents à ne communiquer que s'ils en ont vraiment besoin. Elle permet d'obtenir une approximation de la politique optimale et conduit à de bons résultats dans le problème considéré.

Xuan et al. [XUA 01] ont également présenté différentes approches heuristiques pour la résolution du problème de rencontre sous incertitude. Ils ont cependant posé l'hypothèse que les états des agents étaient localement totalement observables.

Aussi bien dans ce travail que dans celui de Goldman et Zilberstein, les heuristiques proposées ne sont évaluées que sur un problème précis. Leurs performances sur d'autres types de problèmes ne sont pas présentées. Ces travaux montrent que la politique de communication optimale se situe quelque part entre les deux politiques extrêmes qui sont « toujours communiquer » et « ne jamais communiquer ». En fonction du problème considéré, l'une ou l'autre des heuristiques va s'avérer la plus appropriée.

8.6.2.6. *Résolution approchée de* DEC-MDP *avec contraintes*

A partir du formalisme OC-DEC-MDP décrit précédemment, Beynier et al. [BEY 05, BEY 06] ont cherché à développer des algorithmes de résolution approchée efficaces qui puissent être utilisés pour résoudre des problèmes de taille importante.

8.6.2.6.1. Coût occasionné et coordination

La décomposition du problème, réalisée lors de la modélisation sous forme d'OC-DEC-MDP, permet d'envisager une résolution locale de chaque MDP. Au lieu de résoudre un DEC-MDP de taille conséquente, le problème est alors considéré comme consistant en la résolution d'un ensemble de MDP. A partir du MDP local d'un agent $\mathcal{A}g_i$, une politique d'actions de l'agent peut ainsi être déduite.

Les contraintes temporelles et de précédence entre les tâches conduisent néanmoins à des dépendances entre les agents et par conséquent entre les MDP locaux. Afin que les agents adoptent un comportement coopératif et coordonné, il est donc nécessaire que les MDP ne soient pas résolus de manière indépendante. Pour ce faire, Beynier et al. ont eu recours à une notion provenant des sciences économiques : le coût occasionné. La théorie du coût occasionné affirme que toute décision a un coût caché et ignorer ce coût peut conduire à des choix erronés. En effet, le coût d'une action ne se traduit pas seulement en fonction des biens dépensés pour l'exécuter, mais également en fonction des bénéfices qui ne pourront être obtenus par la suite car cette action a été réalisée. Cette perte de bénéfices constitue le coût occasionné.

Pour leur part, Beynier et al. font appel à la notion de coût occasionné afin que chaque agent puisse prendre en compte l'impact de ses décisions sur les autres agents. L'influence d'une action d'un agent $\mathcal{A}g_i$ sur un autre agent $\mathcal{A}g_j$ est alors exprimée par le coût occasionné. La meilleure action qu'un agent puisse exécuter à partir d'un état s_i résulte ainsi d'un compromis entre :

– l'utilité espérée de l'agent V' et

– le coût occasionné provoqué sur les autres agents.

Ainsi, la politique de l'agent $\mathcal{A}g_i$ à partir d'un état s_i est calculée par l'équation suivante :

$$\pi_i(s_i) = \arg\max_{a_i \in A_i} \Big(V' - OC(a_i) \Big), \tag{8.17}$$

où A_i désigne l'ensemble des actions a_i de l'agent $\mathcal{A}g_i$, V' désigne l'utilité espérée et $OC(a_i)$ désigne le coût occasionné.

Plus précisément, le coût occasionné correspond à une différence d'utilité. Il mesure la perte en utilité espérée provoquée sur un autre agent lorsque ce dernier est dévié de sa politique optimale. Différentes mesures du coût occasionné ont été décrites. En particulier, Beynier et al. ont introduit la mesure du coût occasionné espéré permettant une évaluation précise de l'influence d'une décision sur les autres agents. Par ailleurs, différentes approximations du coût occasionné espéré ont été présentées.

8.6.2.6.2. Algorithme de révision des politiques

Afin de résoudre les problèmes formalisés sous forme de OC-DEC-MDP, Beynier et al. ont proposé un algorithme de résolution approchée qui tient compte des différentes contraintes du problème. L'algorithme ainsi développé procède par améliorations successives d'un ensemble de politiques initiales, en ordonnançant la révision des politiques d'exécution de chaque tâche. A partir de cet ensemble de politiques initialement fixé, l'algorithme réalise la révision simultanée des politiques de tous les agents. La politique d'exécution de chaque tâche est alors considérée et modifiée de sorte qu'elle corresponde au meilleur compromis entre l'utilité espérée de l'agent exécutant la tâche et le coût occasionné provoqué sur les autres agents en utilisant l'équation (8.17).

Deux versions de cet algorithme de révision des politiques ont été décrites. La version centralisée permet à une unique entité de faire évoluer les politiques de tous les agents. La version décentralisée permet, quant à elle, l'évolution simultanée des politiques des agents. En effet, ces derniers révisent simultanément leur propre politique en utilisant les valeurs de coût occasionné qu'ils communiquent entre eux.

Contrairement aux autres approches existantes, qui sont de complexité exponentielle, l'algorithme de révision des politiques proposé par Beynier et al. est de complexité polynomiale. En raison de cette faible complexité, des problèmes de taille importante peuvent être traités. Les résultats expérimentaux ont ainsi montré que des missions composées de plusieurs centaines de tâches et d'agents peuvent être résolues. Cette augmentation significative de la taille des problèmes traités (par rapport aux approches existantes) est rendue possible grâce à l'exploitation des caractéristiques du problème et à la recherche d'une solution approchée. Cette approche rejoint les idées exposées par Nair et al. [NAI 05] afin de développer une approche efficace pour la résolution des ND-POMDP. En effet, ils suggèrent d'exploiter les caractéristiques du problème, dans ce cas la localité des interactions, afin de développer un algorithme calculant une politique approchée de la solution optimale.

En ce qui concerne la qualité des solutions obtenues par l'algorithme de révision des politiques, il a été prouvé que certaines propriétés des problèmes (ressources illimitées par exemple) garantissent l'obtention d'une solution optimale. Dans le cas général, une solution approchée est déterminée.

8.6.2.6.3. Itération de la révision des politiques

Afin d'améliorer la qualité des politiques calculées par le précédent algorithme de révision des politiques, une version itérative de l'algorithme a été proposée [BEY 06]. Elle consiste en l'exécution répétée de l'algorithme de révision des politiques et permet de réviser plusieurs fois la politique d'exécution de chaque tâche. A chaque itération, le nouvel ensemble des politiques calculé est utilisé comme ensemble des politiques initiales de l'itération suivante. Ainsi, à l'itération N, l'ensemble des politiques initiales considéré correspond à l'ensemble résultant de la $N - 1^{\text{ème}}$ itération. Le processus de révision des politiques est ainsi répété jusqu'à ce qu'aucun changement ne soit plus possible, c'est-à-dire que l'ensemble des politiques initiales et l'ensemble des politiques finales d'une même itération soient identiques.

La complexité de cette version itérative dépend de la complexité de l'algorithme de révision des politiques (polynomiale) et du nombre d'itérations. Les garanties de convergence de l'algorithme itératif dépendent de la méthode d'estimation du coût occasionné utilisée. Lorsqu'il est fait appel au coût occasionné espéré, la convergence de l'algorithme est garantie. Les résultats expérimentaux montrent qu'un tel point de convergence est, dans la majeure partie des cas, atteint en moins de quatre itérations. La version itérative de l'algorithme de révision des politiques est donc également en mesure de résoudre des problèmes de taille importante. Il a par ailleurs été mis en évidence que les propriétés du problème garantissant l'obtention d'une politique optimale dans le cas de l'algorithme de révision, garantissaient également l'obtention d'une politique optimale lors de l'exécution de l'algorithme itératif. Dans le cas général, il a été prouvé que la politique jointe calculée correspond à une situation d'équilibre. En effet, l'algorithme itératif proposé s'arrête lorsque la politique d'exécution de chaque tâche ne peut plus être améliorée. Ainsi, étant données les politiques des autres agents, aucun agent n'est en mesure de modifier sa politique de sorte que les performances du système augmentent.

8.7. Quelques exemples d'application

8.7.1. *Robotique mobile exploratoire*

Afin de démontré leur applicabilité à des problèmes réels, les travaux développés par Beynier et al. [BEY 05, BEY 06] ont été appliqués à des problèmes de décision décentralisée en robotique collective. Pour ce faire, Beynier et al. se sont inspirés de scénarios de la robotique exploratoire visant à résoudre des problèmes de planification similaires à ceux rencontrés par les robots envoyés sur Mars.

Ils ont ainsi, entre autres, considéré un ensemble de deux robots devant visiter un certain nombre de sites afin d'y réaliser des photographies et/ou des analyses du sol. L'un des avantages lié au développement de colonies de robots autonomes est de permettre la spécialisation des facultés de chacun. Le premier robot considéré est ainsi spécialisé dans la prise de photos alors que le second est équipé d'une foreuse lui permettant de prélever des échantillons du sol. Il dispose également des appareils de mesure nécessaires afin d'analyser ces prélèvements.

Les robots doivent explorer un ensemble de huit sites pour lesquels il est nécessaire de prendre une photo, d'analyser le sol ou bien de réaliser ces deux tâches. Les prélèvements risquant néanmoins de modifier la topologie des sites, il est nécessaire de les réaliser après la prise de photos. De plus, lorsque le premier robot réalise les photographies, il est impératif qu'aucun autre robot ne soit présent sur le site afin de ne pas risquer de masquer une partie de l'image. A partir d'une zone de départ donnée, le robot 1 doit alors prendre des photos des sites A, B, D, E, F, H et J. Quant à lui, le robot 2 doit réaliser des analyses sur les sites C, D, F, H et I. Les sites considérés par un même robot sont ordonnés de sorte de minimiser les déplacements et par conséquent les consommations de ressources. La figure 8.6 présente le déroulement

de la mission. Chacun des robots devant visiter les sites D, F et H, des contraintes de précédence doivent être établies entre les robots. Ainsi le robot 2 peut pénétrer sur le site D si et seulement si le robot 1 l'a quitté. Il en va de même pour les sites F et H.

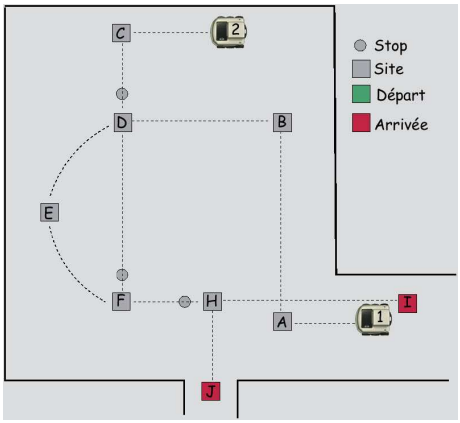


Figure 8.6. *Mission envisagée*

La mission considérée a tout d'abord été représentée sous forme d'un graphe orienté acyclique. Pour chaque tâche de la mission, les données la caractérisant ont été renseignées (agent, durées d'exécution, consommations de ressources, etc.). A partir du graphe ainsi défini, la modélisation du problème sous forme d'OC-DEC-MDP a été automatiquement générée. L'algorithme itératif de révision des politiques a ensuite permis de calculer les politiques des agents. Une fois les politiques déterminées, leur application correspond au parcours d'un automate à états finis, les capacités de calcul et la quantité de mémoire nécessaires à chaque robot restent donc limitées. Le processus délibératif proposé par cette approche fait ainsi preuve d'efficacité et de légèreté.

Ces politiques ont été implémentées sur des robots Koala qui ont ainsi été en mesure de mener à bien la mission d'exploration décrite par la figure 8.6. La figure 8.7 présente le déroulement de la mission au niveau du site D (matérialisé par une croix noire sur le sol). Le robot 2 (dans la partie supérieure de l'image) arrive le premier à proximité du site. Anticipant le comportement du robot 1 et par conséquent le fait que celui-ci n'ait pas encore exploré le site D, le robot 2 attend pour pénétrer sur le site. Nous voyons alors le robot 1 entrer sur le site D, y réaliser sa tâche (image 2) et en repartir (images 3 et 4). Le robot 2 n'observant pas les actions entreprises par le robot 1, il ne peut constater la présence de ce dernier sur le site que s'il tente d'y entrer. Le robot 2 essaie donc ensuite d'entrer sur le site (image 5). Si les contraintes de précédence sont respectées, il achève son déplacement (image 6). En cas d'échec partiel, le robot 2 revient à sa position précédente et attend jusqu'à ce que sa politique lui dicte d'essayer à nouveau d'entrer sur le site. Une configuration similaire est observable pour les croisements des sites F et H. Ces expérimentations ont permis de

démontrer les possibilités d'exploitation, par des robots réels, des politiques calculées par cette approche. Elles ont également montré que les politiques ainsi calculées permettaient aux robots de mener à bien leur mission tout en respectant les contraintes sur l'exécution des tâches et sans nécessiter d'envois de messages entre les robots.

Des expérimentations en simulation [BEY 05, BEY 06] ont également été menées sur des scénarios plus importants afin de tester la taille des missions pouvant être envisagées. Elles ont démontré que des problèmes d'exploration multi-robots composés d'une dizaine d'agents et de plusieurs centaines de tâches pouvaient être résolus (des expérimentations ont été réalisées jusqu'à 2 agents et 800 tâches).

8.8. Conclusion et perspectives

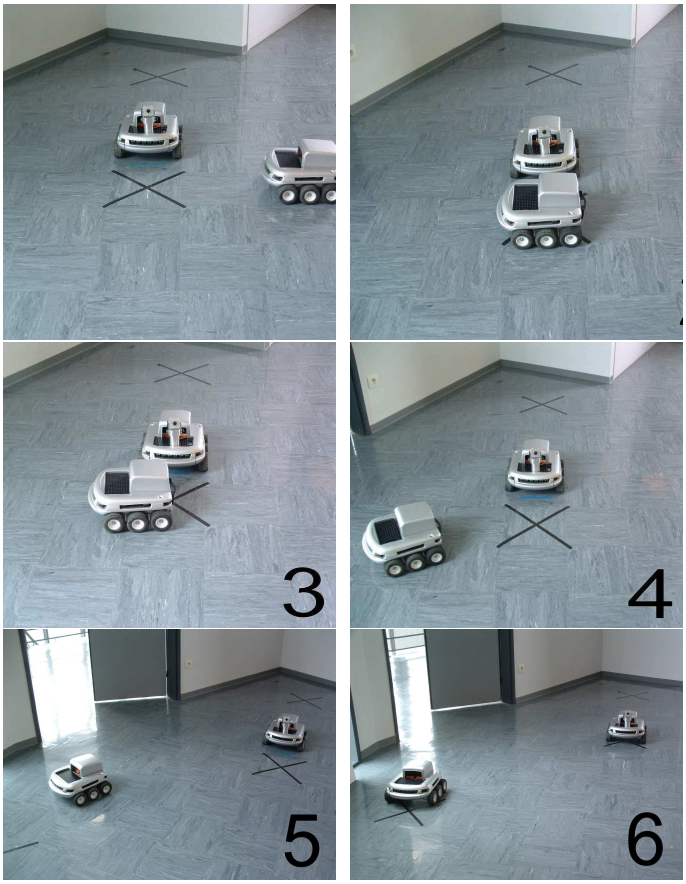


Figure 8.7. *Exécution de la mission (croisement sur le site D)*

Chapitre 9

Représentations factorisées

9.1. Introduction

L'ensemble des solutions décrites dans le cadre des MDP (chapitre 1), que ce soit pour la planification ou l'apprentissage par renforcement, partagent toutes un inconvénient commun : elles ne sont pas adaptées à la résolution de problèmes de grande taille. En effet, l'utilisation de représentations non structurées nécessite une énumération explicite de l'ensemble des états possibles du problème pour représenter les fonctions nécessaires à sa résolution.

EXEMPLE.– Dans le cas de la voiture qu'il faut entretenir (voir tome 1, section 1.1), on conçoit rapidement que l'ensemble des états possibles d'une voiture peut devenir gigantesque. Par exemple, on peut tenir compte de l'état d'usure de chacune des pièces de la voiture. L'idée sous-tendant ce chapitre est que la voiture est constituée d'une collection de sous-systèmes plus ou moins indépendants. Par exemple, faire une vidange ne devrait pas, en principe, influencer l'état des freins de la voiture. Il semble alors possible de prendre en compte la relative indépendance de ces sous-systèmes pour décrire plus efficacement le modèle du problème dans l'espoir que la recherche d'une solution sera plus simple. Peut-être apprendrons-nous ainsi qu'il est toujours optimal de remplacer des freins usés, quel que soit l'état du reste de la voiture...

Ce chapitre vise donc à décrire une extension des MDP présentée par [BOU 95, BOU 99a], appelée *processus décisionnels de Markov factorisés* (*Factored Markov Decision Processes* (FMDP)) et permettant de représenter les fonctions de transition et de récompense d'un problème de façon compacte (section 9.2). Une fois le problème représenté de façon compacte, nous décrirons plusieurs méthodes de planification permettant de trouver les solutions optimales ou optimales approchées (section 9.3), tout

---

Chapitre rédigé par Thomas DEGRIS et Olivier SIGAUD.

en exploitant la structure du problème afin d'éviter une énumération explicite de l'espace d'état. Nous conclurons section 9.4.

9.2. Le formalisme des FMDP

9.2.1. *Représentation de l'espace d'état*

Il est souvent naturel de décrire un problème par un ensemble de paramètres pouvant prendre différentes valeurs décrivant l'état courant du système. Ainsi, l'ensemble des états possibles peut être caractérisé par un ensemble de variables aléatoires. En pratique, l'ensemble des états possibles S est décrit par un ensemble de variables aléatoires $X = \{X_1, \ldots, X_n\}$ où chaque variable X_i peut prendre différentes valeurs dans son domaine $\underline{\text{Dom}}(X_i)$. Un état est donc une instanciation de X décrite sous la forme d'un vecteur $x = \{x_1, \ldots, x_n\}$ de valeurs x_i avec $\forall i\ x_i \in \underline{\text{Dom}}(X_i)$. De plus, on utilise comme raccourci d'écriture $\underline{\text{Dom}}(X)$ pour décrire l'ensemble des instanciations possibles des variables $X_i \in X$. L'espace d'état S du MDP est donc $S = \underline{\text{Dom}}(X)$.

L'avantage d'une telle représentation est qu'il est possible d'exploiter différentes structures existantes dans un problème pour, d'une part, le représenter de façon compacte, d'autre part le résoudre en limitant la complexité de la solution et de la méthode utilisée pour l'obtenir. Étant donnée une telle décomposition, les principales contributions du cadre mathématique des FMDP sont de *décomposer* les fonctions de transition et de récompense (respectivement de façon multiplicative et additive) afin d'exploiter *les indépendances relatives aux fonctions* liées à la structure du problème. De plus, les FMDP offrent un cadre approprié à l'utilisation, de façon complémentaire mais pas obligatoire, de deux autres propriétés liées à la structure d'un problème : *les indépendances relatives aux contextes* et *l'approximation additive*. Dans la suite, nous parlerons d'« indépendances fonctionnelles » (respectivement « contextuelles ») pour désigner les indépendances relative aux fonctions (respectivement aux contextes).

Afin d'illustrer le cadre des FMDP, nous utiliserons l'exemple du *Coffee Robot* proposé par [BOU 00a] bien connu dans la littérature des FMDP. Une fois l'exemple *Coffee Robot* décrit (section 9.2.2), la section 9.2.3 décrit les décompositions des fonctions de transition et de récompense ainsi que la formalisation des indépendances fonctionnelles. La section 9.2.4 propose une formalisation du concept d'indépendance contextuelles. Enfin, l'approximation additive sera étudiée plus tard, dans le contexte de son utilisation lors de la section 9.3.3.5.

9.2.2. *L'exemple* Coffee Robot

Un robot doit aller acheter un café pour sa propriétaire restant au bureau. Quand il pleut, comme le robot doit sortir pour aller chercher le café, il doit se munir d'un parapluie lorsqu'il est au bureau, sinon il sera mouillé. Pour décrire l'état du système,

six variables aléatoires binaires[1] ($\underline{\text{Dom}}(X_i) = \{0, 1\}$ correspondant respectivement à `Faux` et `Vrai`) sont utilisées :

$\mathcal{HOC}$: la propriétaire a-t-elle un café ? *(Has Owner Coffee ?)*

$\mathcal{HRC}$: le robot a-t-il un café ? *(Has Robot Coffee ?)*

$\mathcal{W}$: le robot est-il mouillé ? *(Wet ?)*

$\mathcal{R}$: est-ce qu'il pleut ? *(Raining ?)*

$\mathcal{U}$: le robot a-t-il un parapluie ? *(Umbrella ?)*

$\mathcal{O}$: le robot est-il au bureau ? *(Office ?)*

Par exemple, le vecteur $[\mathcal{HOC}{=}0,\mathcal{HRC}{=}1,\mathcal{W}{=}0,\mathcal{R}{=}1,\mathcal{U}{=}0,\mathcal{O}{=}1]$ représente un état de ce problème dans lequel la propriétaire n'a pas de café, le robot a un café, le robot n'est pas mouillé, il pleut, le robot n'a pas de parapluie et le robot est au bureau. Ce problème étant composé de 6 variables binaires, son espace d'états contient $2^6 = 64$ états possibles.

Dans le problème *Coffee Robot*, le robot dispose de quatre actions possibles :

$\mathcal{G}$o : se déplacer vers le lieu opposé ;

$\mathcal{B}$uyC : acheter un café, que le robot obtient s'il est au café ; *(Buy Coffee)*

$\mathcal{D}$elC : donner le café à sa propriétaire, qu'elle peut obtenir si le robot est au bureau et qu'il a un café ; *(Deliver Coffee)*

$\mathcal{G}$etU : prendre un parapluie, que le robot peut obtenir s'il est au bureau. *(Get Umbrella)*

L'effet de ces actions peut être bruité afin de représenter les cas stochastiques. Par exemple, lorsque le robot donne la tasse de café à sa propriétaire, la propriétaire obtiendra son café avec une certaine probabilité. L'action peut mal se passer, par exemple, lorsque le robot renverse le café. Ainsi, lorsque le robot exécute l'action $\mathcal{D}$elC dans l'état $s = [\mathcal{HOC}{=}0,\mathcal{HRC}{=}1,\mathcal{W}{=}0,\mathcal{R}{=}1,\mathcal{U}{=}0,\mathcal{O}{=}1]$ (le robot a un café et est au bureau, la propriétaire n'a pas de café), la fonction de transition définit :

– $P([\mathcal{HOC}{=}1,\mathcal{HRC}{=}1,\mathcal{W}{=}0,\mathcal{R}{=}1,\mathcal{U}{=}0,\mathcal{O}{=}1]|s, \mathcal{D}\text{elC}) = 0.8$,

– $P([\mathcal{HOC}{=}0,\mathcal{HRC}{=}1,\mathcal{W}{=}0,\mathcal{R}{=}1,\mathcal{U}{=}0,\mathcal{O}{=}1]|s, \mathcal{D}\text{elC}) = 0.2$,

1. Principalement pour des raisons de simplicité d'exposition, la plupart des exemples décrits dans ce chapitre utilisent des variables binaires. Cependant, rien ne limite l'utilisation des méthodes exposées à des problèmes contenant des variables énumérées.

– 0.0 pour les autres probabilités.

Enfin, le robot reçoit une récompense de 0.9 lorsque la propriétaire a un café (0 lorsqu'elle n'a pas de café) ajoutée à 0.1 lorsqu'il est sec (et 0 lorsqu'il est mouillé). La récompense obtenue lorsque la propriétaire a un café est supérieure à la récompense obtenue par le robot lorsqu'il reste sec pour indiquer que le premier objectif est prioritaire sur le deuxième. Dans cet exemple, la fonction de récompense ne dépend pas de l'action réalisée par le robot.

9.2.3. *Décomposition et indépendances relatives aux fonctions*

Les indépendances relatives aux fonctions expriment le fait que certaines définitions du problème ne dépendent pas systématiquement de toutes les autres variables du problème ou de l'action réalisée par l'agent. Par exemple, dans le problème *Coffee Robot*, la valeur de la variable $\mathcal{R}$ au prochain pas de temps —indiquant s'il pleut ou non— ne dépend que de sa propre valeur au pas de temps courant. En effet, le fait qu'il va pleuvoir au prochain pas de temps est indépendant des variables telles que « le robot a-t-il un café ? » (variable $\mathcal{HRC}$) ou de l'action exécutée par l'agent. Le cadre des FMDP permet d'exploiter cette propriété principalement dans la description des fonctions de transition et de récompense du problème et dans l'utilisation de ces fonctions par les algorithmes de planification. Cette notion est formalisée par deux opérateurs, Parents et Scope, qui sont définis respectivement dans le cadre de la représentation de la fonction de transition (section 9.2.3.1) et de la fonction de récompense (section 9.2.3.4).

9.2.3.1. *Décomposition de la fonction de transition*

En supposant que l'ensemble des états possibles se décompose en un ensemble de variables aléatoires (décrit section 9.2.1), il est possible de décomposer une probabilité $P(s'|s)$ en un produit de probabilités, puis d'exploiter les indépendances entre ces variables aléatoires afin de rendre plus compacte la description de la fonction de transition.

Par exemple, admettons qu'un espace d'état soit décrit avec trois variables binaires X, Y et Z. Pour énumérer l'ensemble des combinaisons possibles $P(s'|s)$, il est nécessaire de décrire une table contenant $2^{2\cdot 3} = 64$ entrées. En décomposant la probabilité $P(s'|s)$ en un produit de probabilités, on obtient :

$$\begin{aligned} P(s'|s) &= P(x', y', z'|s) \\ &= P(x'|s)P(y'|s, x')P(z'|s, x', y') \end{aligned}$$

avec x représentant la valeur de la variable X au pas de temps t et, x' la valeur de la variable X au pas de temps $t + 1$. De plus, si les relations de dépendance entre les variables sont connues, alors $P(s'|s)$ peut s'écrire de façon plus compacte. Par exemple, si chacune des variables X, Y et Z ne dépend que de sa valeur dans l'état

précédent sauf la variable Y qui dépend aussi de X dans l'état précédent, alors :

$$\begin{aligned} P(s'|s) &= P(x'|s)P(y'|s,X')P(z'|s,x',y') \\ &= P(x'|x)P(y'|y,x)P(z'|z) \end{aligned}$$

En agrégeant les états pour lesquels la fonction de transition est identique, seules $2^1 + 2^2 + 2^1 = 8$ entrées sont nécessaires et réparties en trois tables différentes, une pour chaque variable (correspondant respectivement à la description des distributions de probabilité $P(X'|X)$, $P(Y'|Y,X)$ et $P(Z'|Z)$).

Ainsi, les indépendances fonctionnelles liées à la structure du problème sont mises en évidence et permettent ainsi d'agréger certaines régularités dans la description de la fonction de transition. De plus, elles correspondent à une représentation intuitive consistant à décrire l'effet de chaque action sur la valeur de chacune des variables du problème. Cette représentation de la fonction de transition est formalisée en utilisant le cadre des réseaux bayésiens dynamiques [BOU 95].

9.2.3.2. *Les réseaux bayésiens dynamiques*

Les réseaux bayésiens [PEA 88] sont un formalisme permettant de représenter graphiquement des dépendances (ou indépendances) entre des variables. Les variables constituent les nœuds d'un graphe orienté, les relations directes de dépendance probabiliste entre deux variables sont représentées par un arc entre les deux nœuds représentant chacun des variables. Les réseaux bayésiens dynamiques [DEA 89] (*Dynamic Bayesian Networks (*DBN*)*) sont des réseaux bayésiens permettant de représenter les données temporelles engendrées par des processus stochastiques.

En faisant l'hypothèse que le problème observé est stationnaire (donc la fonction de transition T du MDP ne varie pas au cours du temps), il est possible de représenter T avec des DBN faisant seulement apparaître deux pas de temps successifs. Dans ce cas, les DBN sont composés de deux ensembles de nœuds :

1) l'ensemble de nœuds représentant l'ensemble des variables de l'espace d'état à l'instant t ;

2) l'ensemble de nœuds représentant l'ensemble des variables de l'espace d'état à l'instant $t+1$.

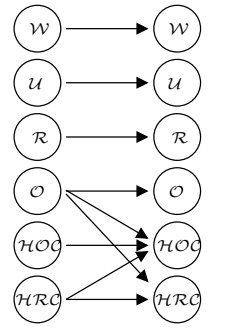
Les arcs indiquent alors les dépendances directes entre les variables à l'instant t et les variables à l'instant $t+1$ ou encore des dépendances entre les variables à l'instant $t+1$ (ces arcs sont appelés arcs synchrones). Dans ce cas particulier, un DBN est quelque fois appelé *2 Time Bayesian Network*. Pour une question de clarté, nous ferons l'hypothèse que les arcs synchrones ne sont pas nécessaires pour décrire le modèle des transitions du problème.

Il est alors possible de représenter graphiquement les dépendances de la fonction de transition en utilisant un DBN par variable et par action. L'action exécutée par l'agent peut aussi être considérée comme une variable à l'instant t. Dans ce cas, un

seul DBN par variable suffit [BOU 96b]. Enfin, pour être complet, un DBN doit être quantifié, comme illustré dans la section suivante pour l'exemple *Coffee Robot*.

9.2.3.3. *Modèle factorisé de la fonction de transition*

La figure 9.1 montre la représentation de l'effet de l'action $\mathcal{D}\mathrm{elC}$ sur l'ensemble des états. Le DBN (figure 9.1(a)) permet de constater facilement que, pour l'action $\mathcal{D}\mathrm{elC}$, la variable $\mathcal{HOC}$ ne dépend que des variables $\mathcal{O}$, $\mathcal{HRC}$ et $\mathcal{HOC}$ au pas de temps précédent et est indépendante des autres variables du problème. On définit $\underline{\text{Parents}}_\tau(X'_i)$ l'ensemble des parents de la variable X'_i dans ce DBN τ. Cet ensemble peut être partitionné en deux sous-ensembles $\underline{\text{Parents}}^t_\tau(X'_i)$ et $\underline{\text{Parents}}^{t+1}_\tau(X'_i)$ représentant respectivement l'ensemble des parents au temps t et l'ensemble des parents au temps $t+1$. Nous supposons l'absence d'arc synchrone, donc $\underline{\text{Parents}}^{t+1}_\tau(X'_i) = \emptyset$ et $\underline{\text{Parents}}_\tau(X'_i) = \underline{\text{Parents}}^t_\tau(X'_i)$. Dans l'exemple de la figure 9.1, nous avons $\underline{\text{Parents}}_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}') = \{\mathcal{O}, \mathcal{HRC}, \mathcal{HOC}\}$.



(a)

$\mathcal{HOC}$	$\mathcal{HRC}$	$\mathcal{O}$	$\mathcal{HOC}'$
1	1	1	1.0
1	1	0	1.0
1	0	1	1.0
1	0	0	1.0
0	1	1	0.8
0	1	0	0.0
0	0	1	0.0
0	0	0	0.0

(b)

Figure 9.1. *Représentation (partielle) de la fonction de transition T pour le problème* Coffee Robot*. La figure (a) représente les dépendances entre les variables pour l'action $\mathcal{D}\mathrm{elC}$ sous la forme d'un* DBN*. La figure (b) définit la distribution de probabilité conditionnelle $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{O}, \mathcal{HRC}, \mathcal{HOC})$ sous forme tabulaire.*

Afin de quantifier l'effet d'une action sur l'espace d'états, on spécifie la probabilité $P_\tau(X'_i|x)$ pour chaque instanciation possible $x \in \underline{\text{Dom}}(\underline{\text{Parents}}_\tau(X'_i))$. Chaque réseau d'action DBN τ est donc quantifié par un ensemble de *distributions de probabilités conditionnelles*[2]. On note $P_\tau(X'_i|\underline{\text{Parents}}_\tau(X'_i))$ une telle distribution pour une variable X'_i. Une probabilité $P_\tau(s'|s)$ de la fonction de transition peut alors être

2. *Conditional Probability Distributions*

définie de façon compacte :

$$P_\tau(s'|s) = \prod_i P_\tau(x_i^{s'}|x^s) \tag{9.1}$$

avec $x_i^{s'}$ l'instanciation de la variable X'_i dans l'état s' et x^s l'instanciation des variables appartenant à $\underline{\text{Parents}}_\tau(X'_i)$.

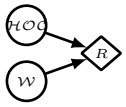
La figure 9.1(b) représente sous forme tabulaire et pour l'action $\mathcal{D}\text{elC}$ la distribution de probabilité conditionnelle $P_{\mathcal{D}\text{elC}}(\mathcal{HOC}'|\mathcal{O}, \mathcal{HRC}, \mathcal{HOC})$ dans le problème *Coffee Robot*. Les colonnes $\mathcal{O}$, $\mathcal{HRC}$ et $\mathcal{HOC}$ représentent la valeur de ces variables à l'instant t, la colonne $\mathcal{HOC}'$ représente la probabilité pour la variable $\mathcal{HOC}$ d'avoir la valeur `Vrai` au temps $t+1$.

La décomposition multiplicative et l'exploitation des indépendances fonctionnelles dans la description du modèle des transitions et dans le calcul des probabilités qui en découlent sont les principales contributions des FMDP par rapport aux MDP. Ces deux propriétés sont exploitées par l'ensemble des algorithmes décrits dans le cadre des FMDP.

9.2.3.4. *Modèle factorisé de la fonction de récompense*

Pour spécifier complètement un MDP, il est nécessaire de décrire la fonction R de récompense. Une représentation similaire à la description de la fonction de transition peut être utilisée pour le cadre des FMDP. En effet, la fonction de récompense d'un MDP peut, d'une part, être décomposée de façon additive et, d'autre part, ne dépend pas nécessairement de toutes les variables d'état du problème.

Par exemple, dans le problème *Coffee Robot*, la fonction de récompense, représentée par un losange dans la figure 9.2, ne dépend que des variables $\mathcal{HOC}$ et $\mathcal{W}$ et elle est indépendante des actions réalisées par le robot ou bien des autres variables du problème.



(a)

$\mathcal{HOC}$	$\mathcal{W}$	R
1	0	1.0
1	1	0.9
0	0	0.1
0	1	0.0

(b)

$\mathcal{HOC}$	R_0
0	0.0
1	0.9

+

$\mathcal{W}$	R_1
0	0.1
1	0.0

(c)

Figure 9.2. *Représentation de la fonction de récompense $R(s)$*

La table de la figure 9.2(b) spécifie que le meilleur état pour le robot est lorsque sa propriétaire a un café et que le robot est sec tandis que le pire cas est lorsque sa propriétaire n'a pas de café et que le robot est mouillé. On observe la préférence donnée au cas où l'utilisateur possède un café et le robot est mouillé par rapport au cas où l'utilisateur n'a pas de café et le robot est sec.

[BOU 00a] définissent la fonction de récompense du problème *Coffee Robot* en faisant la somme des deux critères du problème, « la propriétaire a un café » et « le robot est mouillé ». Pourtant, ces deux critères sont indépendants. Afin de profiter de la décomposition additive de cette fonction de récompense, [GUE 03b] proposent de formaliser la fonction de récompense d'un FMDP en une somme de plusieurs *fonctions de récompense localisées* [3].

Pour le problème *Coffee Robot*, on peut définir la fonction de récompense comme la somme de deux fonctions de récompenses localisées : la première associée à la variable $\mathcal{HOC}$ et la deuxième associée à la variable $\mathcal{W}$ et représentant respectivement les deux critères « la propriétaire a un café » et « le robot est mouillé ».

[GUE 03b] formalisent cette notion en définissant tout d'abord la notion de *scope* d'une fonction f localisée (notée $\underline{\text{Scope}}(f)$), que nous traduirons par « portée ». La portée d'une fonction f localisée est définie tel que :

Définition 29 (scope) *Soit une fonction $f : \{X_1, \ldots, X_n\} \mapsto \mathbb{R}$. $\underline{Scope}(f) = C$ définit la portée de f si et seulement si f ne dépend que des variables de C, les valeurs des autres variables étant indifférentes. On assimilera alors f à sa projection sur $\underline{Dom}(C)$: $f : \underline{Dom}(C) \mapsto \mathbb{R}$ avec $C \subseteq \{X_1, \ldots, X_n\}$.*

Soit une fonction f telle que $\underline{\text{Scope}}(f) = C$ avec $C \subseteq X$, on utilise la notation $f(x)$ comme raccourci pour noter $f(x[C])$ avec $x[C]$ représentant l'instanciation des variables appartenant à C dans l'instanciation x. La portée d'une fonction f permet ainsi de mettre en évidence les indépendances relatives à f [4].

Il est maintenant possible de définir le concept de fonction de récompense localisée. Soit un ensemble de fonctions localisées $R_1^a, \ldots, R_r^a$ avec la portée de chaque fonction R_i^a restreinte à un sous-ensemble $C_i^a \subseteq \{X_1, \ldots, X_n\}$. La récompense associée au fait d'exécuter l'action a dans un état s est alors définie telle que :

$$R^a(s) = \sum_{i=1}^{r} R_i^a(s[C_i^a]) \tag{9.2}$$

$$= \sum_{i=1}^{r} R_i^a(s). \tag{9.3}$$

Ainsi, pour reprendre l'exemple de *Coffee Robot*, le problème est défini par deux fonctions de récompenses R_1 et R_2 définies dans la figure 9.2(c) et correspondant respectivement aux deux critères « la propriétaire a un café » et « le robot est mouillé ». On a

3. *localized reward functions*

4. La notion de portée d'une fonction est similaire à la notion de parent utilisée pour la définition des distributions de probabilité conditionnelles de la fonction de transition.

$\underline{\text{Scope}}(R_1) = \{\mathcal{HOC}\}$ et $\underline{\text{Scope}}(R_2) = \{\mathcal{W}\}$. On utilise $R_1(s)$ comme raccourci pour représenter $R_1(s[\mathcal{HOC}])$, avec $s[\mathcal{HOC}]$ représentant l'instanciation de $\mathcal{HOC}$ dans s.

Bien que l'ensemble des algorithmes décrits dans le cadre des FMDP exploitent les indépendances relatives aux fonctions de récompense du problème, tous n'exploitent pas la décomposition additive de la fonction de récompense. De plus, tous les problèmes ne présentent pas une telle décomposition.

9.2.4. *Indépendances relatives aux contextes*

Les indépendances relatives aux contextes concernent le fait que, pour représenter une fonction du problème à résoudre (quelle que soit la fonction), il n'est pas obligatoire de tester systématiquement l'ensemble des variables nécessaires à la représentation de cette fonction. Un contexte se définit de la façon suivante :

Définition 30 (Contexte) *Soit une fonction $f : \{X_0, \ldots, X_n\} \rightarrow Y$. Un contexte $c \in \underline{Dom}(C)$ est une instanciation d'un sous-ensemble de variables $C = \{C_0, \ldots, C_j\}$ tel que $C \subseteq \{X_0, ..., X_n\}$. Un contexte est noté $(C_0 = c_0) \wedge \ldots \wedge (C_j = c_j)$ ou $C_0 = c_0 \wedge \ldots \wedge C_j = c_j$.*

Par exemple, la description de la politique optimale dans le problème *Coffee Robot* nécessite l'utilisation de toutes les variables du problème. Cependant, dans le contexte $\mathcal{HOC} = 0 \wedge \mathcal{HRC} = 1 \wedge \mathcal{O} = 1$, c'est-à-dire lorsque la propriétaire n'a pas de café et que le robot a un café et qu'il est au bureau, il est possible de déterminer l'action optimale (l'action $\mathcal{D}$elC dans ce cas) sans avoir à tester d'autres variables telles que « est-ce qu'il pleut ? » ou « le robot est-il mouillé ? ».

Contrairement aux indépendances fonctionnelles, l'exploitation des indépendances contextuelles est étroitement liée aux structures de données employées pour représenter les fonctions du problème. En effet, les opérateurs, $\underline{\text{Parents}}$ et $\underline{\text{Scope}}$, qui s'appliquent aux indépendances fonctionnelles définissent le nombre de variables dont une fonction dépend. Comme nous l'avons constaté dans la section 9.2.3 (par exemple dans la figure 9.1), la spécification des indépendances fonctionnelles permet de représenter ces fonctions de façon plus compacte, même lorsque la structure de données utilisée pour leur représentation n'est pas structurée, comme c'est le cas pour les représentations tabulaires.

Pour un ensemble de variables donné (spécifiées par les opérateurs $\underline{\text{Parents}}$ et $\underline{\text{Scope}}$), les indépendances contextuelles sont exploitées pour représenter des fonctions de façon plus compacte. Pour ces indépendances, la principale technique est d'utiliser des représentations structurées permettant d'aggréger des états, contrairement à une représentation tabulaire.

Ainsi, chaque algorithme utilisant une structure de données différente, nous avons préféré illustrer ce concept dans la section suivante, c'est-à-dire dans le cadre de l'algorithme qui l'exploite et des structures de données utilisées.

9.3. Planification dans les FMDP

La section suivante présente un certain nombre de méthodes de planification dans les FMDP. Plutôt que de décrire les algorithmes en détails, nous nous sommes attachés à décrire les différentes représentations utilisées par ceux-ci, afin que le lecteur puisse distinguer rapidement les principales différences et caractéristiques de ces algorithmes. Cependant, nous donnerons l'ensemble des références nécessaires pour que le lecteur puisse obtenir une description exhaustive de ces algorithmes s'il le désire.

9.3.1. *Itérations structurées sur les valeurs et sur les politiques*

Les deux algorithmes d'itération structurée sur les valeurs et sur les politiques, *Structured Value Iteration* (SVI) et *Structured Policy Iteration* (SPI) [BOU 00a] ont été les premiers algorithmes adaptant les algorithmes de programmation dynamique au formalisme des FMDP, illustrant ainsi les avantages (et les inconvénients) de ce formalisme. En plus des indépendances spécifiques aux fonctions utilisées dans la décomposition des fonctions de transition et de récompense, les algorithmes SPI et SVI utilisent une représentation structurée afin d'exploiter les indépendances contextuelles dans la représentation des différentes fonctions du problème.

Par exemple, nous pouvons constater que, dans l'exemple *Coffee Robot*, lorsque la propriétaire a déjà un café, alors il n'est pas nécessaire d'évaluer si le robot a un café ou s'il est au bureau pour déterminer si la propriétaire aura un café au prochain pas de temps. Ainsi, la distribution de probabilité de la variable aléatoire $\mathcal{HOC}'$ dans le contexte $\mathcal{HOC} = 1$, c'est-à-dire « la propriétaire a un café », est indépendante des variables $\mathcal{HRC}$ et $\mathcal{O}$ au pas de temps précédent, c'est-à-dire « le robot a-t-il un café ? » et « le robot est-il au bureau ? », bien que ces deux variables soient nécessaires pour définir complètement la distribution de probabilité de $\mathcal{HOC}'$.

Pour exploiter ce type de régularités, [BOU 00a] suggèrent plusieurs notations pour représenter les fonctions du FMDP à résoudre, telles que les règles [POO 97], les listes de décision [RIV 87] ou les diagrammes de décision binaires [BRY 86]. SPI et SVI sont présentés en utilisant les arbres de décision [QUI 93], principalement à cause de leur simplicité. Nous verrons aussi deux autres méthodes de résolution dans les FMDP utilisant d'autres représentations (section 9.3.2 et 9.3.3).

9.3.1.1. *Les arbres de décision*

Les arbres de décision représentent une fonction en partitionnant son espace d'entrée. Un arbre de décision est composé de :

nœuds intérieurs (ou nœuds de décision) : ils représentent un test sur une variable de l'espace d'entrée. Ils sont parents d'autres nœuds dans l'arbre et définissent la structure de la partition de l'espace d'entrée.

branches : elles connectent un nœud intérieur parent à un nœud enfant en restreignant le domaine des valeurs de la variable en fonction d'un test installé au nœud intérieur parent.

feuilles : elles représentent les nœuds terminaux de l'arbre et sont associées à la valeur de la fonction dans la partition définie par l'ensemble des tests des nœuds intérieurs qui sont les parents de la feuille.

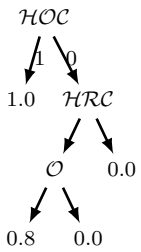
Dans le cadre de SPI et SVI, les arbres de décision sont utilisés pour représenter l'ensemble des fonctions du FMDP à résoudre. Une fonction F représentée avec un arbre de décision est notée $\mathsf{Tree}\,[F]$. Graphiquement, les arbres sont représentés en utilisant la convention suivante : pour un nœud de décision testant une variable X booléenne, les branches de gauche et de droite sont associées respectivement à $X = 1$ et $X = 0$. Lorsque la variable n'est pas booléenne, alors la valeur de X est indiquée sur chaque branche.

9.3.1.2. *Représentation de la fonction de transition*

Dans le problème *Coffee Robot*, la description sous forme tabulaire de la distribution de probabilité $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{O}, \mathcal{HRC}, \mathcal{HOC})$, rappelée figure 9.3(a), fait apparaître plusieurs régularités pouvant être agrégées. Par exemple, on peut remarquer que, comme décrit ci-dessus, dans le contexte $\mathcal{HOC} = 1$, la probabilité que $\mathcal{HOC}'$ soit vrai est égale à 1 quelle que soit la valeur des deux autres variables $\mathcal{O}$ et $\mathcal{HRC}$ appartenant à l'ensemble $\underline{\text{Parents}}_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}')$. En effet, lorsque la propriétaire a un café, alors il est certain qu'elle aura un café au prochain pas de temps. Les arbres de décision permettent de représenter de façon compacte ce type de régularités.

$\mathcal{HOC}$	$\mathcal{HRC}$	$\mathcal{O}$	$\mathcal{HOC}'$
1	1	1	1.0
1	1	0	1.0
1	0	1	1.0
1	0	0	1.0
0	1	1	0.8
0	1	0	0.0
0	0	1	0.0
0	0	0	0.0

(a)



(b)

Figure 9.3. *Représentation sous la forme tabulaire de la distribution de probabilité conditionnelle $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{O}, \mathcal{HRC}, \mathcal{HOC})$ (figure a) et sous la forme d'un arbre de décision (figure b). La feuille notée 0.8 signifie que la probabilité pour la variable $\mathcal{HOC}'$ d'être vraie est : $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{O} = 1, \mathcal{HRC} = 1, \mathcal{HOC} = 0) = 0.8$. Ainsi, certaines régularités sont agrégées, comme par exemple les probabilités $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{HOC} = 1) = 1.0$.*

Un arbre de décision $\mathsf{Tree}\,[P_\tau(X'|\underline{\text{Parents}}_\tau(X'))]$ représentant la distribution de probabilité conditionnelle $P_\tau(X'|\underline{\text{Parents}}_\tau(X'))$ est composé de :

nœuds intérieurs : représentent un test sur une variable $X_j \in \underline{\text{Parents}}_\tau(X')$;

branches : représentent une valeur $x_j \in \underline{\text{Dom}}(X_j)$ de la variable X_j testée au nœud parent et définissant le sous-espace représenté par le nœud enfant connecté à la branche.

les feuilles : représentent la distribution de probabilité $P_f(X'|x[X_j])$, avec $x[X_j]$ l'ensemble des instanciations des variables $X_j \in \underline{\text{Parents}}_\tau(X')$ testées dans les nœuds parents de la feuille f dans l'arbre.

L'interprétation d'un tel arbre est directe : la distribution de probabilité d'une variable X' pour une instanciation x est donnée par l'unique feuille que l'on atteint en choisissant à chaque nœud de décision la branche correspondant à une valeur de test cohérente avec x. Le chemin en question peut spécifier une instanciation partielle de X'.

La figure 9.3(b) représente $\mathsf{Tree}\left[P_{\mathcal{D}\text{elC}}(\mathcal{HOC}'|\mathcal{O},\mathcal{HRC},\mathcal{HOC})\right]$: la distribution de probabilité conditionnelle $P_{\mathcal{D}\text{elC}}(\mathcal{HOC}'|\mathcal{O},\mathcal{HRC},\mathcal{HOC})$ sous la forme d'un arbre de décision. Les valeurs aux feuilles indiquent la probabilité que la variable $\mathcal{HOC}'$ soit vraie. On peut alors remarquer qu'une représentation en arbre de décision, pour la définition de $P_{\mathcal{D}\text{elC}}(\mathcal{HOC}')$, est plus compacte qu'une représentation tabulaire puisqu'elle exploite les indépendances contextuelles : 4 feuilles sont nécessaires à la représentation de la fonction alors que 8 entrées sont nécessaires pour décrire la même fonction sous forme tabulaire. Nous verrons que cette factorisation est utilisée par les algorithmes de planification SPI et SVI.

9.3.1.3. *Représentation de la fonction de récompense*

La représentation d'une fonction de récompense avec des arbres de décision est très similaire à la représentation d'une distribution de probabilité. En effet, la signification des nœuds intérieurs et des branches est la même. Seule change l'étiquette attachée aux feuilles de l'arbre puisqu'elle représente des nombres réels plutôt que des distributions de probabilité.

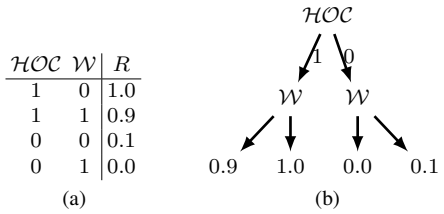


Figure 9.4. *Définition de la fonction de récompense $R(s)$ avec une représentation tabulaire (figure a) et un arbre de décision (figure b). La feuille notée 0.9 signifie $R(\mathcal{HOC}=1,\mathcal{W}=1)=0.9$.*

La figure 9.4 représente la fonction de récompense pour le problème *Coffee Robot* et compare la représentation tabulaire $R(s)$ avec l'arbre de décision $\mathsf{Tree}\left[R(s)\right]$. On

constate qu'aucune indépendance contextuelle n'est utilisable puisque le nombre de feuilles dans l'arbre est égal au nombre de lignes nécessaires à la définition de la fonction avec une représentation tabulaire.

Les algorithmes SPI et SVI ne permettent pas d'exploiter la décomposition additive d'une fonction de récompense telle qu'elle a été définie dans la section 9.2.3.4.

9.3.1.4. *Représentation d'une politique*

Une politique $\pi(s)$ peut aussi être représentée sous la forme d'un arbre de décision $\mathsf{Tree}\,[\pi(s)]$. La figure 9.5 représente une politique stationnaire déterministe $\mathsf{Tree}\,[\pi(s)]$ dans le problème *Coffee Robot*.

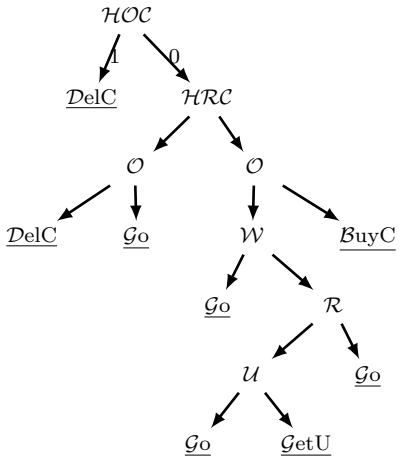


Figure 9.5. *Représentation d'une politique $\pi(s)$ sous la forme d'un arbre de décision* $\mathsf{Tree}\,[\pi(s)]$. *La feuille notée* $\underline{\mathcal{B}\mathrm{uyC}}$ *signifie* $\pi(\mathcal{HOC}=0,\mathcal{HRC}=0,\mathcal{O}=0)=\underline{\mathcal{B}\mathrm{uyC}}$.

L'espace d'état du problème *Coffee Robot* est composé de 6 variables binaires. Une description tabulaire de π aurait donc nécessité $2^6 = 64$ entrées. L'arbre $\mathsf{Tree}\,[\pi]$ ne contient que 8 feuilles (15 nœuds au total). Sur le problème *Coffee Robot*, l'utilisation d'arbres de décision pour représenter une politique permet donc d'exploiter des indépendances contextuelles telles que, lorsque la propriétaire n'a pas de café, que le robot est au bureau et qu'il a un café, il n'est pas nécessaire de connaître la valeur des variables telles que « est-ce qu'il pleut ? » pour déterminer la meilleure action à réaliser.

Lors de l'exécution d'une politique dans un environnement, une telle représentation se révèle avantageuse lorsque déterminer la valeur d'une variable a un coût (par exemple en terme de temps de calcul). En effet, elle permet de n'avoir à déterminer que la valeur des variables strictement nécessaires à l'exécution de la politique de façon

spécifique à l'état courant de l'agent. Une telle propriété permet ainsi d'économiser l'évaluation des variables inutiles.

Enfin, l'utilisation d'un arbre de décision pour la description d'une politique permet d'effectuer un nombre réduit de tests pour déterminer l'action à réaliser pour l'agent. Dans le pire cas, pour un problème décrit avec N variables, seuls N tests sont nécessaires pour déterminer l'action retournée par la politique. Cependant, l'espace mémoire requis pour une telle représentation reste, dans le pire cas, exponentielle en fonction du nombre de variables décrivant l'espace d'états du problème.

9.3.1.5. *Représentation d'une fonction de valeur*

Naturellement, la fonction de valeur V_π d'une politique π peut aussi se représenter sous la forme d'un arbre de décision $\mathsf{Tree}\,[V_\pi]$. La sémantique d'un tel arbre est quasiment identique à celle d'un arbre de décision représentant une fonction de récompense : un nœud de décision représente une variable, une branche représente la valeur de la variable testée au nœud de décision parent et les feuilles représentent la valeur de la fonction de valeur pour la partition délimitée par les tests de ses parents. La figure 9.6 représente la fonction de valeur de la politique $\mathsf{Tree}\,[\pi]$ représentée figure 9.5.

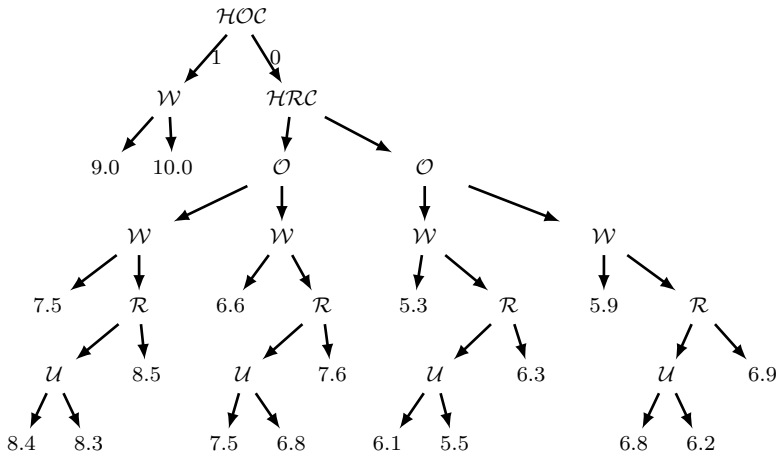


Figure 9.6. *Représentation de la fonction de valeur $V_\pi(s)$ de la politique π sous la forme d'un arbre de décision* $\mathsf{Tree}\,[V_\pi(s)]$ *pour le problème* Coffee Robot. *La feuille notée* 10.0 *signifie* $V_\pi(\mathcal{HOC}=1,\mathcal{W}=0)=10.0$.

L'arbre $\mathsf{Tree}\,[V_\pi]$ ne contient que 18 feuilles (35 nœuds au total) alors qu'une représentation tabulaire aurait nécessité 64 entrées. Sur le problème *Coffee Robot*, une représentation sous la forme d'un arbre de décision permet donc d'exploiter les indépendances contextuelles. Par exemple, la valeur $V_\pi(\mathcal{HOC}=1,\mathcal{W}=0)$ de la politique π, lorsque la propriétaire a un café et que le robot est sec, ne dépend pas des autres

variables du problème. Une telle propriété peut être considérée comme l'agrégation de plusieurs états. Ainsi, lors du calcul itératif d'une fonction de valeur, il n'est nécessaire de calculer qu'une seule fois la mise à jour de la valeur d'une feuille pour mettre à jour la valeur de tous les états représentés par cette feuille.

Cependant, il est possible de constater sur la fonction de valeur $\mathsf{Tree}[V_\pi]$ qu'une telle représentation ne permet pas d'exploiter certaines régularités présentes dans la définition de V_π. En effet, on peut remarquer, par exemple, que la structure des sous-arbres composés des variables $\mathcal{R}$, $\mathcal{W}$, $\mathcal{U}$ et $\mathcal{O}$ est identique. Nous verrons qu'une approximation additive de la fonction de valeur (que nous présenterons section 9.3.3.5, page 297) permet d'exploiter une telle symétrie, contrairement à une représentation telle que les arbres de décision.

Enfin, dans le pire cas, c'est-à-dire lorsque la fonction de valeur de la politique évaluée est différente pour tous les états possibles, la taille de la représentation augmente exponentiellement avec le nombre de variables composant l'espace d'états du problème.

9.3.1.6. *Algorithmes*

Le principe de base des algorithmes SPI et SVI est d'adapter les algorithmes *Policy Iteration* et *Value Iteration* aux arbres de décision. Ainsi, plutôt que d'avoir à calculer une mise à jour de la valeur de chaque état possible lors d'une itération, comme c'est le cas pour *Policy Iteration* et *Value Iteration*, SVI et SPI calculent cette mise à jour pour chaque feuille d'un arbre de décision, permettant de réduire le coût des calculs lorsque plusieurs états sont agrégés et représentés par la même feuille. [BOU 00a] propose une description exhaustive de ces deux algorithmes, que nous ne rappelons pas ici puisque nous nous concentrons sur les représentations.

9.3.2. *L'algorithme* Stochastic Planning Using Decision Diagrams

Dans certains problèmes, la fonction de valeur possède des symétries qui ne sont pas exploitées par les arbres de décision, notamment lorsque la fonction est strictement identique dans plusieurs contextes disjoints. L'algorithme présenté par [HOE 99], nommé SPUDD [5], propose d'utiliser des diagrammes de décision algébriques[6] (ADD), décrits par [BAH 93], pour représenter les fonctions d'un FMDP. De façon semblable à SPI, SPUDD exploite les indépendances relatives à la fois aux fonctions et aux contextes.

L'utilisation d'ADD plutôt que d'arbres de décision présente deux avantages supplémentaires. D'une part, les ADD permettent de mieux factoriser certaines fonctions en exploitant le fait que certaines sous-parties d'une partition de l'espace sont semblables les unes aux autres, alors que les contextes les caractérisants sont disjoints.

---

5. *Stochastic Planning Using Decision Diagrams* (planification stochastique utilisant les diagrammes de décision)

6. *Algebraic Decision Diagrams*

D'autre part, les variables utilisées dans un ADD sont ordonnées. Bien que trouver un ordre optimal des variables à tester pour représenter une fonction de façon la plus compacte possible est un problème difficile, [HOE 00] montrent que plusieurs heuristiques peuvent être utilisées pour trouver un ordre permettant de représenter les fonctions de façon suffisamment compacte pour accélérer nettement les calculs. Un tel ordonnancement est utilisé pour accélérer les calculs réalisés sur les fonctions représentées. Ces deux avantages permettent d'améliorer les algorithmes de programmation dynamique aussi bien en termes d'espace mémoire consommé qu'en termes de temps de calcul.

9.3.2.1. *Les diagrammes de décision algébriques*

Les ADD sont une généralisation des diagrammes de décision binaires (BDD) ou *Binary Decision Diagrams* [BRY 86]. Les BDD sont une représentation compacte de fonctions $\mathcal{B}^n \rightarrow \mathcal{B}$ de n variables binaires vers une valeur binaire. Les ADD généralisent les BDD pour représenter des fonctions réelles $\mathcal{B}^n \rightarrow \mathbb{R}$ de n variables binaires vers une valeur réelle. Un ADD est composé de :

nœuds intérieurs (ou nœuds de décision) : ils représentent un test sur une variable binaire de l'espace d'entrée. Ils sont le parent de deux branches correspondant respectivement au fait que la variable testée est égale à `Vrai` ou `Faux`.

branches : elles connectent un nœud intérieur parent à un nœud enfant en fonction de la valeur `Vrai` ou `Faux` du test installé au nœud intérieur.

feuilles : elles représentent les nœuds terminaux du diagramme et sont associées à la valeur de la fonction dans l'un des sous-espaces définis par l'ensemble des tests des nœuds intérieurs parents de la feuille.

Contrairement à un arbre de décision, les nœuds intérieurs et les feuilles d'un ADD peuvent avoir plusieurs parents. Une fonction F représentée avec un ADD est notée $\mathsf{ADD}\,[F]$. La convention suivante est utilisée pour représenter un ADD graphiquement : pour un nœud de décision testant une variable X, les branches dessinées en trait plein et pointillé sont associées respectivement à $X = 1$ et $X = 0$.

Les ADD possèdent plusieurs propriétés intéressantes. D'une part, pour un ordre de variables donné, chaque fonction distincte n'a qu'une seule représentation. D'autre part, la taille de la représentation de nombreuses fonctions peut être réduite grâce à la réutilisation de sous-graphes identiques au sein de la description. Enfin, il existe des algorithmes optimisés pour la plupart des opérations de base, notamment la multiplication, l'addition ou bien la maximisation.

La figure 9.7 montre l'exemple d'une même fonction F représentée par un arbre de décision et par un ADD. Elle illustre le fait que les arbres de décision, contrairement aux ADD, ne sont pas adaptés pour la représentation de certaines fonctions, notamment les fonctions disjonctives. Ainsi, alors que la représentation $\mathsf{Tree}\,[F]$ contient 5 feuilles différentes (et 4 nœuds intérieurs), la représentation $\mathsf{ADD}\,[F]$ n'en contient que 2 (plus

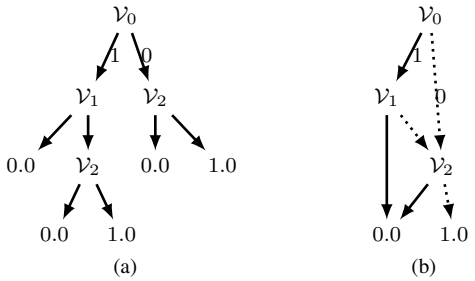


Figure 9.7. *Comparaison des représentations d'une fonction F sous la forme d'un arbre de décision* Tree $[F]$ *(figure a) et d'un diagramme de décision algébrique* ADD $[F]$ *(figure b).*

3 nœuds intérieurs). La mise à jour de cette fonction dans le cas de SPI nécessitera donc 5 calculs de mise à jour différents alors que SPUDD ne réalisera que 2 calculs.

Cependant, l'utilisation des ADD impose principalement deux contraintes sur le FMDP à résoudre. Premièrement, il est nécessaire que les variables du FMDP soient toutes binaires, les ADD ne représentant que des fonctions $\mathcal{B}^n \rightarrow \mathbb{R}$. Pour les problèmes contenant des variables à plus de deux valeurs, il est toujours possible de décomposer ces variables avec de nouvelles variables (binaires). Deuxièmement, les algorithmes basés sur les ADD supposent que, au sein de la structure de données, les tests sur les variables sont ordonnés. Lorsque ces deux contraintes sont satisfaites, il est possible de représenter l'ensemble des fonctions du FMDP à résoudre en utilisant des ADD.

De la même façon que pour SPI et SVI, la plupart des opérateurs sur les fonctions sont redéfinis et optimisés pour manipuler des ADD. L'algorithme SPUDD reprend le principe de l'algorithme *Value Iteration* pour l'adapter aux ADD en supposant que toutes les variables du FMDP à résoudre sont binaires et que les variables sont préalablement ordonnées.

Les travaux sur SPUDD ont été prolongés avec APRICODD [STA 00] qui est une implémentation de SPUDD avec plusieurs améliorations. Premièrement, plusieurs étapes du calcul de l'équation de Bellman sont optimisées afin de permettre à l'utilisateur de pouvoir paramétrer un compromis entre temps de calcul et espace mémoire nécessaire. De plus, il est possible de calculer des fonctions de valeur approchées en spécifiant soit une taille maximale de l'ADD représentant la fonction de valeur, soit une erreur maximale de la représentation [HOE 00]. Enfin, APRICODD propose plusieurs méthodes de réorganisation automatique des variables afin d'éviter à l'utilisateur d'avoir à les

spécifier manuellement. La dernière version d'APRICODD est disponible sur Internet[7]. Les résultats présentés dans [HOE 99, STA 00] suggèrent qu'une telle approche est plus efficace que celle utilisée par les algorithmes SPI ou SVI.

9.3.3. *Programmation linéaire approchée dans un FMDP*

Une alternative à la programmation dynamique pour résoudre un MDP est l'utilisation de la programmation linéaire (section 1.6.2.1). L'utilisation de cette technique pour la résolution d'un FMDP est l'aboutissement de nombreux travaux commencés par [KOL 99, KOL 00] puis menés principalement par Guestrin [GUE 01a, GUE 03a, GUE 03b].

La fonction de valeur optimale d'un MDP peut être calculée en formulant celui-ci sous la forme d'un programme linéaire [MAN 60] :

$$\begin{array}{ll} \text{Pour les variables:} & V(s), \forall s \in S \ ; \\ \text{Minimiser:} & \sum_s \alpha(s) V(s) \ ; \\ \text{Avec les contraintes :} & V(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \\ & \forall s \in S, \forall a \in A. \end{array} \qquad \text{(LP 1)}$$

où $\alpha(s) > 0$ est la pondération d'intérêt de l'état s. La résolution de ce programme linéaire se heurte à un problème de complexité à la fois dans la fonction à optimiser, les variables à déterminer et le nombre de contraintes, ce qui impose d'avoir recours à une méthode approchée pour traiter des problèmes de grande taille.

Ces problèmes sont résolus en exploitant deux idées principales reposant principalement sur les indépendances fonctionnelles et la décomposition additive de la fonction de récompense.

La première idée exploite une représentation approchée de la fonction de valeur, plus précisément une combinaison linéaire de fonctions de base [SCH 85], pour, d'une part, diminuer la complexité de la définition de la fonction à optimiser et du nombre de variables à déterminer et pour, d'autre part, accélérer le calcul de la génération des contraintes. La deuxième idée propose d'utiliser un algorithme de décomposition des contraintes afin de pouvoir représenter l'ensemble des contraintes du programme linéaire de façon compacte.

Ces deux idées sont exploitées par deux algorithmes différents proposés par [GUE 03b]. Le premier algorithme est une reformulation de l'algorithme *Policy Iteration* utilisant la programmation linéaire pour la phase d'évaluation de la politique. Le deuxième algorithme part directement du programme linéaire LP 1 et propose la construction directe d'un programme linéaire afin d'évaluer la fonction de valeur optimale du FMDP à résoudre. La section suivante présente les représentations utilisées par ces deux algorithmes.

7. `http://www.cs.toronto.edu/~jhoey/spudd`

9.3.3.1. *Représentations*

Principalement deux représentations sont utilisées dans l'utilisation de la programmation linéaire telle qu'elle est proposée par Guestrin. La première représentation est une représentation tabulaire classique et permet d'exploiter uniquement les propriétés d'indépendance fonctionnelle et de décomposition additive du problème. La deuxième représentation est une représentation structurée basée sur des règles [ZHA 99] permettant en plus d'utiliser les indépendances contextuelles au sein d'une fonction. Bien que [GUE 03b] montrent que, pour certains problèmes, une représentation tabulaire est plus rapide qu'une représentation structurée, nous pensons que les représentations structurées sont mieux adaptées pour représenter des problèmes réels, justement parce qu'elles exploitent les indépendances contextuelles. De plus, le pire des cas des représentations structurées est souvent moins mauvais que le pire des cas des représentations tabulaires en terme de temps de calcul [STA 00, GUE 03a].

Deux avantages sont avancés par [GUE 03b] pour justifier l'utilisation des règles plutôt qu'une autre représentation telle que les arbres de décision ou les ADD. Premièrement, cette représentation est bien adaptée à leur technique de décomposition des contraintes du programme linéaire. Deuxièmement, contrairement aux arbres de décision ou aux ADD, les règles utilisées pour décrire une fonction peuvent ne pas être exclusives. Deux types de règles sont distinguées : les règles de probabilité (*probability rules*) et les règles de valeur (*value rules*). Les règles de probabilité sont utilisées pour représenter la fonction de transition alors que les règles de valeur sont utilisées pour définir les fonctions de récompense ainsi que les fonctions de valeur. Ces deux types de règles et leurs utilisations dans le cadre de la programmation linéaire approchée dans un FMDP sont décrits dans la suite de cette section, d'après [GUE 03b]. Une fonction $F(x)$ représentée avec un ensemble de règles est notée $\mathsf{Rule}\,[F]$.

9.3.3.2. *Représentation de la fonction de transition*

Le premier type de règles est utilisé pour représenter la fonction de transition, plus précisément les distributions de probabilité conditionnelles quantifiant les DBN. Une règle correspond à un ou plusieurs contextes dans la distribution ayant la même probabilité. Nous commençons par définir la cohérence entre deux contextes :

Définition 31 (Cohérence entre deux contextes) *Soit $C \subseteq \{X, X'\}$, $c \in \underline{Dom}(C)$, $B \subseteq \{X, X'\}$ et $b \in \underline{Dom}(B)$. On dit que les deux contextes b et c sont cohérents s'ils ont tous les deux les mêmes valeurs pour toutes les variables appartenant à l'intersection $C \cap B$.*

Ainsi, des contextes possédant des variables avec des valeurs identiques sont définis comme étant *cohérents*. Les probabilités ayant la même valeur et des contextes cohérents sont représentées avec des règles de probabilité :

Définition 32 (Règle de probabilité) *Une règle de probabilité $\eta = |c : p|$ est une fonction $\eta : \{X, X'\} \mapsto [0, 1]$ avec le contexte $c \in \underline{Dom}(C)$, $C \subseteq \{X, X'\}$ et*

$p \in [0,1]$ *et tel que* $\eta(s,x') = p$ *si les instanciations* s *et* x' *sont cohérentes avec* c*, ou sinon est égal à 1.*

Deux règles sont dites cohérentes si leurs contextes respectifs sont cohérents. On définit maintenant un ensemble de règles de probabilité pour définir complètement une distribution de probabilité conditionnelle :

Définition 33 (Ensemble de règles de probabilité) *Un ensemble de règles* P_a *d'une distribution de probabilité conditionnelle est une fonction* $P_a : (\{X'_i\} \cup X) \mapsto [0,1]$ *composée des règles de probabilité* $\{\eta_1, \ldots, \eta_m\}$ *dont les contextes sont mutuellement exclusifs et exhaustifs. On définit :* $P_a(x'_i|x) = \eta_j(x, x'_i)$ *avec* η_j *l'unique règle appartenant à* P_a *dont le contexte* c_j *est cohérent avec* (x'_i, x)*. De plus, on a nécessairement :* $\forall s \in S : \sum_{x'_i} P_a(x'_i|s) = 1$.

Il est possible de définir $\underline{\text{Parents}}_a(X'_i)$ comme l'union des variables appartenant aux contextes des règles appartenant à $P_a(X'_i)$.

A l'instar des arbres de décision, les ensembles de règles de probabilité permettent d'exploiter les indépendances contextuelles. De plus, les arbres de décision forment une partition complète d'un espace. Il est donc facile de définir un ensemble de règles mutuellement exclusives et exhaustives à partir d'un arbre de décision, comme le montre la figure 9.8 pour définir $P_{\mathcal{D}\text{elC}}(\mathcal{HOC}')$.

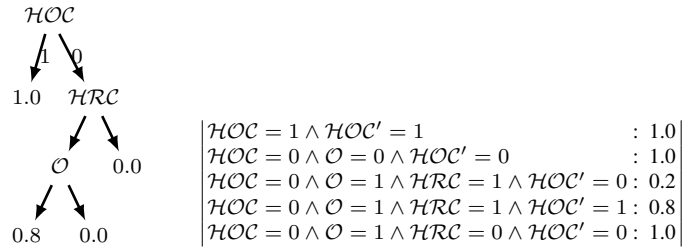


Figure 9.8. *Représentation de la distribution de probabilité conditionnelle* $P_{\mathcal{D}\text{elC}}(\mathcal{HOC}')$ *sous la forme d'un arbre de décision et d'un ensemble de règles. La règle* $|\mathcal{HOC} = 0 \wedge \mathcal{O} = 1 \wedge \mathcal{HRC} = 1 \wedge \mathcal{HOC}' = 1 : 0.8|$ *définit* $P_{\mathcal{D}\text{elC}}(\mathcal{HOC}' = 1|\mathcal{HOC} = 0, \mathcal{O} = 1, \mathcal{HRC} = 1) = 0.8$.

La probabilité $P_{\mathcal{D}\text{elC}}(\mathcal{HOC}' = 1|\mathcal{HOC} = 0, \mathcal{O} = 1, \mathcal{HRC} = 1) = 0.8$ est représentée par la règle correspondante $|\mathcal{HOC} = 0 \wedge \mathcal{O} = 1 \wedge \mathcal{HRC} = 1 \wedge \mathcal{HOC}' = 1 : 0.8|$. On peut remarquer que, pour les tests concernant les variables X au temps t, le contexte de cette règle correspond aux tests réalisés dans l'arbre de décision pour atteindre la feuille 0.8. De plus, la variable X'_i au temps $t+1$ appartiennent aussi au contexte de la règle. Une distribution de probabilité conditionnelle $F(x)$ représentée avec un ensemble de règles de probabilité est notée $\mathsf{Rule_p}\,[F]$.

9.3.3.3. *Représentation de la fonction de récompense*

Pour représenter la fonction de récompense d'un FMDP, on définit les règles de valeur :

Définition 34 (Règle de valeur) *Une règle de valeur $\rho = |c : v|$ est une fonction $\rho : X \rightarrow \mathbb{R}$ telle que $\rho(x) = v$ lorsque x est cohérent avec le contexte c et 0 sinon.*

On note que la portée d'une règle de valeur est $\underline{\text{Scope}}(\rho) = C$ avec C l'ensemble des variables instanciées dans le contexte c de la règle $\rho = |c : v|$.

Il est maintenant possible de définir une fonction comme un ensemble de règles de valeur :

Définition 35 (Ensemble de règles de valeur) *Un ensemble de règles de valeur représentant une fonction $f : X \mapsto \mathbb{R}$ est composé de l'ensemble des règles de valeur $\{\rho_1, \ldots, \rho_n\}$ telles que $f(x) = \sum_{i=1}^{n} \rho_i(x)$ avec $\forall i : \underline{Scope}(\rho_i) \subseteq X$.*

Une fonction F représentée avec un ensemble de règles de valeur est notée $\mathsf{Rule_v}[F]$. De plus, on suppose qu'une récompense $R(s, a)$ peut s'écrire sous la forme d'une somme de fonctions de récompense dont la portée est limitée :

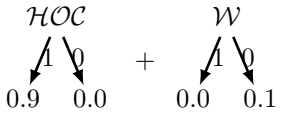
$$R(s, a) = \sum_j r_j^a(s). \tag{9.4}$$

Cette représentation permet de représenter de façon naturelle des fonctions en exploitant à la fois des indépendances contextuelles et une décomposition additive, comme le montre la figure 9.9.

Représentation tabulaire :

$$R(s) = \begin{array}{cc|c} \mathcal{HOC} & & R_0 \\ \hline 0 & & 0.0 \\ 1 & & 0.9 \end{array} + \begin{array}{cc|c} \mathcal{W} & & R_1 \\ \hline 0 & & 0.1 \\ 1 & & 0.0 \end{array}$$

Arbres de décision :



Ensembles de règles de valeur :

$$R(s) = \begin{vmatrix} \mathcal{HOC} = 1 : & 0.9 \\ \mathcal{W} = 0 & : 0.1 \end{vmatrix} = |\mathcal{HOC} = 1 : 0.9| + |\mathcal{W} = 0 : 0.1|$$

Figure 9.9. *Représentation de la fonction de récompense R décomposée en une somme de fonction de récompense dont la portée est restreinte à une seule variable du problème.*

Comme nous l'avons décrit dans la section 9.2.3.4, la fonction de récompense du problème *Coffee Robot* peut être décomposée en une somme de deux fonctions dont

la portée n'est restreinte qu'à une seule variable du problème. Plusieurs représentations peuvent être utilisées pour représenter les fonctions composant la fonction de récompense, notamment une forme tabulaire, d'arbres de décision ou d'un ensemble de règles de valeur.

La figure 9.9 montre que deux configurations sont possibles, soit en regroupant les règles au sein d'un même ensemble pour ne définir qu'une seule fonction, soit en séparant les règles dans deux ensembles différents pour définir deux fonctions différentes. Enfin, on remarque que, même sur cet exemple simple, les règles de valeur permettent d'exploiter les indépendances contextuelles pour décrire la fonction de récompense du problème *Coffee Robot*, contrairement aux arbres de décision. En effet, les arbres de décision requièrent la représentation des feuilles contenant la valeur 0, ce qui n'est pas le cas des règles.

9.3.3.4. *Représentation d'une politique*

Pour représenter une politique π de façon compacte, [GUE 03b] reprennent une technique présentée par [KOL 00]. Plutôt que d'utiliser un arbre de décision $\mathsf{Tree}[\pi]$ ou un ADD $\mathsf{ADD}[\pi]$ pour représenter une définition structurée de π, une action par défaut est choisie a priori dans le FMDP et la politique est représentée comme une liste de décision ordonnée. Chaque élément de la liste est composé de trois informations différentes : un contexte indiquant si la décision peut être prise étant donné un état s, l'action à exécuter si la décision est prise et enfin le bonus indiquant la récompense espérée supplémentaire pour cette décision comparée à la récompense espérée de l'action par défaut. Le dernier élément de la liste est toujours l'action par défaut, associée à un contexte vide (pour que ce dernier élément représente la décision par défaut à prendre si aucun autre n'est cohérent avec l'état) et un bonus de 0. Une politique π représentée sous la forme d'une liste de décision est notée $\mathsf{List}[\pi]$. La figure 9.10 montre l'exemple d'une politique dans le problème *Coffee Robot* dont l'action par défaut est $\mathcal{G}\mathrm{o}$.

	Contexte	Action	Bonus
0	$\mathcal{HOC}=0 \wedge \mathcal{HRC}=1 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=0 \wedge \mathcal{O}=1$	$\mathcal{D}\mathrm{elC}$	2.28
1	$\mathcal{HOC}=0 \wedge \mathcal{HRC}=0 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=0 \wedge \mathcal{O}=0$	$\mathcal{B}\mathrm{uyC}$	1.87
2	$\mathcal{HOC}=0 \wedge \mathcal{HRC}=1 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=1 \wedge \mathcal{O}=1$	$\mathcal{D}\mathrm{elC}$	1.60
3	$\mathcal{HOC}=0 \wedge \mathcal{HRC}=1 \wedge \mathcal{W}=1 \wedge \mathcal{O}=1$	$\mathcal{D}\mathrm{elC}$	1.45
4	$\mathcal{HOC}=0 \wedge \mathcal{HRC}=1 \wedge \mathcal{W}=0 \wedge \mathcal{R}=0 \wedge \mathcal{O}=1$	$\mathcal{D}\mathrm{elC}$	1.44
5	$\mathcal{HOC}=0 \wedge \mathcal{HRC}=0 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=1 \wedge \mathcal{O}=0$	$\mathcal{B}\mathrm{uyC}$	1.27
6	$\mathcal{HOC}=0 \wedge \mathcal{HRC}=0 \wedge \mathcal{W}=1 \wedge \mathcal{O}=0$	$\mathcal{B}\mathrm{uyC}$	1.18
7	$\mathcal{HOC}=0 \wedge \mathcal{HRC}=0 \wedge \mathcal{W}=0 \wedge \mathcal{R}=0 \wedge \mathcal{O}=0$	$\mathcal{B}\mathrm{uyC}$	1.18
8	$\mathcal{HOC}=1 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=0$	$\mathcal{D}\mathrm{elC}$	0.84
9	$\mathcal{HOC}=0 \wedge \mathcal{HRC}=0 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=0 \wedge \mathcal{O}=1$	$\mathcal{G}\mathrm{etU}$	0.18
10	$\mathcal{HOC}=1 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=1$	$\mathcal{D}\mathrm{elC}$	0.09
11	$\emptyset$	$\mathcal{G}\mathrm{o}$	0.00

Figure 9.10. *Représentation d'une politique $\pi(s)$ sous la forme d'une liste de décision $\mathsf{List}[\pi]$ (avec $\mathcal{G}\mathrm{o}$ l'action par défaut).*

On peut remarquer que la politique représentée figure 9.10 n'est pas simplifiée. En effet, par exemple, la règle 3 peut être agrégée avec la règle 1 puisque ces deux règles ont le même contexte (la règle 3 ne sera jamais utilisée puisque la règle 1 sera nécessairement utilisée avant). De plus, contrairement aux arbres de décision ou aux ADD, le nombre de tests réalisés pour déterminer l'action à exécuter peut être supérieur au nombre de variables décrivant le problème.

Enfin, pour certains problèmes de grande taille, quelle que soit la méthode de planification utilisée, une représentation explicite de la politique optimale, même factorisée, est impossible puisqu'il est nécessaire pour chaque état d'évaluer toutes les variables du problème afin de déterminer la meilleure action à réaliser par l'agent. C'est la raison pour laquelle [GUE 03b] propose des algorithmes ne nécessitant pas une représentation explicite de la politique du problème.

9.3.3.5. *Représentation de la fonction de valeur*

Nous avons vu qu'un MDP pouvait s'écrire sous la forme d'un programme linéaire de la façon suivante (section 1.6.2.1, LP 1, page 292) :

$$\begin{array}{ll} \text{Pour les variables:} & V(s), \forall s \in S\ ; \\ \text{Minimiser:} & \sum_s \alpha(s) V(s)\ ; \\ \text{Avec les contraintes :} & V(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \\ & \forall s \in S, \forall a \in A. \end{array} \qquad \text{(LP 2)}$$

Cependant, une telle représentation pose un problème de complexité, aussi bien en le nombre de variables à déterminer, qu'en le nombre de termes de la somme de la fonction objectif, ou en le nombre de contraintes.

Une solution pour éviter l'explosion combinatoire concernant le nombre de variables à déterminer et le nombre de termes dans la fonction à minimiser est l'approximation de la fonction de valeur par une *combinaison linéaire* proposée par [BEL 63] (voir le chapitre 11). L'espace des fonctions de valeur approchées $\tilde{V} \in \mathcal{H} \subseteq \mathbb{R}^n$ est défini via un ensemble de *fonctions de base*, ou *basis functions*, dont la portée est limitée à un petit nombre de variables :

Définition 36 (Fonction de valeur linéaire) *Une fonction de valeur linéaire $\tilde{V}$ sur un ensemble de fonctions de base $H = \{h_0, \ldots, h_k\}$ est une fonction telle que $\tilde{V}(s) = \sum_{j=1}^{k} w_j h_j(s)$ avec $w \in \mathbb{R}^k$.*

Cette approximation peut être utilisée pour redéfinir le programme linéaire simplement en remplaçant la fonction de valeur à déterminer par son approximation [SCH 85] :

$$\begin{array}{ll} \text{Pour les variables:} & w_1, \ldots, w_k\ ; \\ \text{Minimiser:} & \sum_s \alpha(s) \sum_{i=1}^{k} w_i h_i(s)\ ; \\ \text{Avec les contraintes :} & \sum_{i=1}^{k} w_i h_i(s) \geq R(s,a) + \\ & \qquad \gamma \sum_{s'} P(s'|s,a) \sum_{i=1}^{k} w_i h_i(s') \\ & \forall s \in S, \forall a \in A. \end{array} \qquad \text{(LP 3)}$$

Ainsi, plutôt que de déterminer la fonction de valeur dans l'espace complet des fonctions de valeur, l'espace de recherche est réduit à l'espace des valeurs pour l'ensemble des coefficients utilisés dans la combinaison linéaire. De plus, le fait de limiter la portée des fonctions de base permet d'exploiter les indépendances relatives aux fonctions de base.

On peut donc remarquer que le nombre de variables à déterminer du programme linéaire n'est plus le nombre d'états possibles mais le nombre de coefficients dans l'approximation linéaire. Cependant, le nombre de termes dans la fonction à minimiser et le nombre de contraintes sont toujours égaux aux nombres d'états dans le problème.

Pour un tel programme, une solution existe si une fonction de base constante est incluse dans l'ensemble des fonctions de base [SCH 85]. Nous supposons donc qu'une telle fonction h_0, telle que $h_0(s) = 1, \forall s \in S$, est systématiquement incluse à l'ensemble des fonctions de base. De plus, il est important de noter que le choix des pondérations d'intérêt $\alpha(s)$ influe sur la qualité de l'approximation [FAR 01].

En plus de la diminution de la complexité du programme linéaire, une telle approximation de la fonction de valeur permet d'exploiter à la fois les indépendances fonctionnelles et certaines régularités de la structure de la fonction de valeur. Dans le problème *Coffee Robot*, la figure 9.11 montre un exemple de décomposition additive de la fonction de valeur approchée permettant d'exploiter une régularité que des représentations telles que les arbres de décision et les ADD ne pouvaient pas utiliser.

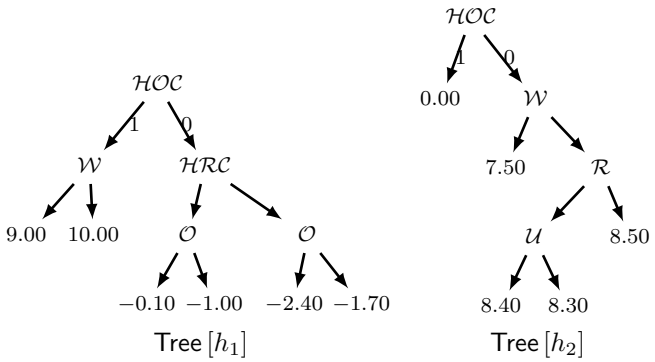


Figure 9.11. *Exemple de décomposition de la fonction de valeur du problème* Coffee Robot *sous la forme de deux arbres de décision représentant deux fonctions de base permettant de calculer la politique $\pi(s)$ (figure 9.10). La fonction de valeur optimale approchée est :* $\tilde{V}^*(s) = 0.63 \cdot \mathsf{Tree}[h_0] + 0.94 \cdot \mathsf{Tree}[h_1] + 0.96 \cdot \mathsf{Tree}[h_2]$. *L'arbre* $\mathsf{Tree}[h_0]$ *n'est pas illustré puisqu'il définit une fonction constante et ne contient donc qu'une seule feuille égale à 1.*

La définition d'une fonction de valeur Tree $[V]$ du problème est décomposée en deux fonctions de base Tree $[h_1]$ et Tree $[h_2]$[8] et permet une approximation de Tree $[V_\pi]$ dont l'erreur est inférieure à 1. La propriété de décomposition additive est exploitée puisque, plutôt que de contenir 18 feuilles, cette représentation ne nécessite que 11 feuilles pour les deux arbres (soit 20 nœuds au total, au lieu de 35 nœuds pour Tree $[V_\pi]$). Cette décomposition contient deux fonctions de base (trois en comptant la fonction constante h_0), donc trois coefficients, w_0, w_1 et w_2, sont à déterminer dans le programme linéaire 2.

Enfin, lorsque la fonction de récompense possède une décomposition additive, comme c'est le cas dans le problème *Coffee Robot*, il semble naturel que la fonction de valeur du problème possède également cette propriété. Cependant, ces deux propriétés ne sont pas nécessairement corrélées. En effet, bien qu'une fonction de récompense puisse ne présenter aucune décomposition additive, une combinaison linéaire de fonctions de base peut quand même permettre de déterminer avec une faible erreur d'approximation les fonctions de valeur du problème. Une telle représentation est donc plus générale que les représentations sous forme d'arbre de décision ou d'ADD proposées par SPI ou SPUDD [GUE 03b]. Réciproquement, des représentations compactes des fonctions de transition et de récompense n'impliquent pas non plus une représentation compacte de la fonction de valeur [KOL 99, MUN 00, LIB 02].

9.3.3.6. *Algorithmes*

Les algorithmes [GUE 03b] permettent ainsi, à partir de la définition d'un problème sous la forme d'un FMDP, de générer le programme linéaire associé afin de calculer une approximation de la fonction de valeur du problème. De plus, des algorithmes sont aussi proposés afin d'obtenir une représentation de la politique (sous la forme exposée dans la section 9.3.3.4). Cependant, une telle représentation peut s'avérer trop coûteuse, c'est la raison pour laquelle les auteurs proposent une autre alternative : une fois la fonction de valeur calculée, pour un état donné, il est facile de calculer la valeur d'action pour chaque action, permettant ainsi de trouver la meilleure action à exécuter pour cet état. Ainsi, une représentation explicite de la politique est évitée. Le lecteur pourra consulter [GUE 03b] pour obtenir une description exhaustive de ces algorithmes.

9.4. Conclusion et perspectives

La résolution efficace de problèmes décisionnels de Markov de grande taille dans un cadre factorisé est un domaine de recherche très actif. Plusieurs extensions de ce cadre ont été proposées, notamment dans le cadre partiellement observable [**?**] et dans celui de l'apprentissage par renforcement. Cette seconde extension se justifie par le

8. Ces deux fonctions Tree $[h_1]$ et Tree $[h_2]$ ont été obtenues à partir de l'arbre de décision représentant la fonction de valeur Tree $[V_\pi]$ dans le problème *Coffee Robot*, figure 9.6 (page 288).

fait que définir complètement la fonction de transition et de récompense d'un FMDP peut s'avérer fastidieux, voir impossible dans certains cas.

Dans ce cadre, la première famille d'algorithmes est l'adaptation directe des algorithmes d'apprentissage par renforcement présentés dans la section 2.5.2, nommément DBN-E^3 [KEA 99], *factored* R-MAX [STR 07] et *factored* I.E. [STR 07]. Ces algorithmes supposent que la structure des DBN du FMDP est connue, mais pas quantifiée. Les algorithmes proposent alors un apprentissage permettant d'obtenir une politique proche d'une politique optimale du FMDP en un temps fini.

La deuxième famille d'algorithmes ne fait aucune hypothèse sur la structure de la fonction de transition et de récompense du FMDP. En s'inspirant de l'apprentissage par renforcement indirect (cf. tome 1, section 2.5), de récents travaux proposés par [DEG 07] proposent d'apprendre la structure des problèmes à partir de l'expérience d'un agent et en utilisant l'induction d'arbres de décisions. Bien que les résultats expérimentaux soient intéressants, une preuve mathématique de cette approche n'existe pas encore. Nous renvoyons le lecteur à [DEG 07] pour un exposé plus précis de ces dernières méthodes.

Des recherches destinées à combler le fossé entre les approches disposant d'une preuve de convergence et celles qui en sont dépourvues sont elles aussi très actives. L'approche suivie par l'équipe de Littman dans ce cadre consiste à proposer des preuves de convergence pour des algorithmes présupposant de moins en moins de connaissances a priori sur la structure du FMDP qu'il s'agit de résoudre. On consultera en particulier [STR 07] sur ce point.

Enfin, une dernière extension du cadre des FMDP consiste à combiner la factorisation et une approche hiérarchique. Les travaux de thèse de Teichteil [TEI 05c] se sont intéressés à ce cadre. On peut aussi citer l'algorithme VISA [JON 06] qui présente des performances comparables à celles des algorithmes de Guestrin avec des méthodes de programmation dynamique.

Chapitre 10

Approches de résolution en ligne

10.1. Introduction

Nous avons vu dans les précédents chapitres de cet ouvrage comment résoudre de manière approchée des MDP de grande taille par différentes techniques reposant sur la représentation paramétrique ou structurée des politiques et/ou fonctions de valeur et sur l'emploi de la simulation pour les techniques d'apprentissage par renforcement. A l'issue du processus d'optimisation, on obtient une politique optimale approchée $\tilde{\pi}$ valide pour l'ensemble de l'espace d'états. Pour des MDP de très grande taille, obtenir une bonne approximation est souvent difficile, d'autant plus que, en général, on ne sait pas quantifier précisément la sous-optimalité de la politique a priori. Une amélioration possible consiste alors à considérer ces méthodes d'optimisation comme un pré-calcul hors ligne. Lors d'une seconde phase en ligne, la politique a priori est améliorée par un calcul non élémentaire pour chaque état rencontré.

10.1.1. *Exploiter le temps en ligne*

Dans le cadre des MDP, l'algorithme utilisé pour déterminer en ligne l'action courante est en général très simple. Ainsi, lorsque $\tilde{\pi}$ est définie par l'intermédiaire d'une fonction de valeur $\tilde{V}$, cet algorithme est une simple comparaison des valeurs des actions "à un coup" (voir algorithme 1.3 du chapitre 1). De même, dans le cas d'une politique paramétrée, l'algorithme permettant de déterminer l'action courante est habituellement très élémentaire. Ce schéma de résolution permet un contrôle très réactif et est bien adapté pour des systèmes embarqués soumis à de fortes contraintes "temps-réel". En revanche, il peut être remis en cause pour améliorer en ligne une politique lorsque du temps est disponible pour décider.

Chapitre rédigé par Laurent PÉRET et Frédérick GARCIA.

C'est l'une des clés du succès des meilleurs programmes de jeux qui combinent intensivement calculs hors ligne et en ligne. Par exemple, le fameux programme d'échecs Deep Blue [**?**] repose sur une fonction de valeur complexe définie par plus de 8000 fonctions caractéristiques. Les poids de cette fonction de valeur ont fait l'objet de nombreuses optimisations hors ligne, tant manuelles qu'automatiques. La fonction de valeur ainsi obtenue fournit une excellente évaluation de la force d'une position, intégrant le jugement d'experts humains. Néanmoins, elle reste synthétique et imparfaite : l'algorithme élémentaire qui consiste à choisir le coup menant à la position de meilleure évaluation depuis la position courante ne permet pas toujours de choisir un coup efficace. Si Deep Blue est un programme capable de battre les meilleurs joueurs humains, c'est parce qu'il couple la fonction de valeur à un algorithme de recherche arborescente qui développe des millions de positions obtenues sur plusieurs coups à partir de chaque position rencontrée en cours de partie.

10.1.2. *Recherche en ligne par simulation*

Pour des MDP de grande taille, effectuer une recherche arborescente peut s'avérer une tâche délicate, en particulier lorsque les probabilités de transition sont inconnues ou lorsque le nombre d'états successeurs d'un état est élevé — autrement dit, en termes de recherche arborescente, lorsque le facteur de branchement est élevé. Comme pour les algorithmes hors ligne, l'emploi de la simulation permet de contourner ces difficultés en effectuant les calculs sur la base d'échantillons de transitions simulées. Récemment, Kearns, Mansour et Ng ont proposé un algorithme [**?**] posant les fondements théoriques pour combiner en ligne recherche arborescente et simulation. Cet algorithme construit un arbre par simulation stochastique sur un *horizon de raisonnement* H et définit une politique stochastique. Cette approche est très séduisante puisqu'elle permet de traiter des MDP arbitrairement grands avec pour seule donnée l'existence d'un simulateur du système[1]. Le prix à payer pour cette généralité est que le nombre d'appels au simulateur requis par cet algorithme est énorme, ce qui le rend inopérant dans de nombreux cas.

Une alternative aux techniques de recherche arborescente pour améliorer une politique en ligne a été proposée par Tesauro et Galperin [**?**]. Il ne requiert que l'existence d'un simulateur et d'une politique a priori $\tilde{\pi}$ — définie par une fonction de valeur ou paramétrée. Il peut être envisagé comme une itération focalisée de l'itération sur les politiques avec approximation. Cet algorithme s'avère parfois très lourd en calculs mais, en revanche, et à la différence de l'algorithme de Kearns et al., il a fait ses preuves expérimentalement, notamment en améliorant le niveau de jeu du programme de backgammon TD-Gammon [**?**].

1. L'intégration d'une fonction de valeur $\tilde{V}$ pour évaluer les feuilles de l'arbre constitue une amélioration naturelle de l'algorithme.

Pour les problèmes de grande dimension, il apparaît ainsi nécessaire de contrôler en ligne la recherche afin d'améliorer rapidement le choix de l'action à effectuer au fur et à mesure que des simulations supplémentaires sont allouées à son calcul. C'est l'objectif visé par les méthodes Focused Reinforcement Learning (Apprentissage par Renforcement Focalisé) et Controlled Rollout (Déroulement Contrôlé) proposées dans [**?**, **?**] et que nous présentons ci-dessous.

10.2. Algorithmes en ligne pour la résolution d'un MDP

Nous formalisons ici le problème de la recherche en ligne pour les MDP : le problème peut être envisagé comme la résolution locale d'un MDP sur un certain *horizon de raisonnement*. Nous recensons ensuite les quelques approches en ligne permettant de résoudre ce problème. Par rapport aux algorithmes hors ligne, l'idée maîtresse est de focaliser les calculs autour d'un état courant.

10.2.1. *Algorithmes hors ligne, algorithmes en ligne*

L'usage que nous faisons des termes "en ligne" et "hors ligne" mérite d'être précisé. Nous aurions pu utiliser pour distinguer les méthodes classiques de résolution présentées dans les chapitres précédents et celles développées ici les termes "non focalisées" et "focalisées" ou "globales" et "locales". Nous avons choisi "hors ligne" et "en ligne" car ces termes reflètent bien la succession de deux phases de résolution qui caractérise cette approche.

Nous devons toutefois préciser que nombre d'algorithmes d'A/R que nous classons dans la catégorie des algorithmes hors ligne se réclament souvent au contraire d'une approche en ligne [**?**, RUM 94]. Ces algorithmes, bien que relevant d'une résolution globale, n'envisagent qu'une unique phase d'optimisation alors généralement appelée phase d'apprentissage. Cette unique phase est qualifiée d'en ligne car elle peut être mise en œuvre directement en situation réelle, par exemple pour contrôler les actions d'un robot s'adaptant à son environnement [**?**]. En effet, les algorithmes de l'A/R effectuent typiquement une succession de mises à jour élémentaires compatibles avec des contraintes "temps-réel" fortes ou des ressources en calcul limitées.

L'intérêt de l'existence d'une phase en ligne est au contraire d'effectuer un calcul non trivial pour chaque état rencontré. Nous nous intéresserons donc dans cette partie aux algorithmes se focalisant sur le calcul d'une action optimale pour un état courant donné.

10.2.2. *Formalisation du problème*

Nous formalisons dans cette section le problème local de la détermination de la meilleure action pour un état courant sur un certain *horizon de raisonnement* H. Nous présentons un algorithme simple par recherche avant qui examine toutes les séquences

d'actions possibles sur cet horizon H. Cet algorithme élémentaire mais lourd permet de quantifier le gain obtenu pour un horizon de raisonnement donné.

Nous développerons dans les sections suivantes les algorithmes permettant de résoudre efficacement ce problème local.

10.2.2.1. *Recherche avant sur un horizon de raisonnement*

Le problème de la détermination de la meilleure action pour l'état courant s_t peut être formulé comme la résolution d'un MDP *local* restreint à l'ensemble $S_H(s_t) \subset S$ *des états pouvant être atteints depuis* s_t *en au plus* H *transitions*. A ce MDP local est associé le graphe ET/OU $G_H(s_t)$, tel que

– les états sont des *nœuds* OU dont sont issus $|A|$ *arcs* ; chaque arc est connecté à un nœud action ; à un nœud état est associé un opérateur de minimisation ;

– les actions sont des *nœuds* ET dont sont issus $|S(s,a)|$ arcs ; chaque arc est connecté à un nœud état s' et évalué par une probabilité de transition $p(s' \mid s,a)$; à un nœud action est associé un opérateur d'espérance.

Nous supposons que nous disposons d'une fonction de valeur $\tilde{V}$ approximant V^* pour tout état s. Cette fonction de valeur est utilisée ici pour évaluer les états s_{t+H} situés à la frontière de $S_H(s_t)$ (voir figure 10.1).

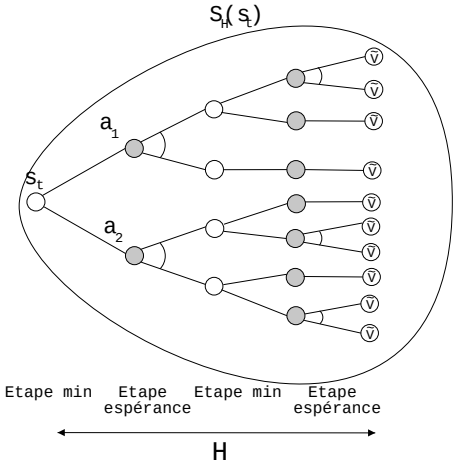


Figure 10.1. *Recherche avant sur un horizon H. Les cercles blancs représentent les nœuds état et les cercles gris les nœuds action.*

Nous supposerons que l'erreur d'approximation sur $\tilde{V}$ est bornée par un réel ϵ :

$$\exists\, \epsilon > 0 \text{ tel que } \| \tilde{V} - V^* \|_\infty \leq \epsilon \tag{10.1}$$

Le moyen le plus direct de résoudre le MDP local associé à s_t consiste à effectuer une *recherche avant* exhaustive sur l'horizon de raisonnement H considéré. Une nouvelle fonction de valeur V_H est alors calculée en ligne. L'ensemble $S_H(s_t)$ est explicité récursivement depuis s_t et V_H est calculée suivant le principe d'optimalité de Bellman selon l'équation (10.2) de l'algorithme 10.1.

Algorithme 10.1 : Algorithme par recherche avant sur un horizon de raisonnement

Entrées : état courant s_t, p, c, horizon H

$$V_H(s_t) = \begin{cases} \tilde{V}(s_t) \quad \text{si } H = 0 \\ \min_{a \in A} Q_H(s_t, a) \quad \text{sinon} \\ \text{où} \quad Q_H(s_t, a) = c(s_t, a) + \gamma \sum_{s' \in S(s_t,a)} p(s' \mid s_t, a) V_{H-1}(s') \end{cases} \tag{10.2}$$

retourner $a_t = \text{argmin}_{a \in A} Q_H(s_t, a)$

10.2.2.2. *Arbre ou graphe ?*

Comme pour les algorithmes classiques de recherche dans les graphes, il est possible de simplifier le graphe markovien en un *arbre*. Dans ce cas, lorsqu'un nouvel état est explicité, on le relie simplement à la paire état/action dont il est issu, sans faire de test d'occurrence.

Cette simplification, couramment utilisée dans les programmes de jeux, présente l'avantage de la rapidité. La contrepartie est un coût en mémoire plus important lorsque des états sont dupliqués (voir [**?**], pages 212-213). La représentation en arbre est privilégiée lorsque le risque de voir un état dupliqué est faible.

Nous verrons que les deux représentations sont utilisées dans le cadre des MDP : les approches heuristiques en ligne décrites section 10.2.3 comme l'algorithme LAO^* utilisent une représentation par graphe tandis que l'algorithme de Kearns et al. décrit section 10.2.4 se base sur une représentation en arbre.

10.2.2.3. *Complexité et efficacité de la recherche avant*

La complexité de l'algorithme 10.1 par recherche avant, exponentielle en H, est en $O((S_{max}|A|)^H)$ où $S_{max} = \max_{s \in S_H(s_t)} \max_{a \in A} |S(s, a)|$ est le nombre maximal d'états successeurs pour une paire état/action.

Comme pour l'algorithme d'itération sur les valeurs, on montre alors la proposition suivante :

PROPOSITION 10.1 [**?**].– *Erreur pour la recherche avant :*

$$\| V_H - V^* \|_\infty \leq \gamma^H \epsilon.$$

Cette proposition établit que V_H converge géométriquement vers V^*. De ce fait, si l'horizon de raisonnement est assez long, l'erreur due à l'imprécision de $\tilde{V}$ sera significativement réduite.

Dans le cas d'un problème de plus court chemin stochastique avec $\gamma = 1$, il est nécessaire que $\epsilon = \epsilon_H$ diminue lorsque H augmente pour rendre la recherche avant bénéfique. En d'autres termes, l'erreur doit être plus petite pour les états feuille que pour l'état racine. Comme dans le cas des jeux à deux joueurs, cette propriété est valable si la fonction de valeur approximative satisfait une propriété de "visibilité accrue" : la précision de $\tilde{V}$ augmente lorsque l'on approche d'un état but.

Ainsi, cet algorithme simple justifie la résolution en ligne de MDP : l'effort consenti en explicitant le graphe markovien sur un horizon H permet d'obtenir une fonction de valeur en ligne améliorant $\tilde{V}$, et par conséquent la politique en ligne π_{arb} qui en est déduite.

En pratique, cet algorithme est lourd à mettre en œuvre en raison de sa complexité exponentielle et parce qu'il requiert de manipuler les distributions de probabilité p en extension. Nous allons voir dans la section suivante comment des algorithmes de recherche heuristique pour les MDP proposent une alternative à cette résolution exhaustive, bien qu'eux aussi manipulent les distributions de probabilité p en extension.

10.2.3. *Algorithmes heuristiques de recherche en ligne pour les* MDP

Depuis le milieu des années 90, des chercheurs, issus notamment de la communauté de la planification, ont proposé différents algorithmes en ligne pour exploiter efficacement l'information apportée par l'état courant dont les principaux sont l'algorithme RTDP (Real-Time Dynamic Programming) [**?**], l'algorithme de l'enveloppe [**?**], et plus récemment l'algorithme LAO^* [HAN 01].

10.2.3.1. *Principes généraux*

Ces algorithmes explicitent progressivement un sous-ensemble d'états contenant les états pouvant être atteints depuis l'état courant sans spécifier d'horizon de raisonnement. Ce sous-ensemble appelé enveloppe dans [**?**] ne contient initialement que l'état courant. Les états situés à la frontière de l'enveloppe peuvent être évalués par une fonction de valeur pré-calculée hors ligne $\tilde{V}$. Le schéma général de fonctionnement de ces algorithmes est une alternance de phases d'expansion et de phases de mises à jour :

1) phase d'expansion : un état s est choisi à la frontière de l'enveloppe et sa nouvelle valeur est calculée :

$$V(s) \leftarrow \min_{a \in A} \left[c(s,a) + \gamma \cdot \sum_{s' \in S} p(s'|s,a) \cdot \tilde{V}(s') \right]$$

2) phase de mise à jour des états pères : la valeur des états pères de l'état nouvellement développé est mise à jour selon le principe d'optimalité de Bellman.

Il faut noter toutefois que leur problématique n'est pas exactement l'amélioration d'une politique a priori pour un MDP de très grande taille et qu'ils ne se basent pas sur la simulation. Leur objectif est plutôt de proposer une alternative efficace à des algorithmes comme l'itération sur les valeurs pour des MDP raisonnablement complexes comportant quelques dizaines de milliers d'états —typiquement des problèmes de navigation de type plus court chemin stochastique. Ainsi, ils permettent de calculer beaucoup plus rapidement une action optimale pour l'état courant qu'un algorithme hors ligne qui ne converge que lorsqu'une politique optimale sur S tout entier est calculée.

Par exemple, RTDP (Real-Time Dynamic Programming) [**?**] réalise la phase d'expansion par trajectoires successives générées depuis l'état courant jusqu'à un état but. Ces trajectoires sont déterminées en choisissant pour chaque état s rencontré l'action a dont la valeur est la meilleure. L'état s' suivant s est défini comme le successeur de s via la transition stochastique simulée (s, a, s'). La valeur d'un état est mise à jour chaque fois qu'il est rencontré. La stratégie d'expansion de RTDP tend à favoriser les trajectoires les plus probables et permet d'obtenir rapidement une politique de bonne qualité. La contrepartie est une convergence lente, les états peu probables étant rarement rencontrés (cf. [BON 03]).

Ce fonctionnement général qui explicite progressivement le graphe markovien associé au MDP depuis l'état courant a inspiré certains travaux ultérieurs sur la résolution en ligne de MDP et nous présentons ci-dessous en détail un de ces algorithmes en ligne, l'algorithme LAO^* proposé par Hansen et Zilberstein [HAN 01].

10.2.3.2. *L'algorithme LAO^**

Nous avons vu qu'un MDP peut être envisagé comme un graphe "ET/OU" avec deux types de nœuds : les nœuds min et les nœuds espérance. L'algorithme AO^* [**?**] est un algorithme de recherche heuristique couramment utilisé pour ce type de problème lorsque le graphe considéré est acyclique. L'algorithme LAO^* d'Hansen et Zilberstein généralise AO^* aux MDP qui sont des graphes comportant des cycles ou boucles[2]. L'algorithme explicite progressivement le graphe markovien initialement réduit à l'état courant s_t. La version originale de l'algorithme a été développée dans le cadre d'un plus court chemin stochastique, supposant l'existence d'états but.

2. Le L de LAO^* signifie loop.

Par analogie avec les algorithmes de recherche heuristique, Hansen et Zilberstein distinguent 3 graphes :

– le graphe implicite G associé au MDP entier ;

– le graphe explicite $G' \subset G$: il contient tous les états développés depuis l'état courant s_t ;

– le graphe solution $\tilde{G} \subset G'$: c'est la restriction du graphe explicite à la meilleure *politique partielle* courante : en s_t, on sélectionne l'action $\tilde{a}$ dont l'évaluation courante est la meilleure ; on se restreint alors aux états successeurs de s_t obtenus par exécution de $\tilde{a}$; on obtient $\tilde{G}$ en itérant récursivement ce processus sur les états successeurs.

Les états non développés sont ceux situés à la frontière du graphe explicite et sont évalués par $\tilde{V}$. Si s est un état but, alors $\tilde{V}(s) = 0$.

Nous donnons ci-dessous (algorithme 10.2, figures 10.2, 10.3 et 10.4) la version de LAO^* basée sur l'itération sur les politiques[3].

Algorithme 10.2 : Algorithme LAO^*

Entrées : A,p,c, état courant s_t, fonction de valeur $\tilde{V}$
`/* G' est le graphe explicite */`
$G' \leftarrow \{s_t\}$
tant que G' *contient un état non développé qui n'est pas un état but* **faire**

(1) *Développement du graphe solution* $\tilde{G}$
(a) Choisir s dans le graphe solution qui ne soit pas un état but
(b) $V(s) \leftarrow \min_{a \in A}[c(s,a) + \sum_{s' \in S(s,a)} p(s'|s,a) \cdot \tilde{V}(s')]$
(c) $G' \leftarrow G' \cup_{a \in A(s)} S(s,a)$

(2) *Mise à jour des valeurs des actions et du graphe solution* $\tilde{G}$
- Déterminer l'ensemble Z contenant s et tous ses ancêtres dans le graphe explicite G' selon les arcs correspondant aux meilleures actions courantes, c'est-à-dire uniquement les ancêtres permettant d'atteindre s selon la meilleure politique partielle courante
- Exécuter l'algorithme 1.5 d'itération sur les politiques sur l'ensemble Z et déterminer les nouvelles meilleures actions courantes

retourner *le graphe solution* $\tilde{G}$

Convergence et efficacité de LAO^*
Si $\tilde{V}$ est *admissible* —c'est-à-dire si $\tilde{V} \leq V^*$— alors :
– $\tilde{V}(s) \leq V^*(s)$ pour chaque état s développé quand l'étape (ii) est achevée ;

3. Une version similaire basée sur l'itération sur les valeurs a également été développée.

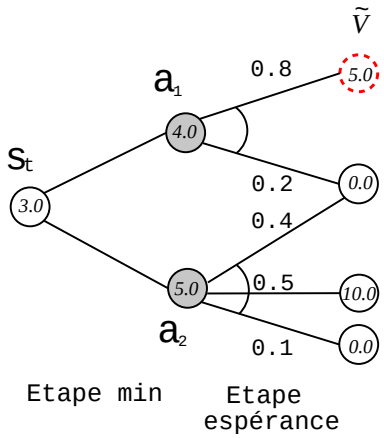


Figure 10.2. *Algorithme LAO^* - Etape (1 a) : choix d'un nœud à développer.*

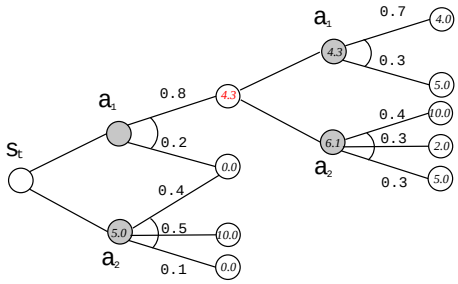


Figure 10.3. *Algorithme LAO^* - Etapes (1 b) et (1 c) : phase d'expansion.*

– $\tilde{V}(s) = V^*(s)$ lorsque LAO^* se termine ;

– LAO^* se termine après un nombre fini d'itérations.

Comme les algorithmes apparentés que sont RTDP et l'algorithme de l'enveloppe, LAO^* se révèle particulièrement efficace lorsque la politique optimale ne visite qu'une faible fraction de l'espace d'états [HAN 01]. Le calcul d'une politique optimale depuis l'état courant se fait alors beaucoup plus rapidement que par itération sur les valeurs ou itération sur les politiques.

L'un des avantages majeurs de LAO^* et des algorithmes apparentés est leur bon profil anytime : ils produisent une politique partielle éventuellement sous-optimale mais qui s'approche rapidement de la politique optimale au fur et à mesure que des états supplémentaires sont développés [BON 03]. De plus, Thiébaux et al. [**?**] notent avec justesse qu'ils peuvent être utilisés aussi bien hors ligne qu'en ligne.

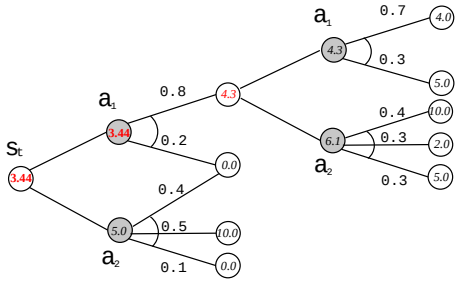


Figure 10.4. *Algorithme LAO^* - Etape (2) : phase de révision des coûts.*

Si ces algorithmes permettent de calculer efficacement une politique partielle, ils ne sont pas directement applicables à des MDP de très grande taille. Ainsi, Dean et al. [**?**] supposent que le nombre d'états successeurs d'une paire état/action reste raisonnable pour le MDP traité par leur algorithme. De plus, lorsque $\tilde{V}$ est issue d'un algorithme comme l'itération sur les politiques avec approximation, elle ne satisfait pas en général les propriétés désirables pour les fonctions heuristiques comme l'admissibilité. Nous allons voir maintenant comment la simulation peut être utilisée en ligne pour dépasser ces limites.

10.2.4. *L'algorithme par simulation de Kearns, Mansour et Ng*

Les approches précédentes ne sont exploitables que tant que les distributions de probabilité $p(s'|s,a)$ sont connues et que le nombre d'états successeurs d'une paire état/action reste raisonnable. Comme pour les méthodes hors-ligne, l'approche par simulation permet de surmonter ces difficultés. Ainsi, pour chaque paire état/action (s,a), la distribution de probabilité sur les états successeurs est implicitement[4] approximée en générant un échantillon de ces états.

Un algorithme simple, proposé par Kearns et al. [**?**], consiste à essayer chaque action N fois depuis chaque état : N fois depuis l'état courant s_t et récursivement N fois depuis chaque état qui a été généré depuis s_t sur un horizon de raisonnement H (voir l'algorithme 10.3 et la figure 10.5). Une nouvelle fonction de valeur est ainsi définie en ligne par les paramètres N et H.

Notons que lorsque $N \rightarrow +\infty$, on retrouve l'algorithme 10.1.

Après que cet arbre a été développé, l'action courante a_t est directement déduite par application de l'argmin aux valeurs des actions à la racine de l'arbre. Cet algorithme définit ainsi une politique stochastique π_{arb}.

4. Comme pour les méthodes "directes" de type Q-learning, on n'estime pas explicitement les probabilités de transition.

Algorithme 10.3 : Algorithme par simulation de Kearns, Ng et Mansour

Entrées : état courant s_t, simulateur markovien du système, horizon de raisonnement H, largeur N

$$V_{H,N}(s_t) = \begin{cases} 0 \quad \text{si } H = 0 \\ \min_{a \in A} Q_{H,N}(s_t, a) \quad \text{sinon} \\ \text{où} \quad Q_{H,N}(s_t, a) = [c(s_t, a) + \\ \gamma \frac{1}{N} \sum_{s' \in S(s_t,a,N)} V_{H-1,N}(s')] \end{cases} \tag{10.3}$$

où $S(s, a, N)$ est l'ensemble des N états qui ont été échantillonnés depuis la paire état/action (s, a)

retourner $a_t = \text{argmin}_{a \in A} Q_H(s_t, a)$

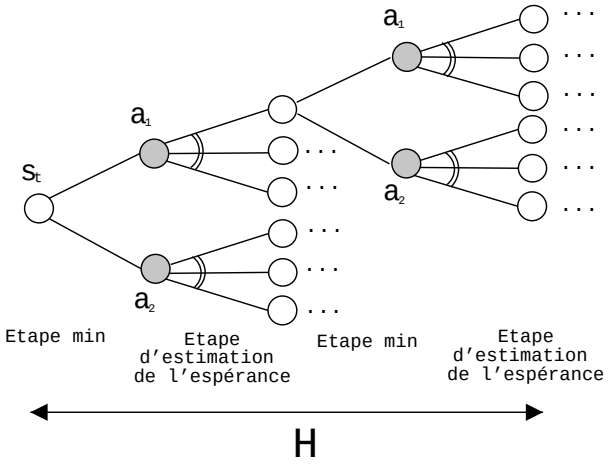


Figure 10.5. *Algorithme de Kearns, Mansour et al. avec deux actions et* $N = 3$.

10.2.4.1. *Complexité et convergence de l'algorithme de Kearns et al.*

La complexité en ligne du calcul de $V_{H,N}(s_t)$ est en $O((|A| \cdot N)^H)$. Le résultat théorique établi par Kearns et al. quantifie l'écart $|V_{\pi_{arb}}(s_t) - V^*(s_t)|$ en fonction de N et H : cet écart peut être rendu arbitrairement petit pour des valeurs de N et H suffisamment grandes et *indépendantes de la taille de l'espace d'états*.

En d'autres termes, la recherche en ligne par simulation stochastique peut théoriquement permettre d'approximer une politique optimale aussi finement que voulu avec la seule donnée d'un simulateur markovien.

Malheureusement, les valeurs requises pour N et H afin d'obtenir une garantie de convergence sont extrêmement grandes et inutilisables en pratique compte tenu de la complexité en $O((|A| \cdot N)^H)$ de l'algorithme.

10.2.4.2. *Efficacité et considérations pratiques*

Kearns et al. évoquent quelques pistes pouvant améliorer l'efficacité de leur algorithme :

– une première amélioration consiste à utiliser une fonction de valeur $\tilde{V}$ pour évaluer les feuilles de l'arbre ;

– une deuxième amélioration consiste à privilégier les nœuds proches de la racine en leur allouant plus de simulations qu'à ceux situés près des feuilles : la largeur N devient variable avec le niveau de l'arbre considéré h , $1 \leq h \leq H$;

– comme pour les algorithmes classiques de recherche arborescente, des techniques d'*élagage*[5] peuvent être envisagées [**?**].

Récemment, Chang et al. [**?**] ont proposé une autre amélioration de l'algorithme de Kearns et al. Il s'agit d'une variante en deux passes, basée sur la théorie des bandits manchots multi-bras [**?**]. Après une phase initiale d'estimation, des simulations additionnelles sont distribuées en prenant en compte l'estimation initiale de la moyenne et de la variance de la valeur des actions et des états. La complexité de l'algorithme reste toutefois en $O((|A| \cdot N)^H)$.

Le résultat de Kearns et al. prouve le bien-fondé théorique de la recherche en ligne par simulation. Il établit que, pour tout MDP, on peut déduire une politique arbitrairement proche d'une politique optimale par construction d'un arbre stochastique de largeur N et sur un horizon de raisonnement H. Toutefois, l'algorithme théorique n'est pas applicable tel quel en raison des valeurs très élevées qu'il requiert pour N et H et de sa complexité exponentielle en H. Nous verrons dans la section suivante comment développer efficacement un arbre de recherche avec des ressources de calcul raisonnables.

10.2.5. *L'algorithme Rollout de Tesauro et Galperin*

L'algorithme Rollout proposé par Tesauro et Galperin [**?**] permet d'améliorer toute politique $\tilde{\pi}$ par l'utilisation en ligne d'un simulateur.

Le principe de l'algorithme est simple : il consiste à estimer par simulation sur N trajectoires les valeurs $Q^{\tilde{\pi}}(s_t, a)$ i.e. l'espérance de l'action a suivie par l'application de la politique $\tilde{\pi}$ sur un horizon de H coups. L'action a_t de meilleure valeur est alors choisie et exécutée en ligne. De même que pour l'algorithme de Kearns et al., une politique stochastique π_{RO} est ainsi définie à partir de $\tilde{\pi}$.

5. Pruning.

Algorithme 10.4 : Algorithme Rollout

Entrées : état courant s_t, simulateur markovien du système, politique a améliorer $\tilde{\pi}$, horizon de raisonnement H, largeur N

$$\forall a \in A \qquad Q^{\tilde{\pi}}_{H,N}(s_t, a) \leftarrow c(s_t, a) + \frac{1}{N} \sum_{i=1}^{N} \sum_{l=1}^{H-1} \gamma^l c_i^l \tag{10.4}$$

où les coûts c_i^l sont générés en suivant $\tilde{\pi}$ à partir des états successeurs obtenus en exécutant a en s_t

retourner $a_t = \mathrm{argmin}_{a \in A}\, Q^{\tilde{\pi}}(s_t, a)$

L'algorithme Rollout peut être envisagé comme une étape focalisée sur s_t de l'itération sur les politiques avec approximation.

10.2.5.1. *Complexité et convergence de l'algorithme Rollout*

La complexité de l'algorithme, linéaire en N et en H, est en $O(|A|N \cdot H)$.

Chang [**?**] montre qu'un nombre suffisant de trajectoires N garantit que $V_{\pi_{RO}} \leq V^{\tilde{\pi}}$. Même si ce nombre peut être élevé, la complexité linéaire de l'algorithme le rend viable dans la plupart des applications.

10.2.5.2. *Efficacité de l'algorithme Rollout*

Remarquons que l'algorithme Rollout ne remet en cause la politique $\tilde{\pi}$ que sur la première des H transitions simulées en ligne. Pour des problèmes complexes, il est ainsi bien sûr préférable que la politique de base $\tilde{\pi}$ soit déjà de bonne qualité pour que la politique π_{RO} puisse également donner de bons résultats.

Pour limiter le nombre de simulations, Tesauro et Galperin suggèrent de contrôler continuellement l'allocation des simulations en éliminant les actions dont les valeurs estimées sont suffisamment éloignées de la valeur de la meilleure action courante. En maintenant un intervalle de confiance pour chaque action, on peut ainsi écarter les actions qui sont probablement sous-optimales — la probabilité étant spécifiée par l'utilisateur dans la définition de l'intervalle de confiance.

Bertsekas [**?**] propose une autre amélioration intéressante consistant à estimer les différences entre actions plutôt que les valeurs des actions elles-mêmes. Chang [**?**] propose une version baptisée "parallel Rollout", qui se base sur plusieurs politiques pour définir une politique en ligne. Sous certaines conditions, cette politique n'est pas moins bonne que la meilleure des différentes politiques pour chaque état rencontré en ligne.

Contrairement à l'algorithme de Kearns et al., l'algorithme Rollout a été mis en œuvre avec succès pour différents MDP de grande taille : problème d'ordonnancement de lignes de code pour un simulateur de compilateur [**?**], amélioration du niveau de

jeu du programme de backgammon TD-Gammon — alors qu'il rivalisait déjà avec les meilleurs joueurs humains — [**?**, **?**].

L'algorithme Rollout de Tesauro et Galperin propose ainsi une alternative simple à la recherche arborescente pour améliorer en ligne une politique a priori sur la base de trajectoires simulées. Cette approche est d'un usage très général mais requiert une bonne politique a priori. De plus, l'algorithme Rollout s'avère souvent très coûteux en calculs. La section suivante décrit une stratégie optimisant l'allocation des simulations pour l'algorithme Rollout.

Notons enfin que quelques autres approches en ligne ont été développées, mais dans le cadre des MDP déterministes. Ainsi, l'algorithme TD-Leaf, combine le principe de TD(λ) avec une recherche arborescente conçue pour le jeu d'échecs [**?**]. Citons également les travaux de Davies et al. [**?**] qui développent dans un cadre déterministe une approche en ligne reposant sur un algorithme de recherche de type A^*.

10.3. Contrôler la recherche

Les approches en ligne par simulation que nous venons de décrire s'avèrent en pratique assez lourdes à utiliser. En particulier, les temps de calcul pour l'évaluation d'une politique améliorée en ligne sont généralement très longs. En effet, obtenir des statistiques fiables sur une politique nécessite de simuler des centaines de trajectoires, soit typiquement des dizaines de milliers de transitions. Lorsque pour chaque transition simulée le calcul de l'action à entreprendre est non trivial, le temps d'évaluation d'une politique peut alors être de l'ordre du jour ou de la semaine.

Dans les approches en ligne par simulation, le contrôle de la recherche consiste à déterminer un ordre de développement des nœuds du graphe markovien, le but poursuivi étant de sélectionner rapidement une action de bonne qualité pour l'état courant. Par rapport aux algorithmes heuristiques de recherche pour les MDP présentés plus haut, ce problème du contrôle de la recherche est beaucoup plus complexe en raison de l'usage de la simulation. En effet, chaque état déjà explicité peut être de nouveau choisi afin d'améliorer la précision avec laquelle sa valeur est estimée. De ce fait, de nouvelles simulations peuvent être allouées non seulement aux états situés à la frontière du graphe mais également à *tous les autres états déjà explicités*.

Dans cette section, nous présentons tout d'abord une analyse du problème de contrôle de la recherche par simulation dans le cas de l'algorithme de Kearns et al. où un arbre est développé uniformément sur un horizon H, et dans le cas où la recherche s'effectue sur un horizon de 1. Cette analyse permet alors d'introduire naturellement les deux algorithmes "Focused Reinforcement Learning" et "Controlled Rollout" proposés dans [**?**].

10.3.1. *Bornes sur l'erreur et pathologie de la recherche avant*

L'utilisation pratique de l'algorithme de Kearns et al. rend nécessaire le choix de valeurs raisonnables pour l'horizon H et la largeur N, en sacrifiant les garanties d'optimalité. Se pose alors la question des principes devant guider le choix de H et de N pour obtenir de bonnes performances tout en conservant un budget de calcul raisonnable — rappelons que la complexité de l'algorithme de Kearns et al. est exponentielle en H.

Nous supposons que nous disposons d'une fonction de valeur $\tilde{V}$ vérifiant l'équation (10.1) qui sera utilisée pour évaluer les feuilles de l'arbre. Du résultat de Kearns et al., nous pouvons dériver une borne *probablement approximativement correcte*[6] sur $|V_{H,N}(s_t) - V^*(s_t)|$, qui lie la probabilité Δ de faire une erreur à l'amplitude de cette erreur :

PROPOSITION 10.2 [**?**].– *Erreur pour l'algorithme de Kearns et al. :*

$$|V_{H,N}(s_t) - V^*(s_t)| \leq \frac{c_{max}}{(1-\gamma)^2} \sqrt{\frac{1}{N} . \log \frac{(|A| \cdot N)^H}{\Delta}} + \gamma^H \epsilon$$

avec une probabilité au moins $1 - \Delta$*, où* $c_{max} = \max_{(s,a) \in S \times A} |c(s,a)|$ *est le coût instantané maximal.*

Pour un horizon fixé H, si $N \to \infty$, le premier terme de l'erreur tend vers 0 et l'on retrouve alors le résultat établi par Hernandez et Lasserre [**?**] pour l'algorithme 10.1 dans la proposition 10.1 et une erreur en $\gamma^H \epsilon$.

10.3.1.1. *Pathologie de la recherche avant par simulation*

Une stratégie naturelle de développement de l'arbre consiste à augmenter progressivement la longueur de l'horizon de raisonnement H tant que les ressources en calcul le permettent. C'est le principe des approches par approfondissement itératif[7] [**?**] couramment mises en œuvre dans les programmes de jeux pour exploiter efficacement le temps en ligne : l'horizon de raisonnement est incrémenté progressivement tant que du temps est disponible.

De manière assez surprenante, accroître l'horizon H avec une largeur N fixée finit par *augmenter* l'erreur sur $V_{H,N}$ pour l'algorithme de Kearns et al. : le premier terme d'erreur lié à l'approximation par échantillonnage croît en effet avec H. Pour une largeur N donnée, il existe ainsi une valeur optimale pour H^* au-delà de laquelle l'erreur augmente, dégradant la qualité de l'action a_t choisie (voir figure 10.6).

Ce phénomène est connu sous le nom de *pathologie de la recherche* dans le domaine des jeux à deux joueurs, pour les algorithmes de recherche arborescente basés sur le principe "minimax" [**?**]. Le coup choisi est celui qui assure la meilleure position contre toute défense de l'adversaire, la force d'une position étant estimée par

6. Probably approximately correct bound.

7. Iterative deepening.

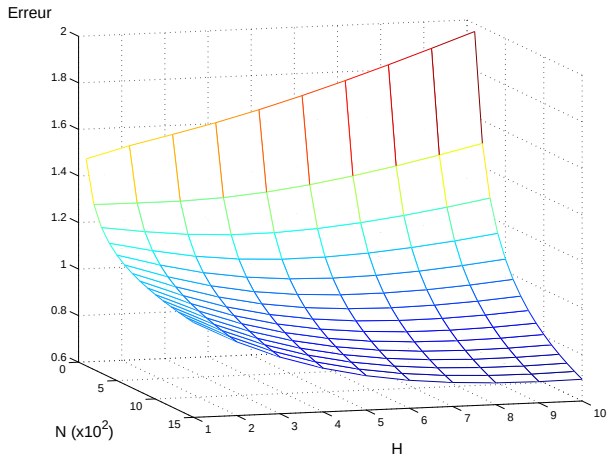


Figure 10.6. *Borne sur l'erreur* $|V_{H,C}(s_t) - V^*(s_t)|$ *en fonction de l'horizon* H *et de la largeur* N*-* $\Delta = 0.1$*,* $\epsilon = 1.0$*,* $c_{max} = 0.1$*,* $|A| = 2$*,* $\gamma = 0.8$*.*

une fonction de valeur heuristique. Ce schéma est répété sur un certain horizon de raisonnement, la fonction de valeur estimant alors les nœuds feuille de l'arbre développé. S'il est généralement admis qu'une recherche plus profonde — i.e. sur un horizon de raisonnement plus long — améliore la qualité du coup choisi, divers travaux théoriques [**?**, **?**, **?**, **?**] ont mis en évidence, qu'au contraire, une recherche plus profonde est susceptible de *dégrader* la qualité du coup choisi. Par exemple, dans le modèle simple développé par Pearl [**?**], une fonction de valeur prédit si une position sera gagnante avec un certain taux d'erreur. Cette fonction possède une propriété de "visibilité accrue" qui diminue le taux d'erreur lorsque l'on cherche plus profondément. Pearl montre alors que, même avec un fort accroissement de la visibilité, la qualité de la décision prise à la racine de l'arbre se dégrade rapidement lorsque la profondeur de la recherche augmente. Les principales raisons expliquant la pathologie de la recherche dans ces modèles de jeux à deux joueurs sont l'indépendance des valeurs heuristiques aux feuilles de l'arbre et un fort facteur de branchement.

En pratique, le phénomène n'a jamais été observé pour des jeux classiques comme les échecs. Diverses caractéristiques des jeux réels ont été avancées pour expliquer cette divergence entre théorie et pratique [**?**] : existence d'états "pièges" correspondant à des positions terminales, dépendance des valeurs heuristiques aux feuilles de l'arbre.

Plus récemment, Bulitko et al. [**?**] ont également démontré l'existence de pathologies dans le cas mono-agent. Le problème classique considéré —assimilable à un MDP déterministe— est celui de la recherche d'un plus court chemin. Bulitko et al. prouvent que la pathologie de la recherche peut survenir même lorsque la fonction de valeur heuristique est admissible. Leur démonstration reste toutefois limitée à des problèmes élémentaires très particuliers pour lesquels les valeurs heuristiques aux feuilles

de l'arbre sont très proches. Péret et Garcia [**?**] ont en revanche mis en évidence la pathologie de la recherche pour un MDP de taille moyenne de type plus court chemin stochastique.

La pathologie de la recherche ainsi mise en évidence dans le cadre des MDP est due à l'existence d'une erreur d'échantillonnage en chaque nœud de l'arbre. Dans le cas des jeux à deux joueurs, elle est inhérente aux propriétés de la fonction heuristique utilisée. La nature de l'erreur d'approximation est ainsi différente mais dans les deux cas, une recherche plus profonde l'amplifie.

10.3.1.2. *A la recherche d'un bon compromis entre profondeur et largeur*

La pathologie de la recherche nous indique qu'une recherche de type "approfondissement itératif" avec une largeur fixée n'est pas fiable dans la mesure où, au-delà d'un certain horizon, les nouveaux nœuds développés dégraderont la qualité de l'action a_t. La borne sur l'erreur établie par la proposition 10.2 peut également être interprétée en termes de compromis entre biais et variance. Le biais provient de l'erreur sur $\tilde{V}$ et est atténué par γ^H tandis que la variance provient du premier terme et décroît suivant $\sqrt{logN/N}$.

En raisonnant avec un nombre de simulations fixé, ce compromis se formule en termes de choix pour la longueur de l'horizon de raisonnement H et la largeur N. La figure 10.7 trace la borne sur l'erreur établie par la proposition 10.2 en fonction de H pour différents budgets de calcul. Le budget de calcul correspond au nombre de simulations allouées pour développer l'arbre pour H et N donnés et vaut :

$$\sum_{h=1}^{H}(|A|N)^h = \frac{(|A|N)^{H+1} - |A|N}{|A|N - 1} \simeq (|A|N)^H.$$

Ces courbes font apparaître que l'horizon optimal dépend du budget alloué.

Si l'on sait de quel budget en ligne on dispose, on peut ainsi spécifier à l'avance les valeurs de H et N. En pratique toutefois, comme c'est le cas dans les algorithmes "Focused Reinforcement Learning" et "Controlled Rollout", il peut être plus judicieux de définir une largeur locale pour chaque paire état/action, et d'incrémenter progressivement horizon et largeurs, avec le souci de conserver un bon compromis pour éviter les phénomènes pathologiques.

10.3.2. *Allocation itérative des simulations*

Nous venons de voir que le développement nécessite l'établissement d'un bon compromis global entre profondeur et largeur. Nous allons à présent nous intéresser au problème local de l'allocation des simulations entre les différentes actions pour une paire état/action donnée.

Il est possible d'améliorer l'allocation uniforme telle qu'elle est définie par l'algorithme de Kearns et al. par une approche itérative prenant en compte les informations disponibles pour les nœuds déjà développés —moyennes et variances, par exemple.

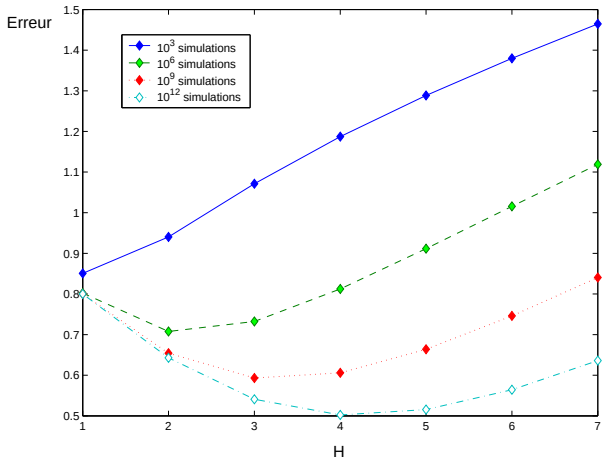


Figure 10.7. *Borne sur l'erreur* $|V_{H,C}(s_t) - V^*(s_t)|$ *en fonction de l'horizon* H *pour différents budgets de calcul.* $\Delta = 0.1$, $\epsilon = 1.0$, $c_{max} = 0.1$, $|A| = 3$, $\gamma = 0.8$.

L'objectif est de privilégier au cours du développement de l'arbre les régions les plus prometteuses. Ce problème est lié à celui du dilemme classique entre exploitation et exploration, en particulier tel qu'il apparaît en A/R.

10.3.2.1. *Bandits multi-bras et exploration dans les* MDP

Nous avons vu précédemment que les algorithmes d'A/R ne considèrent qu'une unique phase d'optimisation, habituellement qualifiée d'en ligne. Ces différents algorithmes requièrent pour converger lors de cette phase que chaque paire état/action soit exécutée infiniment souvent.

Avec un temps fini, on doit à la fois explorer suffisamment pour assurer la convergence et exploiter la politique estimée comme optimale pour minimiser les coûts reçus. Les politiques d'exploration définissant le choix de l'action à exécuter au cours de l'optimisation doivent faire face à ce dilemme entre exploration et exploitation.

La théorie des bandits multi-bras [**?**] propose une solution à ce dilemme. Un bandit à k bras est une machine à sous pour laquelle on a le choix entre k bras, chaque essai nécessitant l'introduction d'une pièce dans la machine. Chaque bras délivre un revenu aléatoire ; les revenus délivrés par les différents bras sont supposés être des variables aléatoires indépendantes dont l'espérance est inconnue. Le problème consiste alors à sélectionner rapidement le bras dont le revenu espéré est le plus haut. Plus précisément, il s'agit de déterminer une politique spécifiant quel sera le prochain bras essayé compte tenu des statistiques accumulées sur tous les bras. Divers critères d'optimisation peuvent être définis pour déterminer cette politique. Un critère classiquement

considéré est le suivant :

$$\max E\left[\sum_{t=1}^{\mathcal{T}} \gamma^t r_t\right], \tag{10.5}$$

où r_t est le revenu reçu lors de l'essai t, $\mathcal{T}$ est le nombre d'essais et γ un facteur d'actualisation. L'exploitation consiste à choisir le bras dont l'estimation courante est la meilleure tandis que l'exploration consiste à choisir un autre bras, peut-être sous-estimé.

Une politique d'exploration optimale vis-à-vis d'un critère tel que celui formulé par l'équation (10.5) peut être calculée sous diverses hypothèses concernant les distributions de probabilité des revenus et dans différents cadres formels — bayésien, non bayésien. Le livre de Berry et Fristedt [**?**] compile les résultats classiques obtenus sur le sujet.

Quelques travaux [**?**, **?**] ont étudié la possibilité de transposer la théorie des bandits multi-bras aux MDP pour concevoir des politiques d'exploration efficaces en A/R. Chang et al. [**?**] ont récemment proposé de l'appliquer à l'algorithme de Kearns (voir section 10.2.4). L'idée est de considérer un MDP comme un ensemble de $|S|$ bandits à $|A|$ bras, le revenu délivré étant le critère à long terme pour chaque état considéré.

Meuleau et Bourgine [**?**] démontrent que certaines des hypothèses requises pour obtenir une exploration optimale — indépendance de chaque bras pour un bandit donné, stationnarité de chaque bandit et indépendance de chaque bandit — ne sont pas satisfaites par les $|S|$ bandits interdépendants associés à un MDP. Néanmoins des politiques d'exploration heuristiques efficaces sont déduites du cadre des bandits multi-bras. Ces politiques sont basées sur la propagation de l'incertitude associée à chaque paire état/action.

Toutefois, le problème de l'exploration pour le développement en ligne d'un arbre se formule différemment de celui habituellement rencontré en A/R. En effet, nous ne devons pas perdre de vue que seule l'action a_t induit un coût réel. Ce coût est induit une fois que l'arbre a été développé et l'action a_t exécutée. Les coûts induits lors du développement de l'arbre ne sont que simulés et un critère comme celui formulé par l'équation (10.5) n'est donc pas approprié pour définir une politique d'exploration en ligne.

Issu du domaine de l'optimisation stochastique, ou optimisation par simulation [**?**], le cadre de l'optimisation ordinale propose par contre un critère d'optimisation plus adapté à notre problème d'exploration en ligne. L'optimisation ordinale est définie par la donnée de k candidats $\theta_1, \ldots, \theta_k$, avec $E[J(\theta_i, \omega)]$ la performance moyenne de chacun des candidats. Il s'agit de trouver :

$$\theta_{i^*} = \underset{\theta_i \in \Theta}{\operatorname{argmin}}\, E[J(\theta_i, \omega)].$$

La performance $J(\theta_i, \omega)$ est le résultat d'une simulation, et $E[J(\theta_i, \omega)]$ est typiquement estimé par la moyenne :

$$\bar{J}_i = \frac{1}{N_i} \sum_{j=1}^{N_i} J(\theta_i, \omega_{ij})$$

sur N_i réalisations pour le candidat θ_i. Pour un nombre total $N = N_1 + \cdots + N_k$ de simulations allouées à la résolution du problème, le meilleur système est alors estimé par $\theta_{\hat{i}}$, avec

$$\hat{i} = \operatorname*{argmin}_i \bar{J}_i.$$

Un critère classiquement considéré pour évaluer des règles d'allocation des simulations est alors défini par $P(SC)$, la probabilité de *sélection correcte* du système optimal θ^* parmi les k candidats :

$$P(SC) = P(\theta_b = \theta_{i^*}).$$

Le résultat principal établi à ce jour pour l'optimisation ordinale est qu'il est possible de construire des règles simples de contrôle pour distribuer N selon N_1, N_2, qui assurent une convergence exponentiellement rapide en N de $P(SC)$ vers 1, malgré le fait que la convergence des $\bar{J}_i$ vers $J(\theta_i)$ ne peut se faire au mieux qu'en $1/\sqrt{N}$. Le principe de l'optimisation ordinale repose ainsi sur ce résultat, à savoir qu'il est bien plus simple de classer les candidats que d'estimer précisément leur valeur. Différentes règles de contrôle ont été proposées dans la littérature, depuis la règle d'allocation uniforme $N_i = N/k$ jusqu'aux règles d'allocation optimale proposées par Chen et al., dont OCBA (*optimal computing budget allocation*) [**?**, **?**], qui reposent sur une approximation du critère $P(SC)$ dans un cadre bayésien et sous des hypothèses de normalité des sorties de simulation $J(\theta_i, \xi)$.

Le problème de l'exploration pour le développement en ligne d'un arbre peut ainsi être abordé comme un problème d'exploration ordinale, un candidat étant une action. Dans le cas particulier où l'arbre ne comporte qu'un unique état — autrement dit lorsque l'on considère un arbre de profondeur 1 — les deux types de problèmes sont équivalents. Des expérimentations numériques menées dans [**?**] montre en particulier sur quelques problèmes simples la supériorité de la méthode OCBA sur les méthodes d'exploration issues de l'A/R. On peut également citer les travaux de Wang et al. [**?**] qui proposent une méthode bayésienne pour la sélection des actions lors du développement de l'arbre. Cette méthode fait preuve d'une grande efficacité expérimentale sur quelques problèmes simples.

10.3.3. *Focused Reinforcement Learning*

Comme pour les algorithmes heuristiques de recherche décrits à la section 10.2.3, le schéma général de l'algorithme "Focused Reinforcement Learning" (FRL) [**?**] est une alternance de phases d'expansion de l'arbre et de phases de mises à jour.

La stratégie de contrôle consiste à suivre répétitivement des trajectoires de longueur H depuis l'état courant s_t jusqu'à un état feuille s_{t+H}. Contrairement à l'algorithme de Kearns et al., la largeur globale N n'est plus spécifiée mais dépend de la paire état/action considérée. Contrairement à l'algorithme Rollout, l'exploration n'est pas limitée au premier coup, mais à tout l'horizon H puisque les trajectoires sont guidées suivant une certaine politique d'exploration.

De plus, afin de maintenir un bon compromis global entre profondeur et largeur, l'horizon H est contrôlé dynamiquement sur la base de l'estimation de l'erreur due à la simulation. Cette estimation est une heuristique indiquant si certaines paires état/action requièrent d'avantage de simulations. L'idée est d'incrémenter H lorsque cette estimation de l'erreur est stabilisée.

10.3.3.1. *Estimation d'une erreur d'échantillonnage globale*

Nous avons vu que la recherche avant pouvait être pathologique en raison de l'amplification de l'erreur due à la simulation. Afin de contrôler l'augmentation de H, FRL estime cette erreur selon une approche heuristique décentralisée. Une telle approche consistant à propager une erreur à travers un graphe markovien a été proposée par Meuleau et Bourgine [MEU 99b] afin de concevoir des politiques d'exploration en A/R.

Pour une paire état/action donnée, l'erreur d'échantillonnage peut être estimée en utilisant un *intervalle de confiance*. Les intervalles de confiance sont un outil statistique élémentaire couramment utilisés pour quantifier l'incertitude, notamment lorsque des méthodes par simulation de type Monte-Carlo sont mises en œuvre. Pour toute paire état action (s, a) de largeur locale N, on définit ainsi l'*erreur locale* $e(s, a)$ comme :

$$e(s,a) = \sigma \frac{t_{\theta\backslash 2}^{N-1}}{\sqrt{N}},$$

où $\sigma^2 = \frac{1}{N-1} \sum_{s' \in S(s,a,N)} \big(Q(s,a) - [c(s,a) + \gamma V(s')]\big)^2$ est la variance empirique de $\gamma V(s')$, qui est l'estimation de la valeur de l'état successeur s'. $t_{\theta\backslash 2}^{N-1}$ est la fonction de Student avec $N-1$ degrés de liberté pour un niveau de confiance asymptotique $\frac{\theta}{2}$ (par exemple $\theta = 0.05$).

L'interprétation de cette erreur est la suivante : si $V(s')$ est une variable aléatoire stationnaire normalement distribuée d'espérance v alors :

$$Q(s,a) \in [c(s,a) + \gamma v - e(s,a), c(s,a) + \gamma v + e(s,a)]$$

avec une probabilité $1 - \theta$.

La variable $V(s')$ n'est pas supposée normale. De plus, comme $V(s')$ est une estimation, elle n'est pas stationnaire — son espérance évolue lorsque des nœuds supplémentaires sont développés. FRL utilise toutefois $e(s, a)$ pour estimer cette erreur

locale. Notons que cette expression quantifie uniquement l'erreur locale d'échantillonnage et n'intègre ni l'erreur due à $\tilde{V}$ ni les erreurs d'échantillonnage associées aux états successeurs de la paire état/action (s, a).

Cette erreur est alors propagée à travers l'arbre, de la même manière que les valeurs des états et des actions sont propagées dans l'équation (10.6). Ainsi, une erreur *globale* d'échantillonnage E est définie par :

$$E_{H,C}(s_t) = \begin{cases} \sigma_{\text{init}} \quad \text{si } H = 0 \\ \min_{a \in A} F_H(s_t, a) \quad \text{sinon,} \\ \text{où} \quad F_H(s, a) = [e(s_t, a) + \\ \gamma \frac{1}{C} \sum_{s' \in S(s_t, a, C)} E_{H-1,C}(s')] \end{cases} \tag{10.6}$$

où σ_{init} est une constante estimant l'erreur $\| V^* - \tilde{V} \| \infty$.

10.3.3.2. *Organiser la recherche en trajectoires successives*

L'algorithme FRL suit depuis s_t des trajectoires successives de longueur H, où H est progressivement incrémenté. La politique guidant ces trajectoires est une certaine politique d'exploration, conformément au schéma adopté par les méthodes de l'A/R. Le choix du prochain état développé ou ré-estimé est ainsi déterminé par la politique d'exploration et la dynamique propre du système.

Par rapport aux algorithmes classiques d'A/R, l'algorithme FRL se distingue sur trois points :

– le graphe markovien est explicité progressivement ; l'idée est de considérer un échantillon d'états de taille suffisante mais néanmoins raisonnable puisqu'il est mémorisé en extension, c'est-à-dire sans structure paramétrique[8] ;

– les trajectoires ont une longueur spécifiée par un horizon de raisonnement, lequel est incrémenté progressivement ;

– les politiques d'exploration les mieux fondées théoriquement sont différentes de celles considérées habituellement.

Comme les algorithmes de recherche heuristique décrits section 10.2.3, l'algorithme FRL alterne des phases d'expansion du graphe et des phases de mise à jour des valeurs des états. Comme le suggèrent les auteurs des algorithmes AO^* et LAO^*, il est en général plus efficace d'effectuer de multiples expansions avant de faire une mise à jour. De manière similaire, FRL effectue M trajectoires entre chaque mise à jour.

La procédure GénérerTrajectoires(s_t,π_{exp},H,M) génère M trajectoires de longueur H en suivant la politique π_{exp} depuis s_t. La procédure MettreAJourValeursEtats

8. La fonction de valeur $\tilde{V}$ utilisée à la frontière du graphe peut bien sûr être définie par une structure paramétrique. L'objectif de la recherche en ligne est précisément de réduire l'approximation induite par cette structure paramétrique en la "repoussant" H transitions plus loin.

Algorithme 10.5 : Algorithme Focused Reinforcement Learning

Entrées : état courant s_t, simulateur markovien du système, politique d'exploration π_{exp}, tolérance δ, taille de l'échantillon de trajectoires M
$H \leftarrow 1$
$OldE \leftarrow +\infty$
répéter
  tant que $|E_H(s_t) - OldE| > \delta$ **faire**
    GénérerTrajectoires(s_t,π_{exp},H,M)
    $OldE \leftarrow E_H(s_t)$
    MettreAJourValeursEtats()
  $H \leftarrow H + 1$
jusqu'à *condition d'arrêt*
retourner $a_t = \text{argmin}_{a \in A} Q_H(s_t, a)$

met à jour les valeurs et les erreurs des états développés et de leurs ancêtres suivant les équations (10.3) et (10.6).

La politique exploratoire π_{exp} peut être choisie parmi les méthodes d'optimisation ordinale, lesquels spécifient pour chaque action a un ratio $0 \leq \rho_a \leq 1$ pour distribuer N simulations parmi les A actions. Une manière naturelle de définir une politique d'exploration consiste alors à considérer la politique stochastique où chaque action a est choisie avec une probabilité ρ_a. Ce procédé permet de distribuer progressivement les simulations suivant les ratios ρ_a au fur et à mesure que de nouvelles trajectoires sont générées. Cette politique d'exploration ne saurait toutefois bénéficier des garanties de convergence établies pour les algorithmes d'optimisation ordinale : pour un MDP quelconque, les valeurs des actions ne sont pas normalement distribuées. De plus, elles sont non stationnaires, les valeurs estimées des actions évoluant au fur et à mesure que des simulations supplémentaires sont allouées parmi les états successeurs de l'état considéré.

10.3.3.3. *Convergence et considérations pratiques*

La convergence de l'algorithme FRL peut être établie pour un horizon H *fixé*. Le mécanisme d'incrément de l'horizon de FRL est lui d'essence heuristique, l'idée étant d'attendre d'avoir suffisamment de simulations pour l'horizon courant avant de l'incrémenter. Il est toutefois possible de déduire du résultat de Kearns et al. le nombre de trajectoires nécessaire pour obtenir une certaine borne sur l'erreur sur la fonction de valeur avec un horizon H donné (cette borne est nécessairement minorée par $\gamma^H \epsilon$).

PROPOSITION 10.3 [**?**].– *Convergence de FRL avec un horizon H fixé : soit V_H^{frl} la fonction de valeur déduite du graphe développé par M trajectoires de longueur H en suivant une politique d'exploration stochastique π_{exp}. Si pour toute paire état/action (s, a) $\pi_{exp}(s, a) > 0$ alors :*

$$V_H^{frl}(s_t) \rightarrow V_H(s_t) \textit{ presque sûrement quand } M \rightarrow +\infty,$$

où V_H est la fonction de valeur déduite de l'algorithme 10.1 de recherche avant.

La représentation utilisée par l'algorithme de Kearns et al. est un arbre, aucun test d'occurrence n'étant jamais effectué — voir figure 10.8 (a). Si elle permet de développer rapidement l'arbre, cette simplification s'avère en général peu efficace pour estimer les probabilités de transition en raison des redondances. Il est ainsi préférable d'effectuer un test d'occurrence pour les états issus d'une paire état/action donnée afin d'estimer les probabilités de transition qui lui sont attachées. Selon le principe généralement utilisé en A/R, on estime $p(s' \mid s, a)$ par $\tilde{p}(s' \mid s, a) = \frac{N^{s'}_{s,a}}{N_{s,a}}$ qui est le rapport entre le nombre de fois où l'état s' a été observé comme résultant de la paire état/action (s, a) et le nombre de fois où cette paire (s, a) a été exécutée. Chaque arc issu du nœud action associé à a est évalué par $\tilde{p}(s' \mid s, a)$, la structure d'arbre étant alors conservée — voir figure 10.8 (b). On peut également faire des tests d'occurrence parmi tous les états issus d'un état s donné. L'arbre devient alors un graphe, un nœud état pouvant avoir de multiples pères — voir figure 10.8 (c). En allant jusqu'au bout de cette démarche, on peut également faire des tests d'occurrence parmi tous les états déjà développés. Il convient de choisir la représentation la mieux adaptée en fonction des caractéristiques de l'application visée : la probabilité qu'un état soit dupliqué, le coût d'un test d'occurrence et la taille des échantillons d'états manipulés en ligne doivent guider ce choix.

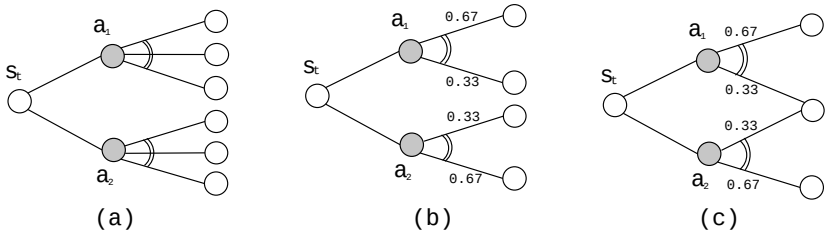


Figure 10.8. *Différentes représentations du graphe markovien explicité en ligne*

Comme Kearns et al. le relèvent, les erreurs survenant sur les nœuds situés à une faible profondeur dans l'arbre ont une plus forte incidence que ceux situés plus profondément. En conséquence, ils suggèrent que la répartition des simulations doit décroître avec la profondeur, par exemple proportionnellement à γ^{2i}, où i, $1 \leq i \leq H$ est la profondeur du nœud considéré. La stratégie par trajectoire suivie dans FRL tend à répartir les simulations proportionnellement à $\frac{1}{(|A||S(s,a)|)^i}$. Ceci tend à favoriser très fortement les nœuds les plus proches de la racine. On peut compenser ce phénomène en choisissant comme état initial d'une trajectoire un état situé à une profondeur $i > 1$. Une manière pertinente de choisir un tel état est de parcourir une trajectoire de longueur i depuis s_t : le rôle des i premières transitions simulées n'est alors plus tant d'améliorer la précision avec laquelle les nœuds proches de la racine sont estimés que

de permettre la sélection de nœuds prometteurs situés plus profondément. Au bilan, suivre continuellement des trajectoires depuis s_t permet de répartir efficacement les simulations dans l'arbre.

Nous avons vu que la valeur d'un état du graphe est estimée par $V_n(s) = \min_{a \in A} Q_{N_{s,a}}(s,a)$ où $n = \sum_{a \in A} N_{s,a}$. Lorsque les valeurs de n considérées sont faibles, la variance de cet estimateur est forte. Il peut ainsi être profitable d'introduire une pondération avec la fonction $\tilde{V}$ et de définir une règle de mise à jour incluant un taux d'apprentissage α_n. L'équation précédente est alors remplacée par $V_n(s) = \alpha_n.\tilde{V} + (1 - \alpha_n).\min_{a \in A} Q_{N_{s,a}}(s,a)$ où $0 < \alpha_n < 1$ décroît avec n. On obtient là une règle de mise à jour similaire à celles définissant les algorithmes de type Q-learning.

L'algorithme FRL n'ambitionne pas d'améliorer la fonction de valeur $\tilde{V}$ pour l'ensemble des états S à travers une structure paramétrique comme le font généralement les algorithmes d'A/R : les calculs sont effectués dans le seul but de sélectionner une bonne action pour l'état courant s_t. Il est toutefois possible de réutiliser le graphe $G_H(s_t)$ déjà développé pour sélectionner une bonne action pour s_{t+1}. Après l'exécution réelle de la transition s_t, a_t, s_{t+1}, si s_{t+1} appartient à l'ensemble $S_H(s_t)$ des états explicités dans $G_H(s_t)$, alors le graphe $G_H(s_{t+1})$ n'est plus initialement réduit à s_{t+1} et contient tous les états successeurs de s_{t+1} dans G_t. En pratique, on peut restreindre le test d'appartenance à l'ensemble $\cup_{a \in A} S(s_t, a, N_{s_t,a})$ des états successeurs de s_t dans $G_H(s_t)$. Ce principe a été appliqué à divers algorithmes de recherche de type A^*, en particulier pour des problèmes de replanification en temps réel — voir par exemple les travaux de Koenig et al. [**?**, **?**] ou ceux de Garcia [**?**].

10.3.4. *Controlled Rollout*

Nous terminons ce chapitre par la présentation d'une version améliorée de l'algorithme Rollout baptisée "Controlled Rollout" [**?**].

10.3.4.1. *Choix de l'horizon*

Comme pour l'algorithme de Kearns et al., la question du choix de l'horizon dans l'algorithme Rollout vise à établir un bon compromis entre biais et variance. Le problème considéré est toutefois différent, puisqu'une seule politique $\tilde{\pi}$ est évaluée au-delà de la première transition simulée (rappelons que la complexité de l'algorithme Rollout est linéaire en H).

Le résultat général suivant propose une borne probablement approximativement correcte sur l'erreur d'évaluation d'une politique π par N trajectoires simulées sur un horizon H. Ce résultat est formulé pour le critère γ-pondéré en horizon infini.

Proposition 1 *Soit $V^{\pi}_{H,N}(s) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{H-1} \gamma^t c_t^n$ l'estimation de $V^{\pi}(s)$ obtenue en simulant une politique π sur N trajectoires de longueur H depuis s.*

Avec une probabilité au moins égale à $1-\Delta$ *:*

$$|V^{\pi}(s) - V^{\pi}_{H,N}(s)| \leq \frac{c_{max}}{1-\gamma}\Big[\gamma^H + (1-\gamma^H)\sqrt{\frac{ln(\frac{2}{\Delta})}{2N}}\Big],$$

où c_{max} *est le coût instantané maximal.*

Il suffit ainsi pour obtenir l'horizon optimal H de minimiser numériquement (la résolution analytique débouche sur une équation du cinquième degré) cette quantité pour une largeur N donnée. La tolérance Δ peut être choisie comme égale à 0.05 par exemple.

Ce résultat est valable au-delà de l'algorithme Rollout pour évaluer une politique par simulation depuis un état donné. Un résultat analogue est établi par Kearns et Singh [**?**] pour l'algorithme TD-λ. Il est basé sur la théorie de l'analyse des grandes déviations. Toutefois, leur démonstration suppose que la politique considérée est évaluée pour chaque état de S, ce qui en limite la portée à des MDP de taille raisonnable.

10.3.4.2. *Allocation itérative des simulations*

Pour chaque état courant, l'algorithme Rollout a pour but de déterminer la meilleure des $|A|$ actions par simulation. Les méthodes d'optimisation ordinale sont directement applicables pour effectuer une allocation informée et incrémentale des simulations par un algorithme comme OCBA. Par rapport à une allocation uniforme de N simulations pour chaque action, elles permettent d'améliorer le choix de l'action courante, au prix de calculs supplémentaires à chaque itération (voir algorithme 10.6 ci-dessous).

Algorithme 10.6 : Algorithme Controlled Rollout

Entrées : état courant s_t, simulateur markovien du système, politique à améliorer $\tilde{\pi}$, nombre total de simulations C, nombre de simulations allouées à chaque itération n, tolérance Δ.
Calculer H^* minimisant l'erreur de la proposition 1 en posant $N = \frac{C}{|A|H}$ pour Δ
$l \leftarrow 1$
répéter
  Calculer les ratios ρ_a pour chaque action a selon la méthode OCBA
  Pour chaque action a, effectuer $n.\rho_a$ trajectoires supplémentaires de longueur H^* pour calculer $Q^{\tilde{\pi}}(s_t, a)$
  $l \leftarrow l + 1$
jusqu'à $nl > C$
retourner $a_t = \mathrm{argmin}_{a \in A}\, Q^{\tilde{\pi}}(s_t, a)$

L'algorithme Controlled Rollout propose ainsi une amélioration de l'algorithme Rollout en optimisant d'une part la longueur des trajectoires simulées, et d'autre part

en contrôlant l'allocation des simulations par une méthode d'optimisation ordinale (OCBA).

10.4. Conclusion

Nous avons vu dans ce chapitre les possibilités offertes par la recherche en ligne pour aborder des MDP de grande taille, en particulier par l'emploi de la simulation stochastique. Ainsi, la recherche en ligne permet de déterminer rapidement une action courante de bonne qualité et d'améliorer ce choix au fur et à mesure que des simulations supplémentaires sont allouées. La clé d'une recherche en ligne efficace réside dans la stratégie d'allocation des simulations qui doit réaliser de bons compromis entre profondeur et largeur d'une part, et entre exploitation et exploration d'autre part.

En dépit de l'intérêt qu'elles présentent, et bien que naturellement utilisées par presque tous les programmes de jeux à deux joueurs, les techniques de recherche en ligne sont encore relativement peu exploitées dans le cadre des MDP. Ainsi, la grande majorité des travaux sur la résolution des MDP de grande taille se concentre sur le calcul hors ligne d'une politique ou une d'une fonction de valeur approximative. La recherche en ligne peut pourtant permettre d'améliorer significativement la performance d'une politique, au prix d'un effort calculatoire certes important.

Si les travaux de Tesauro [**?**] ont ouvert la voie, peu de chercheurs l'ont empruntée. Outre les travaux présentés ci-dessus, on peut toutefois noter un récent regain d'intérêt pour les méthodes en ligne pour les MDP, et en particulier pour les méthodes de recherche basées sur la simulation stochastique — par exemple les travaux de Chang [**?**, **?**] ou de Bent et Van Hentenryck [**?**] appliqués au routage de paquets dans les réseaux de télécommunications. Citons enfin l'algorithme UCT de Kocsis et Szepesvari [**?**] fondé sur la théorie des bandits manchots et qui s'est avéré expérimentalement sur différents problèmes incluant le jeu de Go [**?**].

Chapitre 11

Programmation dynamique avec approximation de la fonction de valeur

L'utilisation d'outils pour l'approximation de la fonction de valeur est essentielle pour pouvoir traiter des problèmes de prise de décisions séquentielles de grande taille. Les méthodes de programmation dynamique (PD) et d'apprentissage par renforcement (A/R) introduites aux chapitres 1 et 2 supposent que la fonction de valeur peut être représentée (mémorisée) en attribuant une valeur à chaque état (dont le nombre est supposé fini), par exemple sous la forme d'un tableau. Ces méthodes de résolution, dites « exactes », permettent de déterminer la solution optimale du problème considéré (ou tout au moins de converger vers cette solution optimale). Cependant, elles ne s'appliquent souvent qu'à des problèmes jouets, car pour la plupart des applications intéressantes, le nombre d'états possibles est si grand (voire infini dans le cas d'espaces continus) qu'une représentation exacte de la fonction ne peut être parfaitement mémorisée. Il devient alors nécessaire de représenter la fonction de valeur, de manière approchée, à l'aide d'un nombre modéré de coefficients, et de redéfinir et analyser des méthodes de résolution, dites « approchées » pour la PD et l'A/R, afin de prendre en compte les conséquences de l'utilisation de telles approximations dans les problèmes de prise de décisions séquentielles.

EXEMPLE.– Revenons, une fois de plus, au cas de la voiture dont on veut optimiser les opérations d'entretien (voir tome 1, section 1.1). Nous avons vu que le nombre d'états possibles de la voiture pouvait être très grand, bien qu'il puisse parfois être factorisé (voir chapitre 9, page 275). Mais, même si on s'intéresse à un seul élément de la voiture, les freins par exemple, le nombre d'états peut être infini : il suffit de caractériser l'état d'usure des freins par l'épaisseur des plaquettes de frein. L'état de

---

Chapitre rédigé par Rémi MUNOS.

la voiture peut varier de façon continue dans un intervalle donné et toutes les méthodes vues précédemment sont alors inutiles car elles font l'hypothèse que l'espace d'état est fini. Les formalismes abordés dans ce chapitre permettent de prendre en compte cette problématique d'état continu (comme le montre explicitement l'exemple de la section 11.2.3) et, par la même, de gérer des problèmes avec un grand nombre d'état.

Dans ce chapitre, nous étudions l'utilisation d'outils d'approximation de fonctions (domaine appelé apprentissage supervisé, apprentissage statistique, ou encore théorie de l'approximation, selon diverses communautés scientifiques) pour la représentation de la fonction de valeur et généralisons les méthodes de PD et d'A/R précédemment étudiées dans ce cadre de résolution approchée. Des bornes sur la perte en performance résultante de l'utilisation de représentations approchées sont établies en fonction de la capacité et la richesse d'approximation des espace d'approximation considérés.

11.1. Introduction

Le désir d'utiliser des fonctions approchées en programmation dynamique (PD) remonte aux origines mêmes de ce domaine. Par exemple Samuel [SAM 67] a utilisé des approximations linéaires de la fonction de valeur pour le jeu de dames, Bellman et Dreyfus [BEL 59] des polynômes afin d'accélérer la PD. Une première analyse théorique est entreprise dans [REE 77]. Plus récemment, l'apprentissage par renforcement (A/R) et la PD combinés à des représentations de fonctions ont permis de résoudre avec succès plusieurs applications de taille réelle ; par exemple le programme *TD-Gammon* [TES 95] utilisant une représentation à l'aide d'un réseau de neurones, a produit un programme champion du monde de backgammon. D'autres exemples d'applications concernent la recherche opérationnelle et l'ordonnancement de tâches [ZHA 95], le contrôle d'une batterie d'ascenseurs [CRI 97], la maintenance de machines en usine [MAH 97], la répartition dynamique de canaux téléphoniques [SIN 97], l'allocation de sièges dans les avions [GOS 04]. Quelques applications spécifiques sont décrites plus précisément dans les parties 2 et 4 de cet ouvrage.

L'objet de ce chapitre est de présenter de manière succincte les problématiques nouvelles d'une *résolution approchée* de problèmes de PD et d'A/R ainsi que des bornes sur la perte en performance lorsque l'on utilise une approximation de la fonction de valeur.

Par souci de simplicité de la présentation, nous considérons ici uniquement le cas d'un *critère actualisé (ou γ-pondéré) sur un horizon temporel infini*. Ainsi, pour une politique stationnaire fixée π (fonction qui associe une action à chaque état), la fonction de valeur V^π est définie par l'espérance de la somme des récompenses actualisées à venir :

$$V^\pi(s) \stackrel{\text{def}}{=} \mathbb{E}\Big[\sum_{t\geq 0} \gamma^t r(s_t, \pi(s_t)) | s_0 = s, \pi\Big]. \tag{11.1}$$

D'autres critères (horizon temporel fini ou problèmes non actualisés) font l'objet d'une analyse conceptuelle similaire, mais présentent des différences dans le formalisme mathématique. Pour ces extensions, nous renvoyons le lecteur au travail de synthèse de Bertsekas et Tsitsiklis [BER 96].

L'approche suivie dans ce chapitre consiste à déterminer une approximation de la fonction de valeur optimale, avec l'espoir que la performance de la politique gloutonne déduite d'une telle approximation soit proche de la performance optimale. Cet espoir est justifié par le fait que si V est une bonne approximation de la fonction de valeur optimale V^*, alors la performance V^π de la politique π gloutonne par rapport à V est proche de l'optimum V^*. Rappelons que l'on dit qu'une politique π est gloutonne par rapport à V (on dit aussi que π est déduite de V) si, en tout état $s \in S$, l'action $\pi(s)$ est une action qui maximise la récompense immédiate plus l'espérance (actualisée) de V à l'état suivant, c'est-à-dire,

$$\pi(s) \in \arg\max_{a \in A} \big[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s') \big].$$

En effet, on a le résultat suivant [BER 96] (p.262) :

PROPOSITION 11.1.– *Soit V une fonction à valeurs réelles définie sur S et π une politique gloutonne par rapport à V. Alors la perte en performance résultante de l'utilisation de la politique π (au lieu de la politique optimale), écart entre la fonction de valeur V^π et la fonction de valeur optimale V^*, est majorée selon*

$$||V^* - V^\pi||_\infty \leq \frac{2\gamma}{1-\gamma} ||V - V^*||_\infty, \tag{11.2}$$

où $||\cdot||_\infty$ est la norme sup, notée L^∞ (i.e. $||f||_\infty \stackrel{\text{def}}{=} \max_{s \in S} |f(s)|$), et γ le coefficient d'actualisation.

Remarquons que, pour toute fonction V, la fonction de valeur V^π n'a aucune raison d'être égale à V. Cette majoration donne une justification à notre approche qui consiste à chercher une bonne approximation de la fonction de valeur optimale ($||V - V^*||$ petit) afin d'en déduire une politique dont la performance est proche de l'optimum ($||V^* - V^\pi||$ petit). La preuve étant élémentaire, on l'inclut maintenant.

PREUVE.– On rappelle la définition des opérateurs de Bellman L et L_π (définis au chapitre 1, section 1.5.2) : pour toute fonction W à valeurs réelles définie sur S,

$$LW(s) \stackrel{\text{def}}{=} \max_{a \in A} \big[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) W(s') \big],$$

$$L_\pi W(s) \stackrel{\text{def}}{=} r(s,\pi(s)) + \gamma \sum_{s' \in S} p(s'|s,\pi(s)) W(s').$$

L_π peut aussi s'écrire (en notation vectorielle) $L_\pi W = r_\pi + \gamma P_\pi W$, où P_π est la matrice de transition pour la politique π, dont les éléments (s,s') sont $p(s'|s,\pi(s))$, et r_π le vecteur de composantes $r(s,\pi(s))$.

Une propriété des opérateurs L et L_π est qu'ils sont des contractions en norme L^∞ avec un coefficient de contraction γ. Cela signifie que, pour tout couple de fonctions W_1 et W_2 définies sur S, on a $||LW_1 - LW_2||_\infty \leq \gamma||W_1 - W_2||_\infty$ et de même pour l'opérateur L_π. Nous laissons la démonstration (simple) de cette propriété au lecteur.

Du fait que V^* et V^π sont respectivement les points fixes des opérateurs L et L_π (i.e. $V^* = LV^*$ et $V^\pi = L_\pi V^\pi$), en utilisant l'inégalité triangulaire, il vient :

$$\begin{aligned} ||V^* - V^\pi||_\infty &\leq ||LV^* - L_\pi V||_\infty + ||L_\pi V - L_\pi V^\pi||_\infty \\ &= ||LV^* - LV||_\infty + \gamma||V - V^\pi||_\infty \\ &\leq \gamma||V^* - V||_\infty + \gamma(||V - V^*||_\infty + ||V^* - V^\pi||_\infty), \end{aligned}$$

où la seconde ligne utilise la définition que π est gloutonne par rapport à V, i.e. $LV = L_\pi V$. On en déduit la majoration sur l'écart entre la performance V^π de la politique π gloutonne par rapport à V, et la performance optimale V^* :

$$||V^* - V^\pi||_\infty \leq \frac{2\gamma}{1-\gamma}||V^* - V||_\infty.$$

□

Afin de construire une bonne approximation de la fonction de valeur optimale, nous généralisons les algorithmes de PD et d'A/R vus aux chapitres précédents. Nous commençons par présenter l'algorithme d'itérations sur les valeurs lorsque l'on utilise une représentation approchée de la fonction de valeur. Nous abordons ensuite dans les sections 11.3 et 11.4 l'algorithme d'itérations sur les politiques et les méthodes de minimisation du résidu de Bellman. Puis, à la section 11.5, nous expliquons les limites de l'analyse en norme L^∞ de la programmation dynamique et explorons une extension à une analyse en norme L^p (pour $p \geq 1$) afin d'établir des premiers liens avec le domaine de l'apprentissage statistique et de fournir des bornes en fonction de la capacité et la richesse des espaces d'approximation considérés.

11.2. Itérations sur les valeurs avec approximation (IVA)

A nouveau, nous considérons la résolution d'un processus décisionnel de Markov (MDP) [PUT 94] utilisant un critère actualisé avec horizon temporel infini. L'espace des états est supposé grand (mais fini pour l'instant, afin de simplifier les notations).

Comme on l'a vu au chapitre 1, l'algorithme d'*itérations sur les valeurs* consiste à calculer la fonction de valeur optimale V^* par évaluation successive de fonctions V_n selon le schéma d'itérations $V_{n+1} = LV_n$, où L est l'opérateur de Bellman défini précédemment. Grâce à la propriété de contraction (en norme L^∞) de l'opérateur L, les itérés V_n convergent vers V^* (le point fixe de L) lorsque $n \to \infty$ (car on a $||V_{n+1} - V^*||_\infty = ||TV_n - TV^*||_\infty \leq \gamma||V_n - V^*||_\infty \leq \gamma^n||V_1 - V^*||_\infty$).

Lorsque le nombre d'états est tel qu'une représentation exacte des fonctions V_n est impossible à mémoriser, nous devons considérer une méthode de résolution utilisant des représentations approchées des V_n ; ce qui nous mène à définir l'algorithme d'*itérations sur les valeurs avec approximation* (IVA). IVA est très populaire et est depuis longtemps implémenté de diverses manières en programmation dynamique [SAM 59, BEL 59] et plus récemment dans le contexte de l'apprentissage par renforcement [BER 96, SUT 98], par exemple le *fitted Q-iteration* [ERN 05] qui construit une approximation de la fonction de valeur d'action optimale, à partir d'observations de transitions.

Notons $\mathcal{F}$ l'espace des fonctions représentables considéré. Par exemple, $\mathcal{F}$ peut être un sous-espace vectoriel engendré par un ensemble fini de fonctions génératrices (appelées *features*). Toute fonction de $\mathcal{F}$ se définit alors par un nombre fini de coefficients, comme la combinaison linéaire des fonctions génératrices pondérées par ces coefficients. On parle alors d'*approximation linéaire*.

L'algorithme IVA construit une séquence de représentations $V_n \in \mathcal{F}$ calculées selon les itérations :

$$V_{n+1} = \mathcal{A}LV_n, \tag{11.3}$$

où L est l'opérateur de Bellman et $\mathcal{A}$ un *opérateur d'approximation* par des fonctions de $\mathcal{F}$. Par exemple, dans le cas de l'approximation linéaire, $\mathcal{A}$ est un opérateur de projection orthogonale sur $\mathcal{F}$, c'est à dire : $\mathcal{A}f \in \mathcal{F}$ est la fonction de $\mathcal{F}$ qui réalise la distance minimale à f : $||\mathcal{A}f - f|| = \inf_{g\in\mathcal{F}} ||g - f||$ (pour une certaine norme issue d'un produit scalaire).

Ainsi, IVA consiste en une séquence d'itérations où à chaque étape, une nouvelle représentation $V_{n+1} \in \mathcal{F}$ s'obtient en projetant sur $\mathcal{F}$ l'image par l'opérateur de Bellman L de l'estimation précédente V_n. L'itération (11.3) est illustrée sur la figure 11.1.

Lorsque l'opération d'approximation est réalisée à partir de données (échantillons) (par exemple pour une projection minimisant une erreur empirique), on parle alors d'*apprentissage supervisé* ou de *régression* (voir par exemple [HAS 01]) ; ce cas est illustré au paragraphe suivant.

11.2.1. *Implémentation à partir d'échantillons et apprentissage supervisé*

A titre d'illustration, IVA utilisant une version échantillonnée pour l'étape de projection est définie de la manière suivante : à l'étape n, on choisit K états $(s_k)_{1\leq k\leq K}$ tirés de manière indépendante selon une certaine distribution μ sur l'espace d'états S. On calcule l'image, par l'opérateur de Bellman, de V_n en ces états, définissant ainsi les valeurs $\{v_k \stackrel{\text{def}}{=} LV_n(s_k)\}$. Puis on fait appel à un algorithme d'apprentissage supervisé avec pour données d'apprentissage les couples (entrées, sorties désirées) :

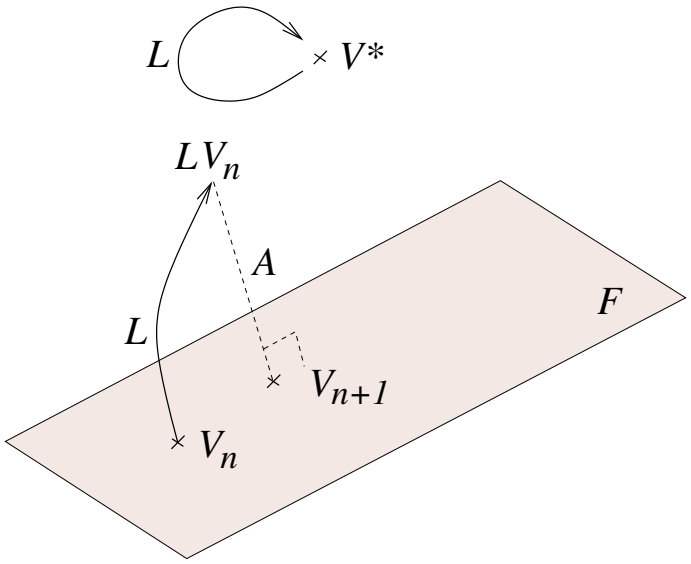


Figure 11.1. *Représentation schématique d'une itération de l'algorithme IVA : l'espace d'approximation $\mathcal{F}$ est un sous-espace vectoriel de dimension finie. $V_n \in \mathcal{F}$ représente l'approximation à l'instant n. L'opérateur de Bellman L est appliqué à V_n (LV_n n'a pas de raison d'appartenir à $\mathcal{F}$, c'est à dire d'être représentable dans cette architecture d'approximation) puis la projection $\mathcal{A}$ sur $\mathcal{F}$ définit la représentation suivante $V_{n+1} = \mathcal{A}LV_n$. La fonction de valeur optimale V^* (point fixe de L) est aussi représentée.*

$\{(s_k, v_k)\}_{1 \leq k \leq K}$. Ce dernier retourne une fonction $V_{n+1} \in \mathcal{F}$ qui minimise une erreur empirique, par exemple :

$$V_{n+1} = \arg\min_{f \in \mathcal{F}} \frac{1}{K} \sum_{1 \leq k \leq K} \big(f(s_k) - v_k\big)^2. \tag{11.4}$$

Cette fonctionnelle utilise une norme quadratique L^2, comme cela est le cas pour les méthodes de moindres carrés, la régression linéaire locale, les réseaux de neurones, et bien d'autres algorithmes d'apprentissage supervisé. Bien entendu, d'autres fonctionnelles utilisant d'autres normes, comme la norme L^1 (valeur absolue) ou des variantes (par exemple la norme L^1 ϵ-insensible utilisée dans les SVM [VAP 98]) ainsi que des fonctionnelles pénalisées sont souvent utilisées. Ce problème de régression est un cas particulier d'apprentissage supervisé (ou apprentissage statistique). Nous n'abordons pas dans ce chapitre les enjeux importants de ce domaine, et renvoyons le lecteur intéressé aux références usuelles, par exemple [HAS 01].

Mentionnons seulement que *l'approximation linéaire* consiste à réaliser une projection sur un espace vectoriel engendré par une famille finie de fonctions données, et

inclut les décompositions sur des splines, fonctions radiales, bases de Fourier ou ondelettes. Cependant, une meilleure régression est souvent obtenue lorsque la famille de fonctions engendrant l'espace sur laquelle est faite la projection est choisie en fonction des régularités de la fonction à approcher. On parle alors d'*approximation non-linéaire*, celle-ci pouvant être particulièrement efficace quand la fonction recherchée possède des régularités locales (par exemple, dans des bases d'ondelettes adaptatives [MAL 97], de telles fonctions peuvent être représentées de manière compacte avec peu de coefficients non-nuls). Des algorithmes gloutons, comme le *matching pursuit* et diverses variantes [DAV 97] sélectionnent les meilleures fonctions de base dans un dictionnaire donné. La théorie de l'approximation étudie les erreurs d'approximation en fonction des régularités de la fonction cible [DEV 97]. En apprentissage statistique, d'autres outils d'approximation non-linéaire très souvent utilisés sont les *réseaux de neurones, la régression linéaire locale* [ATK 97], *les machines à vecteurs de support* et les méthodes à noyaux dans les *espaces de Hilbert à noyaux reproduisant* [VAP 97, VAP 98].

11.2.2. *Analyse de l'algorithme IVA*

Considérons l'algorithme IVA défini par l'itération (11.3) et définissons

$$\varepsilon_n \stackrel{\text{def}}{=} LV_n - V_{n+1} \tag{11.5}$$

l'*erreur d'approximation* à l'étape n. En général, IVA ne converge pas vers la fonction de valeur optimale V^* (contrairement à l'algorithme d'itérations sur les valeurs) car V^* n'a a priori aucune raison d'appartenir à l'espace de représentation $\mathcal{F}$. De plus, même si $V^* \in \mathcal{F}$, on ne dispose en général d'aucune garantie que les itérés V_n convergent vers V^*. En réalité, IVA peut osciller voire diverger, comme cela est illustré sur des exemples simples décrits dans [BAI 95, TSI 96a] et [BER 96] (p. 334). Pourtant, cet algorithme est très populaire car il a souvent produit de bons résultats d'un point de vue expérimental.

Pour comprendre les raisons qui mènent à des comportements si variés selon les applications, on désire analyser le comportement de IVA et établir des bornes sur la performance de cet algorithme. Un premier résultat établit une majoration sur la perte en performance (par rapport à la performance optimale) résultante de l'utilisation de la politique π_n gloutonne par rapport à V_n, en fonction des erreurs d'approximation ε_n commises à chaque étape.

PROPOSITION 11.2 [BER 96].– *En notant π_n la politique gloutonne par rapport à l'approximation V_n, et V^{π_n} la fonction de valeur associée à cette politique, on a :*

$$\limsup_{n\to\infty} ||V^* - V^{\pi_n}||_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \limsup_{n\to\infty} ||\varepsilon_n||_\infty. \tag{11.6}$$

PREUVE.– D'après (11.2) appliqué à V_n, nous déduisons

$$||V^* - V^{\pi_n}||_\infty \leq \frac{2\gamma}{1-\gamma}||V^* - V_n||_\infty. \tag{11.7}$$

De plus,

$$\begin{aligned} ||V^* - V_{n+1}||_\infty &\leq ||LV^* - LV_n||_\infty + ||LV_n - V_{n+1}||_\infty \\ &\leq \gamma||V^* - V_n||_\infty + ||\varepsilon_n||_\infty. \end{aligned}$$

Maintenant, en passant à la limite supérieure, il vient

$$\limsup_{n\to\infty} ||V^* - V_n||_\infty \leq \frac{1}{1-\gamma} \limsup_{n\to\infty} ||\varepsilon_n||_\infty,$$

ce qui, combiné à (11.7), mène à (11.6). □

Remarquons que cette borne s'exprime en fonction de la norme L^∞ des erreurs d'approximation ε_n, c'est-à-dire qu'elle dépend de la plus grande erreur $\varepsilon_n(s)$ sur tout le domaine (lorsque s parcourt S). Cette erreur uniforme est en général difficile à maîtriser, particulièrement pour des problèmes de grande taille. De plus, elle n'est pas très utile en pratique puisque la plupart des méthodes d'approximation de fonction et algorithmes d'apprentissage supervisé résolvent, comme on l'a illustré au paragraphe 11.2.1, un problème de minimisation empirique en norme L^2 ou L^1. Nous verrons à la section 11.5 une extension de la proposition 11.2 à des normes L^p (pour $p \geq 1$). Mentionnons toutefois l'existence de travaux [GOR 95, GUE 01a] portant sur des approximations de fonction utilisant la norme L^∞ tels les *averagers* (implémentés par exemple par la méthode des *k-plus proches voisins*) dans le cadre de la PD.

11.2.3. *Illustration numérique*

Dans cette partie nous illustrons le fonctionnement de l'algorithme IVA pour un problème de remplacement optimal, inspiré de [RUS 96]. Nous montrons aussi sur cet exemple le fait que les résultats précédents se généralisent naturellement au cas où l'espace d'états est continu.

Une variable mono-dimensionnelle $s_t \in S \stackrel{\text{def}}{=} [0, s_{\max}]$ mesure l'utilisation accumulée d'un certain produit (par exemple le compteur kilométrique d'une voiture mesure l'état d'usure de celle-ci). $s_t = 0$ désigne un produit tout neuf. A chaque instant discret t (par exemple tous les ans), il y a deux décisions possibles : soit conserver (a_t = C), soit remplacer (a_t = R) le produit, auquel cas, un coût supplémentaire $C_{remplace}$ (de vente du produit courant suivi du rachat d'un nouveau bien) est perçu.

On suppose que les transitions suivent une loi exponentielle de paramètre β avec une queue tronquée : si l'état suivant y est plus grand qu'une valeur maximale fixée

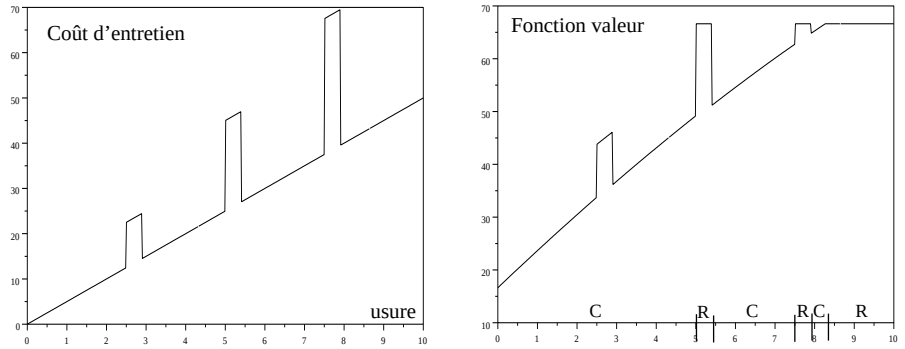


Figure 11.2. *Fonctions coût d'entretien immédiat c et fonction de valeur optimale V^*. L'indication R et C sur la figure de droite indique la politique optimale (gloutonne par rapport à V^*) en fonction de l'état.*

$s_{\max}$ (par exemple un état critique d'usure de la voiture) alors un nouvel état est immédiatement tiré et une pénalité $C_{mort} > C_{remplace}$ est reçue. Ainsi, en notant $p(\cdot|s,a)$, la densité de probabilité de l'état suivant sachant que l'état courant est s et l'action choisie est $a \in \{\mathrm{C}, \mathrm{R}\}$ (c'est-à-dire que pour tout ensemble $B \subset S$, la probabilité d'être dans B à l'instant suivant est $\int_B p(ds'|s,a)$), on définit :

$$p(s'|s,\mathrm{R}) \stackrel{\text{def}}{=} \begin{cases} q(s') & \text{si } s' \in [0, s_{\max}] \\ 0 & \text{sinon.} \end{cases}$$

$$p(s'|s,\mathrm{C}) \stackrel{\text{def}}{=} \begin{cases} q(s'-s) & \text{si } s' \in [s, s_{\max}] \\ q(s'-s+s_{\max}) & \text{si } s' \in [0, s) \\ 0 & \text{sinon.} \end{cases}$$

avec $q(s) \stackrel{\text{def}}{=} \beta e^{-\beta s}/(1-e^{-\beta s_{\max}})$ (densité exponentielle tronquée).

Le coût immédiat (opposé d'une récompense) $c(s)$ est la somme d'une fonction monotone lentement croissante (qui correspond par exemple à des coûts de maintenance) et d'une fonction coût ponctuellement discontinue (par exemple les coûts de révision). Le coût immédiat et la fonction de valeur optimale (calculée de manière analytique) sont représentées sur la figure 11.2 pour les valeurs numériques suivantes : $\gamma = 0.6$, $\beta = 0.6$, $C_{remplace} = 50$, $C_{mort} = 70$ et $s_{\max} = 10$.

Nous considérons l'implémentation de l'algorithme IVA à base d'échantillons décrite au paragraphe 11.2.1. Les états considérés $\{s_k \stackrel{\text{def}}{=} k s_{\max}/K\}_{0 \le k < K}$ (avec $K = 200$) sont uniformément répartis sur le domaine S. L'espace fonctionnel d'approximation considéré $\mathcal{F}$ est l'espace vectoriel de dimension $M = 20$ engendré par

une famille de cosinus :

$$\mathcal{F} \stackrel{\text{def}}{=} \left\{ V_\alpha(s) \stackrel{\text{def}}{=} \sum_{m=1}^{M} \alpha_m \cos(m\pi \frac{s}{s_{\max}}) \right\}_{\alpha \in \mathbb{R}^M}.$$

Ainsi, à chaque itération n, la nouvelle approximation $V_{n+1} \in \mathcal{F}$ est obtenue en résolvant le problème de régression quadratique :

$$V_{n+1} = \arg\min_{f \in \mathcal{F}} \frac{1}{K} \sum_{k=1}^{K} \left[f(s_k) - LV_n(s_k) \right]^2.$$

Nous commençons avec une fonction de valeur initiale $V_0 = 0$. La figure 11.3 représente la première itération : les valeurs itérées $\{LV_0(s_k)\}_{1 \leq k \leq K}$ par l'opérateur de Bellman (indiquées par les croix sur la figure de gauche) et la régression correspondante $V_1 \in \mathcal{F}$ (meilleure approximation de LV_0 dans l'espace $\mathcal{F}$). La figure 11.4 illustre de manière analogue la seconde itération. La figure 11.5 montre la fonction de valeur approchée $V_{20} \in \mathcal{F}$ obtenue après 20 itérations. Dans cette simulation, l'algorithme IVA fonctionne bien et une bonne approximation de la fonction de valeur optimale V^* est obtenue dans $\mathcal{F}$.

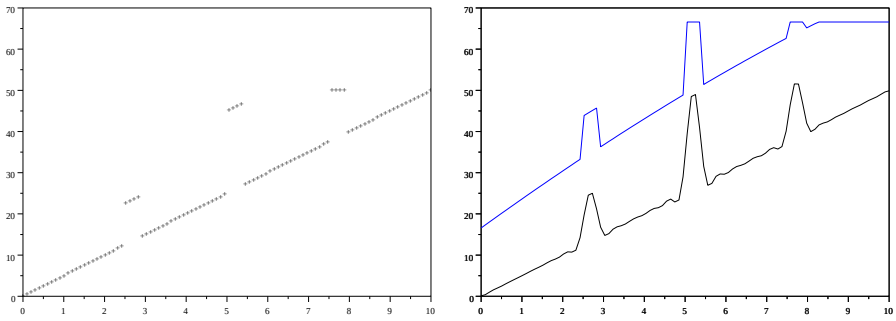


Figure 11.3. *Valeurs itérées $\{LV_0(s_k)\}_{1 \leq k \leq K}$ (figure de gauche), meilleure approximation $V_1 \in \mathcal{F}$ de LV_0, ainsi que la fonction de valeur optimale (figure de droite).*

11.3. Itérations sur les politiques avec approximation (IPA)

Nous considérons maintenant l'algorithme d'*itérations sur les politiques avec approximation* (IPA) [BER 96] généralisant l'algorithme d'itérations sur les politiques vu au chapitre 1 au cas où la fonction de valeur est représentée de manière approchée. Cet algorithme est défini par la répétition des deux étapes suivantes :

– *Évaluation approchée de la politique* : pour une politique π_n, on calcule une approximation V_n de la fonction de valeur V^{π_n} ;

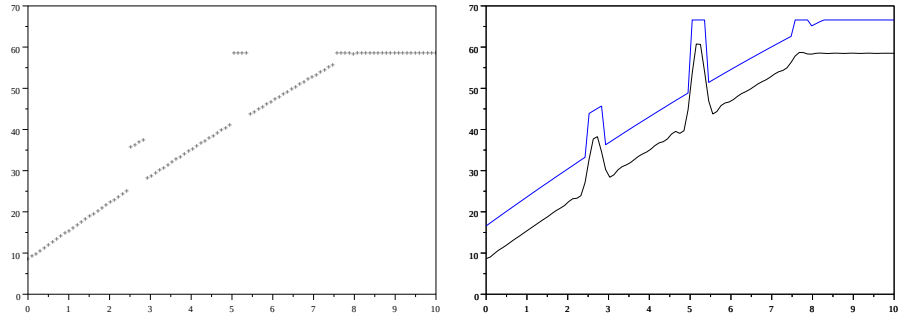


Figure 11.4. *Valeurs itérées* $\{LV_1(s_k)\}_{1\leq k\leq K}$ *(figure de gauche), meilleure approximation* $V_2 \in \mathcal{F}$ *de* LV_1*, ainsi que la fonction de valeur optimale (figure de droite).*

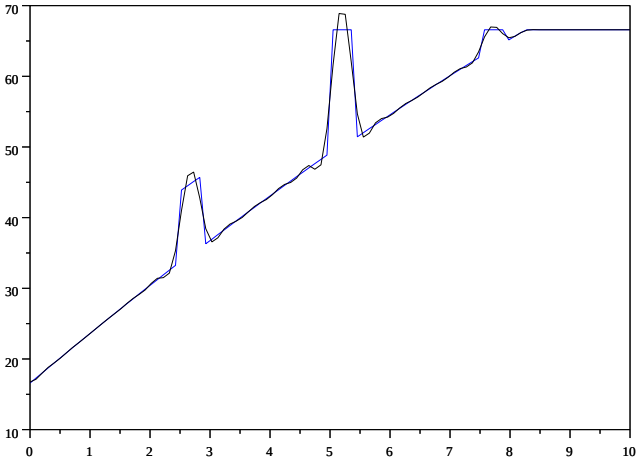


Figure 11.5. *Approximation* $V_{20} \in \mathcal{F}$ *obtenue à la* 20*e itération et la fonction de valeur optimale.*

– *Amélioration de la politique* : on génère une nouvelle politique π_{n+1} gloutonne par rapport à V_n, c'est-à-dire telle que, en tout $s \in S$,

$$\pi_{n+1}(s) \in \arg\max_{a\in A} \left[r(s,a) + \gamma \sum_{s'\in S} p(s'|s,a)V_n(s)\right]. \tag{11.8}$$

La section 11.3.1 fournit un résultat de majoration sur la perte en performance $||V^{\pi_n} - V^*||_\infty$ résultant de l'utilisation des politiques générées par IPA au lieu de la politique optimale, en fonction des erreurs d'approximation des fonctions de valeur $||V_n - V^{\pi_n}||_\infty$. La section 11.3.2 décrit l'étape d'évaluation approchée des politiques et la section 11.3.3 détaille le cas de l'*approximation linéaire* avec une extension de

l'algorithme TD(λ), des méthodes de type moindres carrés et enfin une implémentation en termes de fonction de valeur d'action ne nécessitant pas la connaissance a priori des probabilités de transition.

11.3.1. *Analyse de l'algorithme IPA en norme* L^∞

Notons π_n la politique obtenue par l'algorithme IPA à l'itération n. Soit V_n une approximation de la fonction de valeur V^{π_n} et π_{n+1} une politique gloutonne par rapport à V_n, au sens défini par (11.8). Le résultat suivant (voir par exemple [BER 96], p. 276) fournit une majoration d'erreur en norme L^∞ sur la perte $V^* - V^{\pi_n}$ résultant de l'utilisation de la politique π_n au lieu de la politique optimale, en fonction des *erreurs d'approximation* $V_n - V^{\pi_n}$.

PROPOSITION 11.3.– *Nous avons :*

$$\limsup_{n\to\infty} ||V^* - V^{\pi_n}||_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \limsup_{n\to\infty} ||V_n - V^{\pi_n}||_\infty. \qquad (11.9)$$

PREUVE.– Définissons $e_n \stackrel{\text{def}}{=} V_n - V^{\pi_n}$ l'*erreur d'approximation* de V^{π_n} par V_n, $g_n \stackrel{\text{def}}{=} V^{\pi_{n+1}} - V^{\pi_n}$ le *gain en performance d'une itération à l'autre* et $l_n \stackrel{\text{def}}{=} V^* - V^{\pi_n}$ la *perte en performance* due à l'utilisation de la politique π_n au lieu de π^* (quantité que l'on désire majorer).

Si l'erreur d'approximation est faible, alors la performance de la politique suivante ne peut pas être bien pire que celle de la politique courante. En effet, en utilisant une notation vectorielle, on a les inégalités suivantes, composante par composante :

$$\begin{aligned} g_n &= V^{\pi_{n+1}} - V^{\pi_n} \\ &= L_{\pi_{n+1}} V^{\pi_{n+1}} - L_{\pi_{n+1}} V^{\pi_n} + L_{\pi_{n+1}} V^{\pi_n} - L_{\pi_{n+1}} V_n \\ &\quad + L_{\pi_{n+1}} V_n - L_{\pi_n} V_n + L_{\pi_n} V_n - L_{\pi_n} V^{\pi_n} \\ &\geq \gamma P_{\pi_{n+1}} g_n - \gamma (P_{\pi_{n+1}} - P_{\pi_n})\, e_n \end{aligned}$$

(où l'on a utilisé les définitions des opérateurs de Bellman et le fait que la politique π_{n+1} est gloutonne par rapport à V_n, donc que $L_{\pi_{n+1}} V_n = L V_n \geq L_{\pi_n} V_n$), donc $(I - \gamma P_{\pi_{n+1}}) g_n \geq -\gamma (P_{\pi_{n+1}} - P_{\pi_n}) e_n$. De plus, la matrice $P_{\pi_{n+1}}$ est une matrice stochastique donc n'a que des valeurs propres de module inférieur ou égale à un. Et puisque $\gamma < 1$, la matrice $I - \gamma P_{\pi_{n+1}}$ n'a pas de valeur propre nulle, donc elle est inversible. Et puisque son inverse $(I - \gamma P_{\pi_{n+1}})^{-1}$, qui peut se réécrire $\sum_{t\geq 0} (P_{\pi_{n+1}})^t$, ne contient que des éléments positifs, on en déduit :

$$g_n \geq -\gamma (I - \gamma P_{\pi_{n+1}})^{-1} (P_{\pi_{n+1}} - P_{\pi_n})\, e_n. \qquad (11.10)$$

Maintenant, nous pouvons majorer la perte l_{n+1} à l'itération suivante en fonction de la perte courante l_n : puisque $L_{\pi^*}V_n \leq L_{\pi_{n+1}}V_n$, on a :

$$\begin{aligned} l_{n+1} &= V^* - V^{\pi_{n+1}} \\ &= L_{\pi^*}V^* - L_{\pi^*}V^{\pi_n} + L_{\pi^*}V^{\pi_n} - L_{\pi^*}V_n + L_{\pi^*}V_n - L_{\pi_{n+1}}V_n \\ &\quad + L_{\pi_{n+1}}V_n - L_{\pi_{n+1}}V^{\pi_n} + L_{\pi_{n+1}}V^{\pi_n} - L_{\pi_{n+1}}V^{\pi_{n+1}} \\ &\leq \gamma[P_{\pi^*}l_n - P_{\pi_{n+1}}g_n + (P_{\pi_{n+1}} - P_{\pi^*})e_n]. \end{aligned}$$

D'où l'on déduit, en utilisant (11.10), que

$$\begin{aligned} l_{n+1} &\leq \gamma P_{\pi^*}l_n + \gamma[\gamma P_{\pi_{n+1}}(I - \gamma P_{\pi_{n+1}})^{-1}(P_{\pi_{n+1}} - P_{\pi_n}) + P_{\pi_{n+1}} - P_{\pi^*}]e_n \\ &\leq \gamma P_{\pi^*}l_n + \gamma[P_{\pi_{n+1}}(I - \gamma P_{\pi_{n+1}})^{-1}(I - \gamma P_{\pi_n}) - P_{\pi^*}]e_n. \end{aligned}$$

En notant $f_n \stackrel{\text{def}}{=} \gamma[P_{\pi_{n+1}}(I - \gamma P_{\pi_{n+1}})^{-1}(I - \gamma P_{\pi_n}) - P_{\pi^*}]e_n$, cette dernière inégalité peut se réécrire

$$l_{n+1} \leq \gamma P_{\pi^*}l_n + f_n.$$

En passant à la limite supérieure, il vient

$$(I - \gamma P_{\pi^*}) \limsup_{n\to\infty} l_n \leq \limsup_{n\to\infty} f_n,$$

et en utilisant le même argument que précédemment, on déduit

$$\limsup_{n\to\infty} l_n \leq (I - \gamma P_{\pi^*})^{-1} \limsup_{n\to\infty} f_n. \tag{11.11}$$

L'inégalité ne contenant que des termes positifs se préserve en norme :

$$\begin{aligned} \limsup_{n\to\infty} ||l_n||_\infty &\leq \frac{\gamma}{1-\gamma} \limsup_{n\to\infty} ||P_{\pi_{n+1}}(I - \gamma P_{\pi_{n+1}})^{-1}(I + \gamma P_{\pi_n}) + P_{\pi^*}||_\infty ||e_n||_\infty \\ &\leq \frac{\gamma}{1-\gamma}\left(\frac{1+\gamma}{1-\gamma} + 1\right) \limsup_{n\to\infty} ||e_n||_\infty = \frac{2\gamma}{(1-\gamma)^2} \limsup_{n\to\infty} ||e_n||_\infty \end{aligned}$$

(où l'on a utilisé le fait que toute matrice stochastique P a une norme $||P||_\infty = 1$).

□

11.3.2. *Évaluation approchée d'une politique*

Nous approfondissons maintenant la première étape de l'algorithme IPA, c'est-à-dire l'évaluation approchée de la politique. Ainsi, pour une politique π donnée, nous souhaitons déterminer une bonne approximation de la fonction de valeur V^π.

Rappelons que la fonction de valeur V^π est définie par l'espérance de la somme des récompenses actualisées lorsque la politique π est suivie (11.1) et que V^π satisfait l'équation de Bellman : $V^\pi = L_\pi V^\pi$, où L_π est l'opérateur de Bellman défini par $L_\pi W \stackrel{\text{def}}{=} r_\pi + \gamma P_\pi W$, pour tout vecteur W.

Plusieurs types de méthodes permettent de déterminer une approximation de V^π :

– **Méthodes itératives**, similaires à l'algorithme IVA, où l'opérateur L_π combiné à un opérateur d'approximation $\mathcal{A}$ est itéré selon : $V_{n+1} = \mathcal{A}L_\pi V_n$. Un résultat similaire à la proposition 11.2 se déduit facilement.

– **Méthodes de résolution de systèmes linéaires**, puisque l'opérateur de Bellman L_π est affine, V^π est solution du système linéaire : $(I - \gamma P_\pi)V^\pi = r_\pi$, et des méthodes usuelles pour la résolution (approchée) de système linéaire peuvent s'appliquer. L'inconvénient majeur de ces méthodes est leur lourdeur computationnelle lorsque le nombre d'états est grand.

– **Méthodes de type Monte-Carlo (MC)**. En reprenant la définition même de la fonction de valeur donnée par (11.1), un estimateur non-biaisé de $V^\pi(s)$ est obtenu en lançant $M \geq 1$ trajectoires partant de l'état initial s, générées en suivant la politique π, et en effectuant la moyenne sur ces M trajectoires de la somme des récompenses actualisées recueillies au cours de chaque trajectoire. La variance d'un tel estimateur est alors en $O(1/M)$. Si l'on répète ce procédé à partir d'états initiaux $\{s_k\}_{1 \leq k \leq K}$ répartis selon une distribution μ sur S, afin d'obtenir les estimations $\{v_k\}$ de $\{V^\pi(s_k)\}$, alors la meilleure fonction de $\mathcal{F}$ rendant compte de ces valeurs peut être obtenue par régression, par exemple en minimisant l'erreur empirique quadratique

$$\arg\min_{f \in \mathcal{F}} \frac{1}{K} \sum_{k=1}^{K} (f(s_k) - v_k)^2.$$

Il s'agit de la projection (au sens de la minimisation de la norme L^2 empirique basée sur les données) de V^π sur $\mathcal{F}$. Il existe plusieurs méthodes de réduction de variance permettant d'améliorer la précision des estimés. Par exemple, à partir d'une première approximation grossière de la fonction de valeur, on utilise une méthode de MC sur le résidu pour estimer une correction permettant d'affiner l'approximation courante, puis l'on réitère ce processus (méthodes de Monte-Carlo récursives [MUN 06]).

– **Algorithmes de type TD(λ)**. Il s'agit d'algorithmes d'approximation stochastiques [KUS 97] permettant de déterminer le point fixe d'un opérateur contractant. La généralisation de ces méthodes à des fonctions approchées n'est pas évidente dans le cas général (pas de preuve de convergence pour $\lambda < 1$) mais elles ont été appliquées avec grand succès, par exemple dans la programmation du jeu de backgammon

[TES 95]. Cependant, dans le cas de l'approximation linéaire (méthode détaillée au paragraphe suivant), des résultats de majoration de l'erreur d'approximation existent [TSI 96b].

– **Méthodes des moindres carrés**. Dans le cas de l'approximation linéaire, on peut aussi utiliser les méthodes très efficaces de type moindres carrés, étudiées au paragraphe suivant.

11.3.3. *Approximation linéaire et méthode des moindres carrés*

Dans cette section, nous considérons le cas particulier de l'approximation linéaire. Ce cas a souvent été utilisé en combinaison avec des algorithmes de type TD(λ) [TSI 96b] (voir section 2.4) ou des méthodes de type moindres carrés *Least Squares Temporal Differences* LSTD(0) [BRA 96], LSTD(λ) [BOY 99] et appliqué, avec succès, à des problèmes de contrôle en grande dimension [LAG 03].

La fonction de valeur est approchée par une combinaison linéaire de K fonctions de bases (appelés *features*) $(\phi_k)_{1\leq k\leq K}$, c'est-à-dire que l'espace d'approximation $\mathcal{F}$ est l'espace vectoriel engendré par ces *features* :

$$\mathcal{F} \stackrel{\text{def}}{=} \{V_\alpha(s) \stackrel{\text{def}}{=} \sum_{k=1}^{K} \alpha_k \phi_k(s), \alpha \in \mathbb{R}^K\}.$$

Notre objectif est ainsi de déterminer un paramètre $\alpha \in \mathbb{R}^K$ tel que V_α soit « proche » de V^π. Commençons par étudier l'extension directe de l'algorithme TD(λ) vu au chapitre 2 au cas d'une telle représentation de fonctions.

11.3.3.1. *TD(λ)*

Par soucis de simplicité de la présentation, nous supposons fini le nombre d'états de S. L'algorithme TD(λ) est défini de la même manière qu'au chapitre 2. On utilise un vecteur trace $z \in \mathbb{R}^K$ de même dimension (K) que le paramètre α, initialisé à zéro. A partir d'un état initial quelconque, on génère une trajectoire $(s_0, s_1, s_2, \dots)$ en suivant la politique π. A chaque instant t, on calcule la différence temporelle pour l'approximation V_α courante :

$$d_t \stackrel{\text{def}}{=} r(s_t, \pi(s_t)) + \gamma V_\alpha(s_{t+1}) - V_\alpha(s_t)$$

et l'on met à jour, à la fois le paramètre α et la trace z, selon

$$\begin{aligned} \alpha_{t+1} &= \alpha_t + \eta_t d_t z_t, \\ z_{t+1} &= \lambda\gamma z_t + \phi(s_{t+1}), \end{aligned}$$

où η_t est un pas d'apprentissage et $\phi : S \to \mathbb{R}^K$ la fonction ayant pour composantes les ϕ_k.

Cet algorithme fournit une séquence d'approximations V_{α_t} qui converge, sous une hypothèse d'ergodicité de la chaîne de Markov et une hypothèse sur la décroissance des pas d'apprentisage η_t, vers une fonction dont l'erreur d'approximation (par rapport à V^π) est majorée en fonction de la meilleure approximation possible dans $\mathcal{F}$.

PROPOSITION 11.4 [TSI 96B].– *Supposons que les pas η vérifient $\sum_{t\geq 0}\eta_t = \infty$ et $\sum_{t\geq 0}\eta_t^2 < \infty$, qu'il existe une distribution μ sur S telle que $\forall s, s' \in S$, $\lim_{t\to\infty} P(s_t = s'|s_0 = s) = \mu(s')$ et que les vecteurs $(\phi_k)_{1\leq k\leq K}$ soient linéairement indépendants. Alors α_t converge. Notons α^* sa limite. On a alors*

$$||V_{\alpha^*} - V^\pi||_\mu \leq \frac{1-\lambda\gamma}{1-\gamma}\inf_\alpha ||V_\alpha - V^\pi||_\mu, \tag{11.12}$$

où $||\cdot||_\mu$ désigne la norme L^2 pondérée par la distribution μ, c'est-à-dire $||f||_\mu \stackrel{\text{def}}{=} \left[\sum_{s\in S} f(s)^2\mu(s)\right]^{1/2}$.

Lorsque $\lambda = 1$, on retrouve le résultat que l'estimateur Monte-Carlo donne la meilleure approximation de V^π dans $\mathcal{F}$, c'est-à-dire la projection de V^π sur $\mathcal{F}$. Maintenant si $\lambda < 1$, la qualité d'approximation se détériore (introduction d'un biais), mais la variance de l'estimateur est plus faible, donc sa détermination à une précision donnée peut être plus rapide.

De part sa nature d'algorithme d'approximation stochastique, TD(λ) est très coûteux en quantité de données expérimentales, au sens où il nécessite l'observation d'un grand nombre de transitions $s_t, a_t \to s_{t+1}$ pour que le paramètre α converge. Il faut plusieurs observations des mêmes transitions pour que la valeur du paramètre se stabilise. Ce problème a motivé l'introduction de méthodes de type moindres carrés décrites maintenant, qui sont beaucoup plus économes en termes de données expérimentales.

11.3.3.2. *Méthodes des moindres carrés*

Les méthodes de type moindres carrés *Least Squares Temporal Differences* [BRA 96, BOY 99, LAG 03] partent du constat que, puisque la fonction de valeur approchée V_α est linéaire en α, et que l'opérateur L_π est affine, alors l'application du résidu de Bellman : $\alpha \to R_\alpha \stackrel{\text{def}}{=} L_\pi V_\alpha - V_\alpha$ est affine. Puisque la fonction de valeur recherchée V^π a un résidu de Bellman nul, i.e. $L_\pi V^\pi - V^\pi = 0$, il paraît naturel de chercher un paramètre α tel que le résidu de Bellman R_α soit le plus proche possible de 0. Deux approches sont généralement considérées (voir [SCH 02b, MUN 03]) :

– la **solution du résidu quadratique (RQ)** : le paramètre α_{RQ} est celui qui minimise la norme du résidu de Bellman R_α (voir illustration sur la figure 11.6) :

$$\alpha_{\text{RQ}} = \arg\min_{\alpha\in\mathbb{R}^K} ||R_\alpha||_\mu,$$

pour une certaine norme $||\cdot||_\mu$;

– la **solution des différences temporelles (TD)** : le paramètre α_{TD} est tel que le résidu $R_{\alpha_{\text{TD}}}$ est orthogonal à tous les *features* ϕ_k, donc à $\mathcal{F}$ (voir figure 11.6). Ainsi,

$V_{\alpha_{\mathrm{TD}}}$ est le point fixe de l'opérateur de Bellman L_π combiné à l'opérateur de projection orthogonale $\mathcal{A}$ sur $\mathcal{F}$ (selon la norme L^2 pondérée par une distribution μ). Il s'agit de la même solution que celle obtenue (à la limite) par l'algorithme TD(λ) pour $\lambda = 0$ (ce qui justifie son nom).

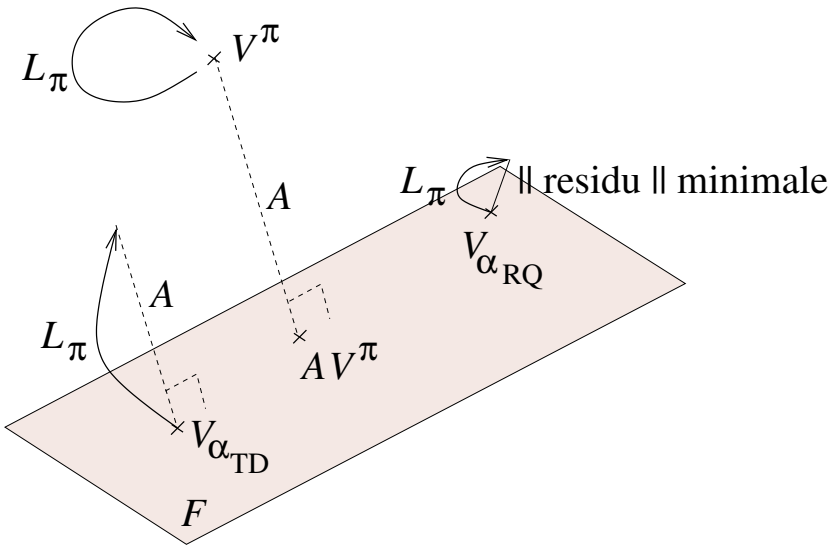


Figure 11.6. *Approximation de la fonction de valeur V^π dans $\mathcal{F}$. La meilleure approximation possible $\mathcal{A}V^\pi$ est la projection de V^π sur $\mathcal{F}$. La solution du résidu quadratique $V_{\alpha_{RQ}}$ est celle qui minimise (dans $\mathcal{F}$) la norme $||L_\pi V_\alpha - V_\alpha||$. La solution des différences temporelles $V_{\alpha_{TD}}$ est telle que $\mathcal{A}L_\pi V_{\alpha_{TD}} = V_{\alpha_{TD}}$*

Dans les deux cas, le paramètre α est obtenu en résolvant un système linéaire de taille K (le nombre de paramètres). Ces méthodes sont appelées *méthodes de projection* [JUD 98] car on cherche α tel que le résidu est orthogonal à un ensemble de fonctions test (les fonctions de base ϕ_k dans le cas TD, les dérivées $\partial_{\alpha_k} R_\alpha$ dans le cas RQ). Etudions les systèmes linéaires correspondants.

Solution du résidu quadratique : Puisque l'application $\alpha \rightarrow R_\alpha$ est affine, la fonction $\alpha \rightarrow ||R_\alpha||_\mu^2$ (pour une norme L^2 pondérée par μ) est quadratique. Son point de minimum (obtenu en écrivant que le gradient de cette fonction est nul) est donc la solution d'un système linéaire qui peut s'écrire $A\alpha = b$, avec la matrice carrée A et le vecteur b (de taille K) définis par :

$$\left\{ \begin{array}{rcll} A_{ij} & \stackrel{\mathrm{def}}{=} & \langle \phi_i - \gamma P_\pi \phi_i, \phi_j - \gamma P_\pi \phi_j \rangle_\mu, & \text{pour } 1 \leq i,j \leq K \\ b_i & \stackrel{\mathrm{def}}{=} & \langle \phi_i - \gamma P_\pi \phi_i, r_\pi \rangle_\mu, & \text{pour } 1 \leq i \leq K \end{array} \right. \tag{11.13}$$

où le produit scalaire $\langle u, v\rangle_\mu$ de deux fonctions u et v (définies sur S) est défini selon $\langle u, v\rangle_\mu \stackrel{\text{def}}{=} \sum_{s\in S} u(s)v(s)\mu(s)$.

Ce système possède toujours une solution lorsque $\mu > 0$ et la famille des ϕ_k est libre (puisqu'alors la matrice A est symétrique définie positive). Remarquons que ce problème peut être considéré comme un problème de régression linéaire avec un ensemble de fonctions de base $\{\psi_i \stackrel{\text{def}}{=} \phi_i - \gamma P_\pi \phi_i\}_{i=1..K}$ où il s'agit de déterminer α qui minimise $||\alpha\cdot\psi - r_\pi||_\mu$. Les méthodes usuelles en apprentissage supervisé peuvent alors être utilisées.

Lorsque μ est la distribution stationnaire associée à la politique π (i.e. μ satisfait la relation $\mu P_\pi = \mu$, c'est à dire $\mu(s) = \sum_{s'} p(s|s', \pi(s'))\mu(s')$ pour tout $s \in S$), on peut obtenir une borne sur l'erreur d'approximation $V^\pi - V_{\alpha_{\text{RQ}}}$ en fonction du résidu minimisé ou de la distance entre V^π et $\mathcal{F}$ (en norme L^2 avec poids μ) :

PROPOSITION 11.5.– *On a :*

$$\begin{aligned} ||V^\pi - V_{\alpha_{RQ}}||_\mu &\leq \frac{1}{1-\gamma}||R_{\alpha_{RQ}}||_\mu = \frac{1}{1-\gamma}\inf_{V_\alpha\in\mathcal{F}}||L_\pi V_\alpha - V_\alpha||_\mu \quad (11.14)\\ &\leq \frac{1+\gamma}{1-\gamma}\inf_{V_\alpha\in\mathcal{F}}||V^\pi - V_\alpha||_\mu. \quad (11.15)\end{aligned}$$

PREUVE.– Puisque μ est la distribution stationnaire associée à π, on a la propriété que $||P_\pi||_\mu = 1$ (voir par exemple [TSI 96b]). De plus, pour tout α, on a

$$R_\alpha = L_\pi V_\alpha - V_\alpha = (I - \gamma P_\pi)(V^\pi - V_\alpha), \quad (11.16)$$

donc en considérant α_{RQ}, nous déduisons que $V^\pi - V_{\alpha_{\text{RQ}}} = (I - \gamma P_\pi)^{-1} R_{\alpha_{\text{RQ}}}$. Donc en norme :

$$\begin{aligned} ||V^\pi - V_{\alpha_{\text{RQ}}}||_\mu &\leq ||(I - \gamma P_\pi)^{-1}||_\mu ||R_{\alpha_{\text{RQ}}}||_\mu\\ &\leq \sum_{t\geq 0}\gamma^t ||P_\pi||_\mu^t ||R_{\alpha_{\text{RQ}}}||_\mu \leq \frac{1}{1-\gamma}||R_{\alpha_{\text{RQ}}}||_\mu,\end{aligned}$$

ce qui prouve (11.14). De plus, en prenant la norme de (11.16), il vient $||R_\alpha||_\mu \leq (1+\gamma)||V^\pi - V_\alpha||_\mu$ et (11.15) s'en déduit. □

Solution des différences temporelles : La solution des différences temporelles, c'est-à-dire le point fixe de l'opérateur de Bellman L_π suivi de l'opérateur de projection $\mathcal{A}$ sur $\mathcal{F}$ (selon la norme $||\cdot||_\mu$), est obtenue en résolvant le système linéaire $A\alpha = b$ avec la matrice A et le vecteur b définis selon :

$$\left\{\begin{array}{rcll} A_{ij} & \stackrel{\text{def}}{=} & \langle \phi_i, \phi_j - \gamma P_\pi \phi_j\rangle_\mu, & \text{pour } 1 \leq i, j \leq K,\\ b_i & \stackrel{\text{def}}{=} & \langle \phi_i, r_\pi\rangle_\mu, & \text{pour } 1 \leq i \leq K.\end{array}\right. \quad (11.17)$$

Il faut faire attention, car selon la distribution μ considérée, la matrice A n'est pas toujours inversible. Elle l'est cependant lorsque les *features* (ϕ_i) forment une famille libre et que μ est la distribution stationnaire associée à la politique π [MUN 03]. Dans ce cas, la solution obtenue est celle vers laquelle converge l'algorithme TD(0) [SCH 02b]. Une conséquence de la proposition 11.4 est la borne sur l'erreur d'approximation en fonction de la distance entre V^π et $\mathcal{F}$ (en norme L^2 pondérée par μ) :

$$||V^\pi - V_{\alpha_{\text{TD}}}||_\mu \leq \frac{1}{1-\gamma} \inf_{V_\alpha \in \mathcal{F}} ||V^\pi - V_\alpha||_\mu.$$

La généralisation des méthodes de moindres carrés à des systèmes de type TD(λ) avec $\lambda > 0$ (i.e. fournissant la solution de TD(λ)) se trouve dans [BOY 99].

Remarquons que la taille du système linéaire à résoudre est K, le nombre de coefficients de l'approximation linéaire (nombre de features), qui en général est très inférieur au nombre d'états du système (ce dernier pouvant même être infini). Cependant, pour être applicable telle quelle, la méthode des moindres carrés nécessite de savoir calculer le résultat de l'opérateur P_π appliqué aux fonctions de base ϕ_k, ainsi que les produits scalaires (qui consistent en une somme pondérée sur tous les états pour lesquels $\mu > 0$). Ce problème est d'autant plus délicat lorsque l'on se situe dans un cadre apprentissage par renforcement où les probabilités de transition sont a priori inconnues de l'agent décisionnel. De plus, lorsque cette procédure d'évaluation de la politique courante est mise en œuvre dans un algorithme IPA, cette méconnaissance des probabilités rend problématique l'étape d'amélioration de la politique, qui nécessite le calcul de la politique gloutonne, selon (11.8). Ces problèmes sont résolus en introduisant, comme dans le chapitre 2, une approximation des fonctions de valeur d'action Q. Cette implémentation est explicitée maintenant.

11.3.3.3. *Approximation linéaire de la fonction de valeur d'action*

Ici nous ne faisons plus l'hypothèse que les probabilités de transition $p(s'|s,a)$ sont connues de l'agent décisionnel. Par contre, nous supposons que l'agent dispose d'un *modèle génératif* [KAK 03] qui permet de générer un état successeur $s' \sim p(\cdot|s,a)$ lorsque l'on se situe en un état s et choisit une action a, et ainsi de générer des trajectoires en suivant une certaine politique.

L'approche suivie ici consiste à approcher la fonction de valeur d'action Q^π [WAT 89, SUT 98] au lieu de la fonction de valeur V^π. Nous rappelons que la fonction Q^π est définie, pour tout couple état-action (s,a), par la récompense immédiate lorsque l'action a est choisie en s plus l'espérance de la somme des récompenses actualisées lorsqu'ensuite l'agent utilise la politique π, c'est-à-dire, en utilisant la définition de V^π,

$$Q^\pi(s,a) \stackrel{\text{def}}{=} r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^\pi(s').$$

Les représentations en termes de fonction de valeur et fonction de valeur d'action sont équivalentes : Q^π s'exprime en fonction de V^π comme le montre la relation précédente, et réciproquement, V^π se définit à partir de Q^π selon : $V^\pi(s) = Q^\pi(s, \pi(s))$. Cependant, un intérêt de la représentation en valeurs d'action est que la politique gloutonne se calcule très simplement : en tout $s \in S$, la politique gloutonne par rapport à V^π en s est $\arg\max_{a \in A} Q^\pi(s, a)$.

De manière analogue à l'approche suivie au paragraphe précédent, nous considérons un espace d'approximation linéaire pour la fonction Q^π de type :

$$\mathcal{F} \stackrel{\text{def}}{=} \left\{ Q_\alpha(s, a) \stackrel{\text{def}}{=} \sum_{k=1}^{K} \alpha_k \phi_k(x, a), \alpha \in \mathbb{R}^K \right\},$$

où les fonctions de base ϕ_k sont désormais définies sur l'espace produit $S \times A$.

L'algorithme IPA utilisant les représentations sous forme de valeurs d'action se définit ainsi : à l'itération n, l'étape d'évaluation approchée de la politique π_n détermine une approximation Q_n de la fonction Q^{π_n}. L'étape d'amélioration de la politique définit la politique suivante π_{n+1} selon :

$$\pi_{n+1}(s) \stackrel{\text{def}}{=} \arg\max_{a \in A} Q_n(s, a).$$

Les deux méthodes de type moindres carrés pour l'évaluation de la politique décrites précédemment s'appliquent immédiatement. De plus, la matrice A et le vecteur b des systèmes linéaires définis par (11.13) et (11.17) peuvent être estimés à partir de données observées, comme cela est expliqué dans [LAG 03] : une base de données D est construite à partir de séquences de trajectoires. A chaque transition (état s, action a) vers état $s' \sim p(\cdot|s, a)$ avec récompense correspondante r, on ajoute à la base D, la donnée (s, a, s', r). Les données expérimentales sur les transitions sont obtenues de manière incrémentale [BOY 99] ou à partir de l'observation de trajectoires induites par différentes politiques [LAG 03], ou encore à partir de données provenant de connaissances a priori sur les dynamiques d'état.

A partir de cette base d'expériences D, à chaque étape n de l'algorithme IPA, lorsque l'on désire évaluer une politique π_n, nous sélectionnons dans D les données $\{(s_m, a_m, s'_m, r_m)\}_{1 \leq m \leq M}$ telles que l'action choisie corresponde à la politique évaluée (i.e. $a_m = \pi_n(s_m)$). A partir de cette sélection de données, nous formulons un estimateur non-biaisé de A et b pour le système des différences temporelles (11.17) selon : pour $1 \leq i, j \leq K$,

$$\left\{ \begin{array}{rcl} \widehat{A}_{ij} & \stackrel{\text{def}}{=} & \frac{1}{M} \sum_{m=1}^{M} \phi_i(s_m, a_m)[\phi_j(s_m, a_m) - \gamma \phi_j(s'_m, \pi_n(s'_m))], \\ \widehat{b}_i & \stackrel{\text{def}}{=} & \frac{1}{M} \sum_{m=1}^{M} \phi_i(s_m, a_m) r_m. \end{array} \right.$$

En effet, puisque les états suivants s' sont générés selon $p(\cdot|s, a)$, on a la propriété que $\phi_j(s', \pi_n(s'))$ est un estimateur non biaisé de l'opérateur P_{π_n} appliqué à ϕ_j en

s, c'est-à-dire $\sum_{s' \in S} p(s'|s,a)\phi_j(s', \pi_n(s'))$. De plus, par la loi des grands nombres, lorsque le nombre de données M est grand, les moyennes réalisées sur les M échantillons approchent les produits scalaires dans (11.17). Ainsi $\widehat{A}$ et $\widehat{b}$ sont des estimateurs non-biaisés et consistants de A et b ayant une variance en $O(1/M)$. Lorsque le système (11.17) admet une solution, la solution $\widehat{\alpha}$ du système approché $\widehat{A}\widehat{\alpha} = \widehat{b}$ tend vers la solution α_{TD} du système des différences temporelles (11.17) lorsque $M \to \infty$.

De manière analogue, nous pourrions penser que

$$\frac{1}{M}\sum_{m=1}^{M}\left[\phi_i(s_m,a_m) - \gamma\phi_i(s'_m, \pi_n(s'_m))\right]\left[\phi_j(s_m,a_m) - \gamma\phi_j(s'_m, \pi_n(s'_m))\right] \tag{11.18}$$

fournisse un estimateur non-biaisé de l'élément A_{ij} pour le système du résidu quadratique (11.13). Or, cela est faux (voir [SUT 98], p. 220 ou [MUN 03, LAG 03]). Cet estimateur mènerait en effet à une paramétrisation qui chercherait à réduire la variance de la fonction de valeur d'action aux états successeurs. Le problème vient du fait que les variables aléatoires $\phi_i(s', \pi_n(s'))$ et $\phi_j(s', \pi_n(s'))$ sont corrélées. Plusieurs possibilités pour retrouver un estimateur non biaisé de A et b sont :

– Pour chaque couple état-action (s_m, a_m), utiliser deux échantillons indépendants s'_m et s''_m tirés selon $p(\cdot|s,a)$ en utilisant le modèle génératif, afin de décorréler $\phi_i(s', \pi_n(s'))$ et $\phi_j(s'', \pi_n(s''))$. Alors, un estimateur non-biaisé de A et b pour le système du résidu quadratique (11.13) est

$$\left\{\begin{array}{rcl} \widehat{A}_{ij} & \stackrel{\text{def}}{=} & \frac{1}{M}\sum_{m=1}^{M}\begin{array}{l}\left[\phi_i(s_m,a_m) - \gamma\phi_i(s'_m, \pi_n(s'_m))\right] \\ \left[\phi_j(s_m,a_m) - \gamma\phi_j(s''_m, \pi_n(s''_m))\right],\end{array} \\ \widehat{b}_i & \stackrel{\text{def}}{=} & \frac{1}{M}\sum_{m=1}^{M}\left[\phi_i(s_m,a_m) - \gamma\phi_i(s'_m, \pi_n(s'_m))\right] r_m. \end{array}\right.$$

– Si nous ne disposons que d'un seul échantillon s'_m par couple état-action (s_m, a_m) (par exemple parce que les données ont été générées en suivant des trajectoires) nous pouvons considérer le plus proche voisin $s_{m'}$ de s_m tel que $(s_{m'}, a_m, s'_{m'}, r_{m'})$ soit dans la base D. A l'introduction d'un biais près (qui provient de la distance entre s_m et $s_{m'}$) qui dépend de la régularité des probabilités de transition et de la densité de données, ceci fournit à nouveau deux échantillons décorrélés et un estimateur de A et b pour le système (11.13) se déduit de la même façon.

– Une troisième approche, étudiée dans [**?**], consiste à modifier la fonction à minimiser en soustrayant du résidu ($||L_\pi Q_n - Q_n||^2_\mu$) un terme ($||\mathcal{A}L_\pi Q_n - L_\pi Q_n||^2_\mu$ où $\mathcal{A}L_\pi Q_n$ est la projection sur $\mathcal{F}$ de $L_\pi Q_n$) dont l'estimation (à partir des données) possède le même biais que celui du résidu (11.18), annulant ainsi le biais de l'estimateur résultant. Nous ne détaillons pas ici cette approche mais mentionnons le fait que cette approche se généralise à des approximations non-linéaires, et que dans le cas particulier de l'approximation linéaire, la solution du système correspondant se ramène à celle des différences temporelles.

Les algorithmes présentés dans cette section sont efficaces en termes d'utilisation des données expérimentales, puisque les données d'observation des transitions $s, a \rightarrow s', r$ sont mémorisées et permettent de déterminer directement le paramètre α par la résolution d'un système linéaire. Des versions incrémentales pour la résolution de ces systèmes sont bien entendu possibles.

11.4. Minimisation directe du résidu de Bellman

En plus des méthodes usuelles d'itérations sur les valeurs et d'itérations sur les politiques étudiées aux sections précédentes, nous présentons ici une méthode qui consiste à chercher directement à minimiser la norme du résidu de Bellman (pour l'opérateur L). L'idée est très simple : puisque la fonction de valeur optimale V^* est le point fixe de l'opérateur L, c'est-à-dire que la norme de son résidu $||LV^* - V^*||$ est nulle, nous pouvons chercher à déterminer le minimum dans $\mathcal{F}$ de la norme du résidu :

$$\inf_{V_\alpha \in \mathcal{F}} ||LV_\alpha - V_\alpha||, \tag{11.19}$$

où $||\cdot||$ est une certaine norme.

Nous avons le résultat suivant, en norme L^∞, qui nous indique que si le résidu est bien minimisé, alors la performance de la politique résultante est proche de l'optimum.

PROPOSITION 11.6 [WIL 93].– *Soit V une fonction définie sur S et π une politique gloutonne par rapport à V. La perte en performance résultant de l'utilisation de la politique π au lieu de la politique optimale est majorée en fonction du résidu de Bellman de V selon :*

$$||V^* - V^\pi||_\infty \leq \frac{2}{1-\gamma} ||LV - V||_\infty. \tag{11.20}$$

PREUVE.– Puisque $L_\pi V = LV \geq L_{\pi^*} V$, on a

$$\begin{aligned} V^* - V^\pi &= L_{\pi^*} V^* - L_{\pi^*} V + L_{\pi^*} V - LV + L_\pi V - L_\pi V^\pi \\ &\leq \gamma P_{\pi^*}(V^* - V^\pi + V^\pi - V) + \gamma P_\pi (V - V^\pi). \end{aligned}$$

Donc

$$(I - \gamma P_{\pi^*})(V^* - V^\pi) \leq \gamma (P_{\pi^*} - P_\pi)(V^\pi - V),$$

et en utilisant la relation $V^\pi - V = (I - \gamma P_\pi)^{-1}(LV - V)$, il vient

$$\begin{aligned} V^* - V^\pi &\leq \gamma (I - \gamma P_{\pi^*})^{-1} (P_{\pi^*} - P_\pi)(I - \gamma P_\pi)^{-1}(LV - V) \\ &= \left[(I - \gamma P_{\pi^*})^{-1} - (I - \gamma P_\pi)^{-1}\right](LV - V), \end{aligned} \tag{11.21}$$

donc en norme L^∞,

$$\begin{aligned} ||V^* - V^\pi||_\infty &\leq \left[||(I-\gamma P_{\pi^*})^{-1}||_\infty + ||(I-\gamma P_\pi)^{-1}||_\infty\right] ||LV - V||_\infty \\ &\leq \frac{2}{1-\gamma} ||LV - V||_\infty. \end{aligned}$$

Ainsi nous avons une bonne garantie de performance des politiques gloutonnes par rapport à des fonctions V_α minimisant bien la norme du résidu. Cependant, le problème de minimisation (11.19) peut être difficile à résoudre, même dans le cas d'une paramétrisation linéaire, car l'opérateur L n'est pas affine (contrairement à l'opérateur L_π) à cause de l'évaluation du maximum sur toutes les actions. Il n'existe pas de méthode simple pour déterminer le minimum global (contrairement au cas vu précédemment utilisant l'opérateur L_π et une approximation linéaire où le résidu à minimiser $\alpha \rightarrow ||L_\pi V_\alpha - V_\alpha||_\mu$ était une fonction quadratique en α) mais des méthodes locales de type gradient (où la direction opposée au gradient $\nabla_\alpha ||LV_\alpha - V_\alpha||_\mu^2$ est suivie), par exemple dans des réseaux de neurones, sont largement employées dans la pratique pour minimiser la norme L^2 du résidu, bien qu'il n'y ait pas de garantie de convergence vers un minimum global.

11.5. Vers une analyse de la programmation dynamique en norme L^p

Nous avons donné plusieurs résultats de majoration ((11.6) et (11.9) respectivement pour les algorithmes IVA et IPA) de la perte en performance en fonction des erreurs d'approximation, *en norme* L^∞, commises à chaque étape. Cependant, comme illustré à la section 11.2.1, les algorithmes usuels en apprentissage supervisé réalisent un problème de minimisation en norme L^p (avec $p = 1$ ou 2). Ainsi, les bornes sur l'erreur d'approximation (par exemple (11.12) ou (11.15)) sont de nature L^p alors que celles sur la propagation d'erreur en programmation dynamique sont de nature L^∞.

Le problème fondamental de l'analyse de la programmation dynamique (PD) avec approximation de fonctions réside dans l'utilisation d'outils différents en programmation dynamique et en théorie de l'approximation :

– L'analyse usuelle en PD utilise la norme L^∞, qui est très naturelle car les opérateurs de Bellman L et L_π sont des contractions en norme L^∞. Les algorithmes d'itération sur les valeurs, itération sur les politiques et leurs variantes en A/R sont basés sur cette propriété.

– L'approximation de fonction utilise très majoritairement les normes L^p : par exemple, les méthodes de type moindres carrés, les réseaux de neurones, les méthodes à noyaux, etc.

Cette différence de norme complique l'analyse des méthodes de PD combinées à des fonctions approchées. Par exemple si l'on considère l'algorithme IVA, défini par (11.3), l'opérateur de Bellman L est une contraction en norme L^∞, l'opérateur

d'approximation $\mathcal{A}$ est une non-expansion en norme L^2 (dans le cas d'une projection orthogonale sur $\mathcal{F}$), mais on ne peut rien dire de l'opérateur composé $\mathcal{A}L$. L'analyse L^∞ de cet algorithme, illustrée par le résultat (11.6), établit une majoration sur la performance en fonction de l'erreur d'approximation uniforme $||\varepsilon_n||_\infty$, quantité qu'il est très difficile d'estimer, surtout pour des problèmes de grande taille. Ce résultat est difficilement exploitable d'un point de vue pratique. De plus, la plupart des opérateurs d'approximation et algorithmes d'apprentissage supervisé résolvent un problème de minimisation empirique en norme L^1 ou L^2, par exemple (11.4). L'erreur uniforme $||\varepsilon_n||_\infty$ se contrôle difficilement par l'erreur réellement minimisée par le problème L^p empirique (11.4).

Une analyse L^p de l'algorithme IVA qui prendrait en compte les erreurs d'approximation en norme L^p permettrait d'évaluer la performance des algorithmes en fonction des erreurs empiriques réellement commises ainsi que d'un terme de capacité (de type dimension de Vapnik-Chervonenkis ou nombre de couverture [POL 84, VAP 98]) de l'espace fonctionnel $\mathcal{F}$ considéré. En effet, en utilisant les résultats récents en apprentissage statistique, l'erreur d'approximation en norme L^p (appelée « erreur en généralisation » selon la terminologie apprentissage supervisé) peut être majorée par l'erreur empirique (ou « erreur en apprentissage ») réellement minimisée plus un terme de capacité de $\mathcal{F}$. De plus, puisque la plupart des opérateurs d'approximation et algorithmes d'apprentissage supervisé fournissent de bonnes régressions en minimisant une norme L^p, il apparaît donc essentiel de pouvoir analyser la performance des algorithmes de PD en utilisant cette même norme.

Des premiers travaux dans cette direction [MUN 07b, MUN 07a] sont brièvement décrits dans les deux sections suivantes.

11.5.1. *Intuition d'une analyse L^p en programmation dynamique*

Rappelons tout d'abord la définition de la norme L^p pondérée par une distribution μ sur S : $||f||_{p,\mu} \stackrel{\text{def}}{=} \left[\sum_{s\in S} \mu(s)|f(s)|^p\right]^{1/p}$. Lorsque $p = 2$, nous utilisons la notation allégée $||f||_\mu$.

L'intuition sous-jacente à une analyse en norme L^p de la PD est simple et passe par la déduction de bornes composante par composante. En effet, soient f et g définies sur S, à valeurs positives, telles que $f \leq Pg$, avec P une matrice stochastique. Bien sûr, ceci implique que $||f||_\infty \leq ||g||_\infty$ (puisque $||P||_\infty = 1$), mais de plus, si ν et μ sont des distributions sur S telles que $\nu P \leq C\mu$, (il faut comprendre la notation νP au sens matriciel comme le produit du vecteur ligne ν par la matrice P) avec $C \geq 1$ une constante, alors nous déduisons aussi que $||f||_{p,\nu} \leq C^{1/p}||g||_{p,\mu}$.

En effet, nous avons

$$\begin{aligned} ||f||_{p,\nu}^p = \sum_{s\in S} \nu(s)|f(s)|^p &\leq \sum_{s\in S} \nu(s)\big|\sum_{s'\in S} P(s'|s)g(s')\big|^p \\ &\leq \sum_{s\in S} \nu(s) \sum_{s'\in S} P(s'|s)|g(s')|^p \\ &\leq C\sum_{s'\in S} \mu(s')|g(s')|^p = C||g||_{p,\mu}^p, \end{aligned}$$

où l'on a utilisé l'inégalité de Jensen (convexité de $x \to |x|^p$) à la seconde ligne.

Par exemple, la borne composante par composante (11.21) permet de déduire la majoration (11.20) en fonction de la norme L^∞ du résidu de Bellman. De cette même borne (11.21), nous pouvons aussi déduire une majoration en fonction de la norme L^p du résidu de Bellman :

$$||V^* - V^\pi||_{p,\nu} \leq \frac{2}{1-\gamma} C(\nu,\mu)^{1/p} ||LV - V||_{p,\mu}, \tag{11.22}$$

où ν et μ sont deux distributions sur S et $C(\nu,\mu)$ une constante qui mesure la concentration (relativement à μ) de la répartition d'états futurs (sachant que l'état initial est tiré selon ν) du MDP (voir [MUN 07b, MUN 07a] pour une définition précise et le lien avec les exposants de Lyapunov dans les systèmes dynamiques). Cette majoration est plus fine que la borne L^∞ puisque, lorsque $p \to \infty$, on retrouve la borne (11.20).

Il en va de même pour les majorations des algorithmes IVA et IPA. A titre d'illustration, pour l'algorithme IVA, on peut montrer la borne composante par composante suivante :

$$\begin{aligned} \limsup_{n\to\infty} V^* - V^{\pi_n} \leq \limsup_{n\to\infty} (I - \gamma P_{\pi_n})^{-1} \\ \Big(\sum_{k=0}^{n-1} \gamma^{n-k}\big[(P_{\pi^*})^{n-k} + P_{\pi_n} P_{\pi_{n-1}} \dots P_{\pi_{k+2}} P_{\pi_{k+1}}\big]|\varepsilon_k|\Big), \end{aligned}$$

avec $\varepsilon_k = LV_k - V_{k+1}$ (l'erreur d'approximation à l'étape k). Prise en norme L^∞, cette borne mène à (11.6). Mais on en déduit aussi la borne L^p :

$$\limsup_{n\to\infty} ||V^* - V^{\pi_n}||_{p,\nu} \leq \frac{2\gamma}{(1-\gamma)^2} C(\nu,\mu)^{1/p} \limsup_{n\to\infty} ||\varepsilon_n||_{p,\mu}. \tag{11.23}$$

De même pour l'algorithme IPA, la borne composante par composante (11.11) permet de montrer le résultat en norme L^∞ (11.9) comme cela a été démontré précédemment, mais aussi de déduire la borne L^p suivante :

$$\limsup_{n\to\infty} ||V^* - V^{\pi_n}||_{p,\nu} \leq \frac{2\gamma}{(1-\gamma)^2} C(\nu,\mu)^{1/p} \limsup_{n\to\infty} ||V_n - V^{\pi_n}||_{p,\mu}. \tag{11.24}$$

Cette analyse L^p en PD permet d'établir un lien avec l'apprentissage statistique et de déduire des bornes PAC (*Probablement Approximativement Correctes*) [VAL 84, BOU 92] pour des algorithmes d'A/R.

11.5.2. *Bornes PAC pour des algorithmes d'A/R*

Nous donnons ici un résultat (explicité dans [MUN 07a]) de borne PAC pour un algorithme d'A/R basé sur la méthode d'IVA. On se place dans le cas d'un espace d'état S très grand, par exemple continu. A chaque itération, l'opérateur d'approximation consiste en une régression empirique réalisée à partir d'un nombre fini N d'états où, en chacun de ces états, l'opérateur de Bellman est estimé à l'aide d'un nombre M d'états successeurs obtenus en faisant appel au modèle génératif.

Plus précisément, on répète K étapes d'itération sur les valeurs avec approximation (11.3). A l'étape $1 \leq k < K$, on dispose d'une représentation courante $V_k \in \mathcal{F}$ de la fonction de valeur et on calcule une nouvelle approximation V_{k+1} de la manière suivante. On tire N états $\{s_n\}_{1\leq n\leq N} \in S$ selon une distribution μ sur S. Pour chaque état s_n et chaque action $a \in A$, on génère M successeurs $\{s'_{n,a,m} \sim p(\cdot|s_n, a)\}_{1\leq m\leq M}$ et on calcule une estimation empirique de l'opérateur de Bellman appliqué à V_k en s_n :

$$v_n \stackrel{\text{def}}{=} \max_{a\in A} \big[r(s_n, a) + \gamma \frac{1}{M} \sum_{m=1}^{M} V_k(s'_{n,a,m})\big],$$

et l'on déduit V_{n+1} en résolvant le problème de régression en norme L^p :

$$V_{n+1} \stackrel{\text{def}}{=} \arg\min_{f\in\mathcal{F}} \frac{1}{N} \sum_{n=1}^{N} \big|f(s_n) - v_n\big|^p.$$

A l'étape K, on note π_K une politique gloutonne par rapport à V_K et on souhaite évaluer sa performance (par rapport à la performance optimale) en fonction du nombre d'itérations K, du nombre d'états N, du nombre de successeurs M, de la capacité de l'espace fonctionnel $\mathcal{F}$, de la régularité du MDP (en termes de la constante $C(\nu, \mu)$ apparaissant dans (11.23)), et du résidu de Bellman $d(L\mathcal{F}, \mathcal{F})$ inhérent à $\mathcal{F}$. On a le résultat suivant.

PROPOSITION 11.7 [MUN 07A].– *Pour tout $\delta > 0$, avec probabilité au moins $1 - \delta$, on a :*

$$\begin{aligned} ||V^* - V^{\pi_K}||_{p,\nu} \quad \leq \quad & \frac{2\gamma}{(1-\gamma)^2} C(\nu,\mu)^{1/p}\, d(L\mathcal{F}, \mathcal{F}) + O(\gamma^K) \qquad (11.25) \\ & + O\left\{ \left(\frac{V_{\mathcal{F}^+} \log(1/\delta)}{N} \right)^{1/2p} + \left(\frac{\log(1/\delta)}{M} \right)^{1/2} \right\}, \end{aligned}$$

où $d(L\mathcal{F}, \mathcal{F}) \stackrel{\text{def}}{=} \sup_{g\in\mathcal{F}} \inf_{f\in\mathcal{F}} ||Lg - f||_{p,\mu}$ *est le résidu de Bellman inhérent à l'espace* $\mathcal{F}$*, et* $V_{\mathcal{F}^+}$ *une mesure de capacité (la pseudo-dimension) de* $\mathcal{F}$ *[HAU 95].*

Les quatre termes de cette borne expriment respectivement :

– le résidu de Bellman $d(L\mathcal{F}, \mathcal{F})$, qui généralise la notion de résidu de Bellman à un espace fonctionnel. Il mesure de combien on peut approcher les fonctions Lg, images par l'opérateur de Bellman de fonctions $g \in \mathcal{F}$, par des fonctions de $\mathcal{F}$. Ce terme est analogue à la notion de distance à $\mathcal{F}$ de la fonction à apprendre dans le cas de la régression, à part qu'ici il ne s'agit pas d'approcher (à l'aide de fonctions de $\mathcal{F}$) une fonction cible donnée, mais de déterminer le point fixe d'un opérateur de Bellman à l'aide de fonctions de $\mathcal{F}$. Lorsque le MDP est régulier (par exemple si les probabilités de transition $p(s'|\cdot, a)$ et la fonction récompense $r(\cdot, a)$ sont lipschitziennes), on peut montrer (résultat non évident !) que ce terme décroît lorsque l'espace d'approximation $\mathcal{F}$ grossit (car alors l'opérateur de Bellman a un effet régularisant, et l'espace $L\mathcal{F}$ reste inclus dans un espace de fonctions lipschitziennes dont le coefficient est indépendant de $\mathcal{F}$, voir [MUN 07a]) ;

– le terme dû au nombre fini K d'itérations réalisées, qui tend vers 0 rapidement ;

– deux termes en $O((V_{\mathcal{F}^+}/N)^{1/2p}) + O(1/\sqrt{M})$ qui bornent l'erreur d'estimation en fonction du nombre d'échantillons N et M utilisés.

Le résultat obtenu exprime que, si l'on utilise suffisamment de ressources computationnelles (N, M, K), la performance de cet algorithme peut être rendue arbitrairement proche (à une constante près) du résidu de Bellman inhérent à $\mathcal{F}$, qui lui-même peut être rendu petit en utilisant une architecture d'approximation suffisamment riche. Ce type de majoration est analogue aux bornes obtenues en apprentissage supervisé [GYÖ 02] et permet d'analyser la stabilité et la vitesse de convergence de IVA à partir d'échantillons, en particulier,

– de comprendre le *compromis biais-variance* en programmation dynamique avec approximation. En effet, la borne (11.25) sur la perte en performance contient un terme de biais, le résidu de Bellman $d(L\mathcal{F}, \mathcal{F})$, qui décroît lorsque $\mathcal{F}$ grossit, et un terme de variance, dû à la capacité $V_{\mathcal{F}^+}$ de $\mathcal{F}$, qui croît avec la richesse de $\mathcal{F}$, mais qui peut être atténué en utilisant un plus grand nombre d'échantillons N (pour éviter le sur-apprentissage) ;

– de rendre compte des contre-exemples (de divergence de l'algorithme IVA) mentionnés dans la littérature (en particulier, le résidu de Bellman $d(L\mathcal{F}, \mathcal{F})$ inhérent aux espaces utilisés dans [BAI 95, TSI 96a] est infini) et de pouvoir prédire le comportement de ce type d'algorithmes en fonction des caractéristiques du MDP, de la capacité et richesse de l'espace $\mathcal{F}$ et de la quantité de ressources computationnelles utilisées.

11.6. Conclusion et perspectives

Les résultats décrits au paragraphe précédent découlent directement de la combinaison d'outils d'apprentissage statistique à l'analyse en norme L^p en programmation

dynamique qui a été brièvement présentée. De nombreuses extensions sont bien entendu possibles, par exemple pour des méthodes d'A/R basées sur l'algorithme IPA (tel que LSPI de [LAG 03]) et même lorsque les échantillons sont obtenus par l'observation d'une unique trajectoire [**?**]. En parallèle de ces travaux théoriques, mentionnons la diversité et la quantité des travaux portant sur l'utilisation de représentations approchées de la fonction valeur pour la prise de décisions. Les méthodes à noyaux [SCH 01] ont été appliquées à la PD et l'A/R [ORM 02, RAS 04], ainsi que les arbres de décision [WAN 99, ERN 05], les réseaux de neurones [BER 96, COU 02], les approches bayésiennes [DEA 98], les représentations factorisées [GUE 01a, KOL 00, DEG 06] (voir chapitre 9) pour ne citer que quelques travaux.

Ce chapitre s'est consacré à la représentation approchée de *la fonction valeur*. Cependant, une tout autre approche, quelque peut orthogonale à celle-ci, consiste à représenter de manière approchée *la politique*, et à résoudre un problème d'optimisation paramétrique (maximisation de la performance de la politique paramétrée). Alors que l'approche d'approximation de la fonction valeur est basée sur la méthode de la programmation dynamique, initiée par Bellman [BEL 57], l'approche de recherche d'une politique paramétrée localement optimale s'inspire des travaux de Pontryagin [PON 62] sur la détermination de conditions nécessaires d'optimalité et l'analyse de sensibilité (estimation du gradient de la performance par rapport au paramètre de la politique) en contrôle et dans les MDP. Il s'agit de l'objet du chapitre suivant.

Chapitre 12

Méthodes de gradient pour la recherche de politiques paramétrées

La plupart des approches de résolution de MDP passe par l'évaluation d'une fonction de valeur, laquelle permet de déterminer les actions optimales dans chaque état. Une première difficulté pour ces approches est de traiter des problèmes de grande taille. Comme l'a montré le chapitre 11, une solution est de passer par des méthodes d'approximation de la fonction de valeur, mais l'amélioration monotone de la politique n'est alors plus garantie. Une deuxième difficulté est la gestion d'observations partielles, cadre dans lequel la propriété de Markov n'est plus vérifiée.

Une approche tout à fait différente est d'effectuer une recherche directe de politique, celle-ci prenant la forme d'un contrôleur paramétré. Le choix de ce contrôleur permet de s'adapter au type de problème considéré et de faire un compromis entre la possibilité d'exprimer une grande variété de politiques d'une part et une taille mémoire réduite d'autre part. On se retrouve avec un problème d'optimisation de paramètres pour lequel des algorithmes existent qui garantissent une amélioration monotone de la politique vers un optimum local, y compris, dans certains cas, si l'observabilité est partielle.

EXEMPLE.– Et pour les voitures[1] qu'il faut entretenir alors ? Un parallèle possible est de considérer un garagiste qui, bien que cherchant à minimiser le coût d'entretien, décide de ne réparer une voiture que si le temps de la réparation est inférieur à un certain seuil, disons 5 heures. Avec le temps et l'expérience, en fonction du coût à long terme de cette stratégie, le garagiste sera sans doute amené à modifier son seuil de travail de

Chapitre rédigé par Olivier BUFFET.

1. voir détails tome 1, section 1.1

manière à diminuer le coût. Les approches par montée de gradient fonctionnent selon ce principe. Plutôt que de chercher à estimer la valeur à long terme de chaque action dans chaque état, on cherche plutôt à modifier les paramètres d'une politique (ici le temps de travail maximum) pour diminuer le coût à long terme.

On pourrait envisager d'adapter divers types d'algorithmes d'optimisation. Ce chapitre se concentre sur les méthodes de gradient, lesquelles ont été particulièrement étudiées. Après de brefs rappels sur la notion de gradient (section 12.1), nous présentons ici deux types d'approches : l'utilisation directe de méthodes de gradient en section 12.2 et les méthodes de gradient acteur-critique en section 12.3.

12.1. Rappels sur la notion de gradient

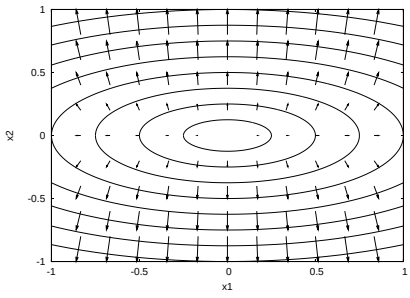
Nous revenons ici sur la notion générale de gradient et d'optimisation par méthode de gradient avant d'introduire le cas des MDP.

12.1.1. *Gradient d'une fonction*

Le gradient est un opérateur vectoriel qui, à une fonction différentiable $f : \mathbb{R}^n \rightarrow \mathbb{R}$, associe $\vec{\nabla} f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ définie par :

$$\vec{\nabla} f(x_1, \ldots, x_n) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x_1, \ldots, x_n) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x_1, \ldots, x_n) \end{bmatrix} \tag{12.1}$$

Le gradient d'une fonction f en un point $\vec{x}$ est un vecteur perpendiculaire à la surface d'égale valeur $f(\vec{x})$, vecteur orienté dans la direction selon laquelle la fonction croît et dont la norme indique la « vitesse » de variation de f dans cette direction.



$$\vec{\nabla} f(x_1, x_2) = \begin{bmatrix} 2x_1 \\ 8x_2 \end{bmatrix}$$

Figure 12.1. *Lignes équipotentielles de la fonction $f(x_1, x_2) = x_1^2 + 4x_2^2$ et le champ de vecteurs gradient associé*

12.1.2. *Descente de gradient*

La descente de gradient est un algorithme itératif permettant de trouver un minimum local d'une fonction f dont le gradient existe et est connu. Le principe est de construire une suite de points $(\vec{x}_n)_{n\in\mathbb{N}}$ en suivant en chaque point la direction opposée au gradient, de manière à descendre au point suivant. L'algorithme 12.1 donne le détail de l'algorithme, lequel requiert :

– une fonction différentiable f,

– un point de départ $\vec{x_0}$,

– un pas $\alpha > 0$, coefficient multiplicateur permettant d'ajuster la taille des pas de la descente de gradient ; et

– souvent un seuil $\epsilon > 0$ utilisé par un critère d'arrêt de l'algorithme (ici fonction des deux derniers points visités).

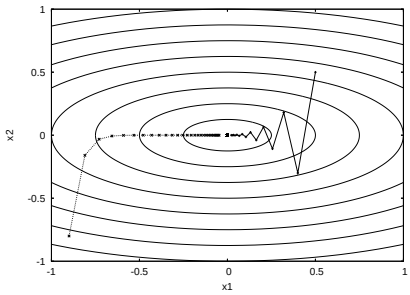
Le critère d'arrêt peut par exemple être ($\|\vec{x}_n - \vec{x}_{n-1}\| \leq \epsilon$) ou ($\|f(\vec{x}_n) - f(\vec{x}_{n-1})\| \leq \epsilon$). Par la suite on éludera ce problème en considérant une boucle infinie.

Algorithme 12.1 : Descente de gradient$(f, \vec{x_0}, \alpha, \epsilon)$

Initialisation :
$n \leftarrow 0$
répéter
  $n \leftarrow n + 1$
  $\vec{x}_n \leftarrow \vec{x}_{n-1} - \alpha\vec{\nabla} f(\vec{x}_{n-1})$
jusqu'à *critère_d'arrêt*$(\vec{x}_n, \vec{x}_{n-1}, \epsilon)$
retourner x_n

La figure 12.2 montre deux séquences de points obtenues par ce même algorithme mais avec des points de départ différents et, surtout, des pas α différents. Si ce pas est trop grand, il provoque des oscillations ; s'il est trop petit, l'algorithme se rapproche trop lentement du minimum à chaque itération. Dans les deux cas, la convergence est ralentie. Trouver un bon pas est un problème en soi, en particulier parce que le pas peut évoluer. Des recherches spécifiques se concentrent sur ce problème, mais nous le laisserons de côté.

NOTE.– Pour un problème de maximisation, comme ce sera le cas par la suite, on parlera de *montée de gradient*, l'algorithme employé n'étant modifié qu'au niveau du signe précédant le pas α.



	gauche	droite
$\vec{x}_0$	$[-.9, -.8]$	$[+.5, +.5]$
α	.1	.2
ϵ	.005	

Les deux séquences de point convergent vers le minimum $(0, 0)$ mais avec des comportements très différents liés à leurs pas d'apprentissage.

Figure 12.2. *Lignes équipotentielles de la fonction $f(x_1, x_2) = x_1^2 + 4x_2^2$ et deux séquences de points obtenues par descente de gradient depuis des origines différentes et avec des pas différents*

12.2. Optimisation d'une politique paramétrée par méthode de gradient

Historiquement, l'algorithme de référence dans les recherches directes de politique par méthode de gradient est REINFORCE de Williams [WIL 87, WIL 92]. Ces travaux ont ensuite été généralisés et améliorés par Baird et Moore [BAI 99a, BAI 99b], comme par Baxter et Bartlett [BAX 01a, BAX 01b].

12.2.1. *Application aux* MDP *: aperçu*

Dans le cas d'un MDP, on cherche à maximiser une mesure de performance dépendant de la politique (stochastique) π appliquée. Notons par exemple $g(\pi)$ un critère tel que ceux introduits en section 1.2.3. Pour pouvoir employer une montée de gradient, on écrit la politique comme une fonction d'un vecteur de paramètres $\vec{\theta}$: $\pi = h(\vec{\theta})$, de manière à ce que la mesure de performance soit une fonction $f(\vec{\theta}) = g \circ h(\vec{\theta})$. Si cette fonction est différentiable, les conditions sont réunies pour appliquer une méthode de gradient.

Exemple de définition d'une politique paramétrée

Considérons que, pour chaque paire état-action (s, a), on dispose d'un vecteur $\vec{\phi}_{s,a}$ de caractéristiques. Dans chaque état s, la politique doit fournir une distribution de probabilité sur les actions a en fonction des vecteurs $\vec{\phi}_{s,a}$, ce que l'on peut faire en deux étapes :

1) calculer une grandeur scalaire $K_s(a)$ pour tout a (mesurant « l'importance » de a dans la situation courante) et

2) transformer ces grandeurs en une distribution de probabilité sur les actions.

Exemple 1 *Un exemple type d'une telle définition de politique paramétrée est alors de prendre :*

– pour $K_s(a)$ une combinaison linéaire des caractéristiques de (s, a) — un produit scalaire entre $\vec{\phi}_{s,a}$ et un vecteur de paramètres $\vec{\theta}$: $K_s(a) = \vec{\theta}^T \vec{\phi}_{s,a}$ — et

– d'en tirer des probabilités d'action à l'aide d'une distribution de Gibbs, comme suit :

$$q_\pi(a|s) = q(a|s;\vec{\theta}) = \frac{e^{K_s(a)}}{\sum_{b\in A} e^{K_s(b)}}, \tag{12.2}$$

où $q_\pi(a|s)$ est la probabilité d'exécuter l'action a dans l'état s sous la politique stochastique π.

En précisant la forme prise par les vecteurs caractéristiques, l'exemple 1 conduit aux deux formulations courantes suivantes :

– **Exemple 2** *si l'état s est connu, définissons les vecteurs $\vec{\phi}_{s,a}$ comme étant tous de taille $|S| * |A|$, chaque composante $\phi_{s,a}(s', a')$ correspondant à une paire état-action et étant nulle, sauf pour celle de la paire (s, a) qui vaut 1 ; on a ainsi un paramètre $\theta_{s,a}$ par paire état-action, d'où :*

$$q(a|s;\vec{\theta}) = \frac{e^{\theta_{s,a}}}{\sum_{b\in A} e^{\theta_{s,b}}};\ \textit{et} \tag{12.3}$$

– **Exemple 3** *si l'état est connu à travers un vecteur caractéristique $\vec{\phi}_s$ (dépendant des observations, pas des actions), on peut décomposer $\vec{\theta}$ en un vecteur $\vec{\theta}_a$ par action et réécrire :*

$$q(a|s;\vec{\theta}) = \frac{e^{\vec{\theta}_a^T \vec{\phi}_s}}{\sum_{b\in A} e^{\vec{\theta}_b^T \vec{\phi}_s}}. \tag{12.4}$$

Le second cas (exemple 3) pose problème s'il existe un état s tel que $\vec{\phi}_s = \vec{0}$, puisqu'on a alors $q_\pi(a|s) = 1/|\mathcal{A}|$ pour tout $a \in \mathcal{A}$. Il est usuel de résoudre ce problème en ajoutant une caractéristique constante non nulle[2] (un bit à 1) à $\vec{\phi}$. Mais d'autres paramétrisations que celles-ci sont envisageables : à l'aide de perceptrons multi-couches, d'arbres de décision, de modèles graphiques, etc.

Remarques

Le choix de l'écriture paramétrée de $q_\pi(a|s)$ définit un sous-espace de l'espace des politiques stochastiques. Idéalement, il faut que ce sous-espace soit le plus petit possible, mais qu'il contienne les meilleures politiques. Dans ce sous-espace, il n'existe plus nécessairement de politique optimale déterministe et une recherche peut tomber dans un optimum local.

Le gradient de f en un point $\vec{\theta}$ représente la sensibilité de la mesure de performance f par rapport à ces paramètres de contrôle. Si un modèle du MDP est connu, il est envisageable de faire un calcul exact de ce gradient (voir l'algorithme GAMP

2. C'est tout à fait équivalent au paramètre seuil dans un neurone selon le modèle de McCulloch et Pitts.

[ABE 03]). Mais souvent on l'estime par l'expérience, soit parce qu'aucun modèle n'est disponible, soit parce qu'une méthode exacte serait trop coûteuse (requérant de développer l'espace des états et des actions).

Dans les sections qui suivent, nous allons voir comment peut être faite l'estimation de ce gradient dans différentes situations.

12.2.2. *Estimation du gradient de f dans un* MDP*, cas de l'horizon temporel fini*

12.2.2.1. *Horizon temporel 1*

Considérons d'abord un MDP à horizon temporel fini de longueur 1 et dont l'état initial est tiré au hasard selon une distribution de probabilité d. On cherche alors à optimiser

$$J(\pi) = E_\pi[r] = \sum_{(s,a,s')\in S\times A\times S} \underbrace{d(s)q_\pi(a|s)p(s'|s,a)}_{P(s,a,s')} r(s,a,s').$$

Avec la paramétrisation par $\vec{\theta}$, cette mesure de performance se réécrit :

$$f(\vec{\theta}) = \sum_{s,a,s'} d(s)q(a|s;\vec{\theta})p(s'|s,a)r(s,a,s').$$

On voit immédiatement que l'application d'une montée de gradient nécessite que, pour toute paire (s,a), la fonction qui à $\vec{\theta}$ associe $q(a|s;\vec{\theta})$ soit différentiable. En supposant que les rapports de vraisemblance $\frac{\vec{\nabla}q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})}$ existent (et sont bornés), le gradient de f s'écrit alors :[3]

$$\begin{aligned}
\vec{\nabla}E\left[r(s,a,s')\right] &= \vec{\nabla}\left[\sum_{s,a,s'} d(s)q(a|s;\vec{\theta})p(s'|s,a)r(s,a,s')\right] \\
&= \sum_{s,a,s'} d(s)\vec{\nabla}\left[q(a|s;\vec{\theta})\right]p(s'|s,a)r(s,a,s') \\
&= \sum_{s,a,s'} d(s)\left[\frac{\vec{\nabla}q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})}q(a|s;\vec{\theta})\right]p(s'|s,a)r(s,a,s') \\
&= E\left[\frac{\vec{\nabla}q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})}r(s,a,s')\right].
\end{aligned}$$

3. Les gradients sont toujours pris par rapport au vecteur $\vec{\theta}$.

Ce résultat permet, avec N échantillons (s_i, a_i, s'_i) indépendants et identiquement distribués selon les distributions d, q et p, d'estimer le gradient de f au point $\vec{\theta}$ par :

$$\tilde{\nabla} f(\vec{\theta}) \quad = \quad \frac{1}{N} \sum_{i=1}^{N} \frac{\vec{\nabla} q(a_i|s_i;\vec{\theta})}{q(a_i|s_i;\vec{\theta})} r(s_i, a_i, s'_i).$$

Pour exploiter ce résultat dans un algorithme de montée de gradient, il faut donc ajouter à l'algorithme 12.1 une boucle produisant ces N échantillons pour estimer le gradient pour le vecteur de paramètres courant.[4] Mais il faut aussi pouvoir calculer $\frac{\vec{\nabla} q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})}$ (appelé parfois log-gradient de la politique parce qu'il peut s'écrire $\vec{\nabla} \log q(a|s;\vec{\theta})$). Dans le cas de la formulation (12.2) par exemple, on a :

$$\begin{aligned}
\frac{1}{q(a|s;\vec{\theta})} \frac{\partial q(a|s;\vec{\theta})}{\partial \theta_i} \quad &= \quad \frac{1}{q(a|s;\vec{\theta})} \left(\phi_{s,a,i} \frac{e^{\vec{\theta}^T \vec{\phi}_{s,a}}}{\sum_{a' \in \mathcal{A}} e^{\vec{\theta}^T \vec{\phi}_{s,a'}}} \right. \\
&\qquad \left. - \frac{e^{\vec{\theta}^T \vec{\phi}_{s,a}}}{\left[\sum_{a' \in \mathcal{A}} e^{\vec{\theta}^T \vec{\phi}_{s,a'}} \right]^2} \sum_{b \in \mathcal{A}} \phi_{s,b,i} \; e^{\vec{\theta}^T \vec{\phi}_{s,b}} \right) \\
&= \quad \phi_{s,a,i} - \sum_{b \in \mathcal{A}} \phi_{s,b,i} \; q(b|s;\theta),
\end{aligned}$$

ce qui permet d'écrire :

$$\frac{\vec{\nabla} q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})} \quad = \quad \vec{\phi}_{s,a} - \sum_{b \in \mathcal{A}} q(b|s;\theta) \; \vec{\phi}_{s,b}. \tag{12.5}$$

12.2.2.2. *Horizon temporel T*

NOTE.– Un procédé estimant une grandeur par échantillonnage est en général appelé méthode de Monte-Carlo. Dans le cas présent, on simule des trajectoires sur une chaîne de Markov sur les états du MDP et dont la probabilité de transiter de s à s' est $P(s'|s) = \sum_{a \in A} p(s'|s,a) q(a|s;\vec{\theta})$. On parle alors de méthode de Monte-Carlo par chaîne de Markov (MCMC) [MAC 03].

Dans la section précédente, nous avons montré comment est calculé un estimateur du gradient de f dans le cas d'un MDP à horizon temporel fini de longueur 1. Nous étendons maintenant ce calcul au cas d'un MDP à horizon temporel fini de longueur T.

4. Sans oublier de changer le signe devant α puisqu'on passe d'une descente à une montée de gradient.

La mesure de performance considérée est l'espérance de récompense actualisée avec $\gamma \in]0,1]$. Soit $V_t(s_t)$ l'espérance de récompense actualisée de t à T en partant de l'état s. On a, pour tout $s \in \mathcal{S}$ et tout $t \in \{1, \ldots, T-1\}$:

$$\begin{aligned} f(\vec{\theta}) &= E\left[V_0(s_0)\right] = \sum_s d(s) V_0(s), \\ V_t(s) &= E\left[r(s_t, a_t, s_{t+1}) + \gamma V_{t+1}(s_{t+1}) | s_t = s\right] \\ &= \sum_{a,s'} q(a|s;\vec{\theta}) p(s'|s,a) \left(r(s,a,s') + \gamma V_{t+1}(s')\right) \text{ et} \\ V_T(s) &= 0. \end{aligned}$$

Le gradient de f peut alors se calculer par :

$$\begin{aligned} \vec{\nabla} f(\vec{\theta}) &= \sum_s d(s) \vec{\nabla} V_0(s) = E\left[\vec{\nabla} V_0(s_0)\right], \\ \vec{\nabla} V_t(s) &= \sum_{a,s'} \vec{\nabla}\left[q(a|s;\vec{\theta}) p(s'|s,a) \left(r(s,a,s') + \gamma V_{t+1}(s')\right)\right] \\ &= \sum_{a,s'} q(a|s;\vec{\theta}) p(s'|s,a) \left(\frac{\vec{\nabla} q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})} \left(r(s,a,s') + \gamma V_{t+1}(s')\right) \right. \\ &\quad \left. + \gamma \vec{\nabla} V_{t+1}(s') \right) \\ &= E\left[\frac{\vec{\nabla} q(a_t|s_t;\vec{\theta})}{q(a_t|s_t;\vec{\theta})} \left(r_t + \gamma V_{t+1}(s_{t+1})\right) + \gamma \vec{\nabla} V_{t+1}(s_{t+1}) | s_t = s \right], \\ &\quad \text{où } r_t = r(s_t, a_t, s_{t+1}) \text{ et} \\ \vec{\nabla} V_T(s) &= \vec{0}. \end{aligned}$$

En développant le calcul de $\vec{\nabla} f(\vec{\theta})$ et en factorisant les termes associés à chaque récompense, on obtient :

$$\vec{\nabla} f(\vec{\theta}) = E\left[\sum_{t=0}^{T-1} \gamma^t r_t \sum_{t'=0}^{t-1} \frac{\vec{\nabla} q(a_{t'}|s_{t'};\vec{\theta})}{q(a_{t'}|s_{t'};\vec{\theta})} \right].$$

Comme dans le cas à horizon 1, on va pouvoir estimer le gradient par échantillonnage. Lors de l'acquisition d'un échantillon $s_0, a_0, \ldots, s_T$, on calcule un estimateur

non biaisé $\vec{g}$ de $\vec{\nabla} f(\vec{\theta})$ en itérant :

$$\begin{aligned} \vec{z}_{t+1} &= \vec{z}_t + \frac{\vec{\nabla} q(a_t|s_t;\vec{\theta})}{q(a_t|s_t;\vec{\theta})} \text{ et} \\ \vec{g}_{t+1} &= \vec{g}_t + \gamma^t r_t \vec{z}_t, \end{aligned}$$

où $\vec{z}$ est une trace d'éligibilité (estimation locale du log-gradient de la politique) et les deux suites sont initialisées avec le vecteur nul. On obtient un meilleur estimateur (de plus faible variance) en calculant la moyenne de N estimateurs $\vec{g}$ calculés indépendamment les uns des autres.

L'algorithme 12.2 récapitule les opérations à réaliser pour effectuer une montée de gradient dans un MDP à horizon fini. La récompense instantanée r et le prochain état s' sont obtenus par interaction avec un environnement réel ou un simulateur.

Quelques remarques sur cet algorithme :

– Le vecteur initial $\vec{\theta}$ doit idéalement permettre une bonne exploration. Un vecteur nul est souvent une bonne solution, sauf par exemple quand on utilise un perceptron multi-couches, puisque $\vec{\theta} = \vec{0}$ correspond alors à un point de gradient nul. On préfère alors un vecteur aléatoire de petite norme.

– Le choix de N est important : trop petit, l'estimation du gradient sera mauvaise ; trop grand, le temps de calcul d'une estimation sera trop long. Un problème intéressant est de trouver un estimateur dont la variance soit plus faible à nombre d'échantillons égal. Intuitivement, il s'agit de trouver un estimateur dont la probabilité de fournir une estimation proche de la vraie valeur est plus grande.

– Une caractéristique intéressante de l'estimation du gradient est que, à supposer que l'on utilise un simulateur, elle est très facilement parallélisable : N processeurs peuvent effectuer les N simulations en parallèle (gain linéaire).

12.2.3. *Extension au cas de l'horizon temporel infini : critère actualisé, critère moyen*

Dans une méthode MCMC, il faut s'assurer que la fréquence de visite des états reflète la probabilité réelle d'être dans ces états. Dans le cas à horizon temporel fini, il a suffi pour cela de commencer les simulations dans les états initiaux possibles en les tirant au sort selon la distribution $d(\cdot)$, puis d'exécuter la politique et les transitions selon le modèle, jusqu'à un redémarrage. De cette manière, tout état est visité avec une fréquence représentative de la probabilité d'être dans cet état à un instant quelconque.

Si l'horizon temporel est infini, on ne peut plus travailler sur des trajectoires complètes –puisque les trajectoires sont sans fin– et les états ne sont visités avec une fréquence représentative qu'après un temps de simulation suffisamment long. Dans le cas de la chaîne de Markov représentée par la figure 12.3 par exemple, le temps d'attente

Algorithme 12.2 : Montée de gradient pour MDP à horizon fini $(\vec{\theta}, \alpha, N)$

Initialisation :
$\vec{\theta_0} \leftarrow \vec{\theta}$
$i \leftarrow 0$
`/* Boucle de la montée de gradient. */`
pour *toujours* **faire**
  $i \leftarrow i + 1$
  $\tilde{\nabla} f \leftarrow \vec{0}$
  `/* Boucle recueillant` N `échantillons pour estimer le gradient. */`
  pour $n = 1, \ldots, N$ **faire**
    $\vec{z} \leftarrow \vec{0}$
    $\vec{g} \leftarrow \vec{0}$
    ReçoitEtatInitial(s)
    `/* Boucle simulant le MDP pour obtenir un échantillon. */`
    pour $t = 0, \ldots, T-1$ **faire**
      Echantillonne$(a, q(\cdot|s;\vec{\theta}))$
      Exécute(a)
      Reçoit(s', r)
      $\vec{z} \leftarrow \vec{z} + \frac{\vec{\nabla} q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})}$
      $\vec{g} \leftarrow \vec{g} + \gamma^t r \vec{z}$
      $s \leftarrow s'$
    $\tilde{\nabla} f \leftarrow \tilde{\nabla} f + \frac{1}{N} \vec{g}$
  $\vec{\theta_i} \leftarrow \vec{\theta}_{i-1} + \alpha \tilde{\nabla} f$
retourner x_n

moyen avant d'arriver dans l'état s_2 est d'autant plus long que ϵ est petit. Pourtant, une fois le « régime permanent » atteint, la distribution de probabilité sur les états possibles est toujours donnée par $P(s_0) = P(s_1) = 0$ et $P(s_2) = 1$, quel que soit $\epsilon \in]0, 1]$.

Pour pouvoir étendre les algorithmes de gradient au cas de l'horizon temporel infini, il est usuel de s'assurer qu'un tel régime permanent existe. Il faut ainsi que soit vérifiée l'hypothèse que, pour un vecteur de paramètres $\vec{\theta}$ donné, la chaîne de Markov induite est ergodique (voir figure 12.4), ce qui garantit qu'il existe une unique distribution P_{stat} sur les états telle que :

$$\forall s \in S,\ P_{\text{stat}}(s) = \sum_{s' \in S} \sum_{a \in A} p(s|a, s') q(a|s', \vec{\theta}) P_{\text{stat}}(s').$$

A titre d'exemple, la figure 12.4 montre deux cas types de chaînes non-ergodiques.

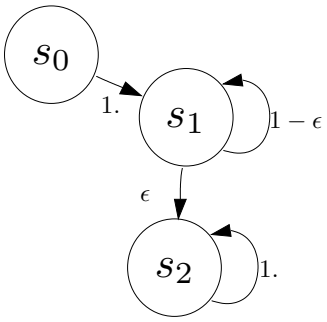


Figure 12.3. *Chaîne de Markov dont le temps d'établissement du « régime permanent » est d'autant plus long que ϵ est petit*

Figure 12.4. *Deux principales raisons pour qu'une chaîne de Markov ne soit pas ergodique : à gauche une chaîne périodique, à droite une chaîne avec deux sous-graphes absorbants (deux classes récurrentes, chacune réduite à un état)*

On considère ici uniquement le cas $\gamma = 1$.

12.2.3.1. *Cas d'un processus régénératif*

Le problème qui se pose est d'estimer le gradient. Une première approche peut être employée quand un état récurrent s_r existe, c'est-à-dire un état dans lequel le système revient nécessairement. On parle alors de *processus régénératif*. Dans un tel cas de figure, les échantillons employés sont les séquences d'états entre deux passages par l'état s_r et l'estimation (non biaisée) du gradient se fait de manière très comparable au cas de l'horizon temporel T (section 12.2.2.2). Toutefois, il est difficile de garantir qu'un processus est régénératif et de spécifier un état récurrent s_r à utiliser. On va donc chercher d'autres moyens d'estimer le gradient.

Si on prend pour échantillons des séquences infinies, il n'est plus possible de recueillir des échantillons un par un pour faire des mises à jour de l'estimation du gradient. Il faut donc trouver un autre moyen de faire une estimation du gradient, à l'aide d'échantillons partiels.

12.2.3.2. *Utilisation d'une fenêtre flottante*

Une deuxième approche consiste à prendre pour échantillons des séquences d'états de longueur fixe n. Plus précisément, à l'instant t, la trace d'éligibilité est calculée sur la base des n derniers instants :

$$\begin{aligned}\vec{z}_t(n) &= \sum_{t'=t-n+1}^{t} \frac{\vec{\nabla} q(a_{t'}|s_{t'};\vec{\theta})}{q(a_{t'}|s_{t'};\vec{\theta})} \\ &= \vec{z}_{t-1}(n) + \frac{\vec{\nabla} q(a_t|s_t;\vec{\theta})}{q(a_t|s_t;\vec{\theta})} - \frac{\vec{\nabla} q(a_{t-n}|s_{t-n};\vec{\theta})}{q(a_{t-n}|s_{t-n};\vec{\theta})}.\end{aligned}$$

La trace d'éligibilité peut toujours se calculer de manière itérative, mais il faut pour cela mémoriser les $n+1$ dernières étapes. De là, l'estimation du gradient après T étapes est donnée par :

$$\vec{\nabla} f(\vec{\theta}) = \frac{1}{T-n+1} \sum_{t=n-1}^{T-1} \vec{z}_t(n) r_t.$$

Cette estimation est biaisée. Mais si le biais diminue quand n grandit, c'est aux dépens de la variance qui, elle, diverge. Le choix de n correspond donc à un compromis entre biais et variance.

12.2.3.3. *Utilisation d'un coefficient d'atténuation*

Une troisième approche existe qui permet de ne pas avoir à mémoriser n étapes. Il s'agit de calculer une trace d'éligibilité atténuée et non tronquée, comme suit :

$$\vec{z}_{t+1} = \beta \vec{z}_t + \frac{\vec{\nabla} q(a_{t+1}|s_{t+1};\vec{\theta})}{q(a_{t+1}|s_{t+1};\vec{\theta})},$$

où $\vec{z}_0 = 0$ et $\beta \in [0,1[$ est un coefficient d'atténuation. Après T pas de temps, l'estimation (biaisée) du gradient est alors donnée par :

$$\vec{\nabla} f_\beta(\vec{\theta}) = \frac{1}{T} \sum_{t=0}^{T-1} \vec{z}_t(\beta) r_t.$$

Le compromis entre biais et variance est toujours présent, dépendant ici du paramètre β : le biais tend vers 0 quand β tend vers 1, mais alors la variance diverge.

Sur cette base, on peut proposer un algorithme de montée de gradient qui, comme les précédents algorithmes rencontrés, alternera une phase d'estimation du gradient avec un pas de suivi du gradient. Mais on peut aussi proposer un algorithme de montée

de gradient « en ligne », c'est-à-dire qui suit la direction du gradient à chaque nouvelle expérience. Il n'y a pas de calcul d'une estimation du gradient, puisqu'un gradient instantané est utilisé à la place ($\vec{z}_t(\beta)r_t$). Cet algorithme est connu sous le nom de `OLpomdp` (« `OL` » pour « *on-line* » et « `pomdp` » parce que cet algorithme reste valide dans un cadre partiellement observable[5]) et décrit dans l'algorithme 12.3.

Algorithme 12.3 : `OLpomdp`$(\vec{\theta}, \alpha, \beta)$
Montée de gradient en ligne pour MDP (PO) à horizon infini

Initialisation :
$\vec{\theta_0} \leftarrow \vec{\theta}$
$\vec{z} \leftarrow \vec{0}$
$\vec{g} \leftarrow \vec{0}$
ReçoitEtatInitial(s)
pour *toujours* **faire**
  Echantillonne$(a, q(\cdot|s;\vec{\theta}))$
  Exécute(a)
  Reçoit(s', r)
  $\vec{z} \leftarrow \beta\vec{z} + \frac{\vec{\nabla} q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})}$
  $\vec{\theta} \leftarrow \vec{\theta} + \gamma r \vec{z}$
  $s \leftarrow s'$
retourner $\vec{\theta}$

Cet algorithme a l'avantage d'être très simple à mettre en œuvre, et s'est avéré efficace dans diverses applications (voir chapitre 15).

12.2.4. *Cas partiellement observable*

Intéressons-nous un instant à l'utilisation de telles montées de gradient pour des MDP partiellement observables (avec un ensemble d'observations fini). Diverses options s'offrent à nous, parmi lesquelles on peut citer :

– comme au chapitre 3, se ramener à un MDP sur les états de croyance et

– optimiser une politique dont l'entrée dépend de l'historique des observations et actions passées.

Si l'horizon temporel est fini, alors ces deux options sont envisageables en se ramenant à un nouveau MDP à espace d'états fini.

Si l'horizon temporel est infini, la première option requiert la connaissance d'un modèle pour estimer l'état de croyance à chaque instant et suppose que l'on sache

5. Nous revenons sur l'observabilité partielle dans la section suivante.

étendre les algorithmes de montée de gradient à des MDP sur des espaces d'état continus. Nous allons préférer la seconde option, en limitant l'entrée à la dernière observation o perçue.

Pour voir comment traiter le cas POMDP, notons que l'observabilité partielle recouvre deux aspects :

1) le fait que l'information reçue sur l'état est incomplète et

2) le fait que cette information est bruitée.

Or le bruit peut être « extrait » de la fonction d'observation et mis dans l'état : si l'état est défini par un ensemble de variables aléatoires, le bruit n'est qu'une variable supplémentaire (non observée) B, indépendante du passé et des autres variables. Ceci fait, la fonction d'observation devient déterministe ; à tout état s correspond une unique observation $o = O(s)$. De là, la composition de $O(\cdot)$ et $q(a|o;\vec{\theta})$ permet de se ramener au cas d'un MDP, la perte d'information due à l'observation partielle entrant dans la fonction d'approximation qui définit la politique :

$$\begin{aligned} q(a|s;\vec{\theta}) &= \sum_{o'} q(a|o';\vec{\theta}) O(o'|s) \\ &= q(a|O(s);\vec{\theta}). \end{aligned}$$

La figure 12.5 présente un POMDP (et le contrôleur associé) de manière schématique en mettant en évidence les composantes « bruit » et « perception incomplète ». La figure 12.6 montre le même problème sous la forme d'un MDP et d'un contrôleur modifié.

Cette interprétation explique comment les algorithmes de montée de gradient s'étendent dans un certain nombre de cas aux POMDP. C'est en particulier le cas d'`OLpomdp` (algorithme 12.3) en remplaçant toute occurrence de l'état s par l'observation o. Mais ce n'est pas le cas par exemple des algorithmes pour processus régénératifs puisque l'observabilité partielle peut empêcher l'identification d'un état récurrent.

12.3. Méthodes "Acteur-Critique"

Les méthodes "Acteur-Critique" effectuent elles aussi une recherche dans l'espace des politiques, mais elles font en plus usage d'une représentation de la fonction de valeur.

Les algorithmes vus dans la section précédente se basent sur une estimation du gradient dont la variance peut être importante, ce qui rend l'optimisation des paramètres difficile. Un objectif essentiel est donc de réduire cette variance, objectif qui peut être atteint par exemple grâce à des estimations basées sur la fonction de valeur (ou une approximation de celle-ci). C'est ainsi qu'ont été introduits l'algorithme VAPS (*Value and Policy Search*) [BAI 99a, BAI 99b] et ses descendants (on citera par exemple

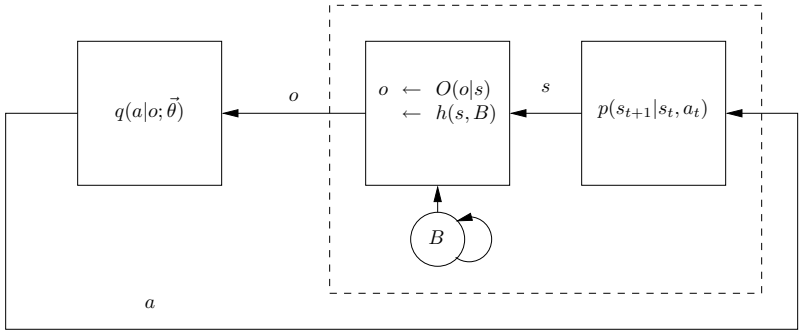


Figure 12.5. *Schéma de principe d'un* MDP *partiellement observable et de son contrôleur*

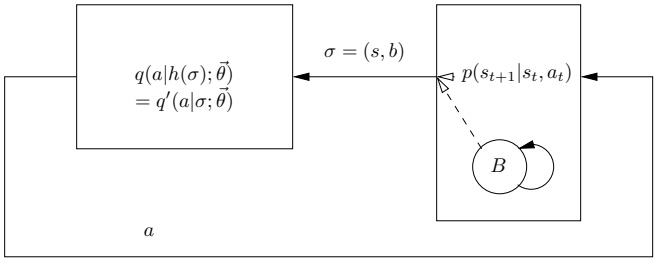


Figure 12.6. *Schéma de principe du même problème, l'observabilité partielle étant intégrée dans le contrôleur*

[SUT 00]). Ils combinent une montée de gradient et l'approximation de la fonction de valeur, ce qui les fait entrer dans la famille des architectures acteur-critique présentées en section 2.4.5.

Nous revenons ici dans le cadre d'un MDP à horizon temporel fini, avec un facteur d'actualisation γ.

12.3.1. *Estimateur du gradient utilisant les Q-valeurs*

Avant toute chose, introduisons $d^\pi(s)$ (à ne pas confondre avec $d(s)$), qui représente le poids de l'état s étant donnés 1) la probabilité de le retrouver dans le futur et 2) le coefficient d'atténuation γ : $d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t Pr(s_t = s|d(\cdot);\pi)$.

Pour calculer un estimateur du gradient utilisant la fonction des Q-valeurs $Q(s,a)$, partons de l'équation de Bellman :

$$V^{\vec{\theta}}(s) \quad = \quad \sum_{a\in\mathcal{A}} q(a|s;\vec{\theta}) \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[r(s,a,s') + \gamma V^{\vec{\theta}}(s) \right]$$

et dérivons son gradient :

$$\begin{aligned}
\vec{\nabla} V^{\vec{\theta}}(s) &= \sum_{a\in\mathcal{A}} \vec{\nabla} q(a|s;\vec{\theta}) \sum_{s'\in\mathcal{S}} p(s'|s,a)\left[r(s,a,s') + \gamma V^{\vec{\theta}}(s)\right] \\
&\quad + \sum_{a\in\mathcal{A}} q(a|s;\vec{\theta}) \sum_{s'\in\mathcal{S}} p(s'|s,a)\vec{\nabla}\left[r(s,a,s') + \gamma V^{\vec{\theta}}(s)\right] \\
&= \sum_{a\in\mathcal{A}} q(a|s;\vec{\theta}) \sum_{s'\in\mathcal{S}} p(s'|s,a) \\
&\quad \left(\frac{\nabla q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})}\left[r(s,a,s') + \gamma V^{\pi_{\vec{\theta}}}(s)\right] + \gamma\vec{\nabla} V^{\vec{\theta}}(s)\right) \\
&= \sum_{a\in\mathcal{A}} q(a|s;\vec{\theta}) \sum_{s'\in\mathcal{S}} p(s'|s,a)\left(\frac{\vec{\nabla} q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})} Q^{\vec{\theta}}(s,a) + \gamma\vec{\nabla} V^{\vec{\theta}}(s)\right).
\end{aligned}$$

On reconnaît en cette dernière équation une équation de Bellman définissant le gradient de la fonction de valeur à l'aide de $\frac{\vec{\nabla} q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})} Q^{\vec{\theta}}(s,a)$ (au lieu de $r(s,a,s')$ dans la définition de la fonction de valeur). De là, on déduit que ce gradient s'écrit aussi :

$$\vec{\nabla} V^{\vec{\theta}}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t \frac{\nabla q(a_t|s_t;\vec{\theta})}{q(a_t|s_t;\vec{\theta})} Q^{\vec{\theta}}(s,a)|s_0 = s;\pi\right],$$

ce qui nous amène à l'estimateur du gradient de f :

$$\begin{aligned}
\vec{\nabla} f(\vec{\theta}) &= \sum_{s\in\mathcal{S}} d(s)\vec{\nabla} V^{\vec{\theta}}(s), \text{ qui peut aussi s'écrire} \\
&= \sum_{s\in\mathcal{S}} d^{\pi}(s) \sum_{a\in\mathcal{A}} q(a_t|s_t;\vec{\theta})\left[\frac{\vec{\nabla} q(a_t|s_t;\vec{\theta})}{q(a_t|s_t;\vec{\theta})} Q^{\vec{\theta}}(s,a)\right] \\
&= \sum_{s\in\mathcal{S}} d^{\pi}(s) \sum_{a\in\mathcal{A}} \vec{\nabla} q(a_t|s_t;\vec{\theta}) Q^{\vec{\theta}}(s,a). \qquad (12.6)
\end{aligned}$$

NOTE.– Les mêmes résultats peuvent être obtenus avec comme critère la récompense moyenne par pas de temps [SUT 00].

12.3.2. *Compatibilité avec l'approximation d'une fonction de valeur*

Une des motivations pour employer des méthodes de recherche directe de politique est de réduire les besoins en mémoire et en temps de calcul en limitant l'espace de

recherche à un sous-espace des politiques possibles. Dans le cas des méthodes acteur-critique, on va donc naturellement s'intéresser à approximer la fonction $Q^{\vec{\theta}}(s,a)$ plutôt que de faire un calcul exact. Une question qui se pose en particulier est de savoir si l'estimateur vu précédemment reste valide.

12.3.2.1. *Approximation de $Q^{\vec{\theta}}$*

Soit $Q^{\vec{w}} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ cette approximation de $Q^{\vec{\theta}}$. L'apprentissage des paramètres $\vec{w}$ peut se faire en parcourant les paires état-action (s,a) et en mettant à jour $\vec{w}$ en suivant la direction

$$\begin{aligned} \vec{\delta w} &\propto \vec{\nabla}[\hat{Q}^{\pi}(s,a) - Q^{\vec{w}}(s,a)]^2 \\ &\propto [\hat{Q}^{\pi}(s,a) - Q^{\vec{w}}(s,a)]\vec{\nabla}Q^{\vec{w}}(s,a), \end{aligned}$$

où $\hat{Q}^{\pi}(s,a)$ est un estimateur non-biaisé de $Q^{\pi}(s,a)$ et $\propto$ est le symbole « proportionnel à ». Une fois un optimum local atteint, on a :

$$\sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} q(a|s;\vec{\theta})[Q^{\pi}(s,a) - Q^{\vec{w}}(s,a)]\vec{\nabla}Q^{\vec{w}}(s,a) = \vec{0}. \tag{12.7}$$

12.3.2.2. *Compatibilité des approximateurs*

On peut retrouver le gradient de f de l'équation (12.6) dans l'équation (12.7) ci-dessus, s'il est vérifié que :

$$\vec{\nabla}Q^{\vec{w}}(s,a) = \frac{\vec{\nabla}q(a|s;\vec{\theta})}{q(a|s;\vec{\theta})}. \tag{12.8}$$

THÉORÈME 12.1 [SUT 00].– *Si cette condition, dite de* compatibilité*, est vérifiée, alors le gradient de f s'écrit :*

$$\vec{\nabla}f(\vec{\theta}) = \sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} \vec{\nabla}q(a_t|s_t;\vec{\theta})Q^{\vec{w}}(s,a).$$

PREUVE.– De (12.7) et (12.8), on tire :

$$\sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} \vec{\nabla}q(a|s;\vec{\theta})[Q^{\pi}(s,a) - Q^{\vec{w}}(s,a)] = \vec{0}.$$

En soustrayant cette expression (de valeur nulle) à (12.6), on obtient :

$$\begin{aligned}\vec{\nabla} f(\vec{\theta}) &= \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \vec{\nabla} q(a_t|s_t;\vec{\theta}) Q^\pi(s,a) \\ &\quad - \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \vec{\nabla} q(a|s;\vec{\theta}) [Q^\pi(s,a) - Q^{\vec{w}}(s,a)] \\ &= \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \vec{\nabla} q(a_t|s_t;\vec{\theta}) Q^{\vec{w}}(s,a).\end{aligned}$$

Le théorème 12.1 permet de trouver une paramétrisation de la fonction de valeur qui, étant donnée la forme choisie pour la politique, garantit que l'estimation du gradient de la politique n'est pas biaisée. Avec l'exemple de l'équation (12.2) (page 361), en reprenant le log-gradient (12.5) (page 363) on a immédiatement une paramétrisation possible :

$$Q^{\vec{w}}(s,a) = \vec{w}^T \left[\vec{\phi}_{s,a} - \sum_{b\in\mathcal{A}} q(b|s;\vec{\theta})\ \vec{\phi}_{s,b} \right]. \tag{12.9}$$

$Q^{\vec{w}}$ doit donc être linéaire en les mêmes caractéristiques que la politique, avec une normalisation à une moyenne nulle pour chaque état. En observant que

$$\sum_{a\in\mathcal{A}} q(a|s;\vec{\theta}) Q^{\vec{w}}(s,a) = 0,$$

il apparaît que c'est plutôt la fonction *avantage* $A^\pi(s,a) = Q^\pi(s,a) - V(s)$ [BAI 93] qui est approchée par $Q^{\vec{w}}(s,a)$. Cela n'est pas surprenant puisque, dans un état s donné, c'est la différence relative de valeur entre les actions qui informe réellement de leurs intérêts relatifs.

12.3.2.3. *Algorithme acteur-critique*

Comme on peut le voir dans l'algorithme 12.4, le principe de l'algorithme de montée de gradient utilisant une approximation de la fonction de valeur est simple. Si la procédure calculant w_k est itérative, on peut l'initialiser avec la solution w_{k-1}, la fonction à approcher ne changeant souvent que peu pendant un pas de gradient.

THÉORÈME.– *Si q et $Q^{\vec{w}}$ sont deux approximateurs de fonction différentiables, satisfaisant la condition de compatibilité (12.8) et tels que* $\max_{\vec{\theta},s,a,i,j} |\frac{\partial^2 q(a|s;\vec{\theta})}{\partial\theta_i \partial\theta_j}| < B < \infty$. *Soit* $(\alpha_k)_{k=0}^\infty$ *une suite de tailles de pas telle que* $\lim_{k\to\infty} \alpha_k = 0$ *et* $\sum_{k=0}^\infty \alpha_k = \infty$. *Alors, pour tout* MDP *avec des récompenses bornées, la suite* $(\theta_k)_{k=0}^\infty$ *calculée par l'algorithme 12.4 converge telle que* $\lim_{k\to\infty} \vec{\nabla} f(\vec{\theta}_k) = 0$.

Algorithme 12.4 : Montée de gradient avec approximation de la fonction de valeur
(algorithme acteur-critique) $(\vec{\theta}, (\alpha_k)_{k=0}^{\infty})$

Initialisation :
$i \leftarrow 0$
$\vec{\theta}_0 \leftarrow \vec{\theta}$
répéter
  $k \leftarrow k+1$
  $w_k \leftarrow$ solutionDe(
  $\sum_{s\in\mathcal{S}} d^{\pi_k} \sum_{a\in\mathcal{A}} q(a|s;\vec{\theta}_k)[Q^{\pi_k}(s,a) - Q^{\vec{w}}(s,a)]\vec{\nabla}Q^{\vec{w}}(s,a) = \vec{0}$
)
  $\vec{\theta}_{k+1} \leftarrow \vec{\theta}_k + \alpha_k \sum_{s\in\mathcal{S}} d^{\pi_k}(s) \sum_{a\in\mathcal{A}} \vec{\nabla}q(a|s;\vec{\theta}_k)Q^{\vec{w}_k}(s,a)$
jusqu'à `faux`
retourner $\vec{\theta}_k$

12.4. Compléments

Ces méthodes de gradient (avec ou sans approximation de la fonction de valeur) posent des problèmes variés dont les solutions permettraient d'améliorer les algorithmes existants. Nous abordons ici brièvement quelques-uns de ces problèmes.

Réduction de la variance

Dans une méthode de gradient stochastique, un premier problème est de trouver un bon estimateur du gradient, ce qui veut dire qu'on cherche un estimateur :

– de faible biais, c'est-à-dire qui, quand on cherche $\vec{x}$, estime $\vec{x} + \delta\vec{x}$ avec $|\delta\vec{x}|$ petit ; et

– de faible variance, c'est-à-dire très stable (renvoyant une valeur proche de $\vec{x}$ avec une grande probabilité).

L'existence d'un biais n'empêche pas le bon fonctionnement de l'algorithme : l'important est de suivre une direction de montée (dans notre cadre de maximisation), pas nécessairement celle du gradient. Et il est parfois préférable d'introduire un biais dans un estimateur pour en réduire la variance. Cela peut permettre d'utiliser moins d'échantillons.

Nous avons déjà évoqué le fait que l'utilisation d'une approximation de la fonction de valeur a pour but de réduire la variance de l'estimation du gradient. Dans ce même but, il est aussi courant dans les algorithmes directs de soustraire à toutes les récompenses une « ligne de base » (*baseline*), en général la récompense moyenne estimée. Greensmith et al. [GRE 01] ont toutefois prouvé que la récompense moyenne n'est pas toujours le meilleur choix. [MUN 06] discute plus particulièrement le problème de la réduction de variance dans le cas des algorithmes acteur-critique.

Gradient naturel

Nous abordons ici une difficulté particulière : le fait que le gradient ne donne pas nécessairement la direction de plus grande pente. Nous décrivons très intuitivement cette difficulté et une façon de la contourner. De plus amples détails peuvent être trouvés dans [AMA 98, KAK 02, BAG 03, PET 05], un algorithme actuellement populaire étant NAC (*Natural Actor-Critic*).

En choisissant une paramétrisation de la politique, on définit en fait 1) un sous-ensemble de l'ensemble des politiques stochastiques et 2) une façon de le parcourir. En mathématiques, un tel objet est appelé une *variété*. Des variétés typiques en trois dimensions sont la sphère, le tore ou la bouteille de Klein, exemples sur lesquels on peut observer que nombre de variétés ne forment pas des espaces euclidiens. Et c'est justement parce qu'on n'est plus dans un espace euclidien que le gradient $\vec{\nabla} f$ en un point $\vec{\theta}$ n'est pas nécessairement la direction de plus grande pente. Une paramétrisation différente du même espace de politiques aurait donné une direction différente.

Dans une variété dotée d'une structure métrique riemanienne, la direction de plus grande pente est le *gradient naturel*, lequel est obtenu en corrigeant le gradient à l'aide d'une matrice représentant la « déformation locale » de l'espace en le point $\vec{\theta}$. Cette matrice est la *matrice d'information de Fischer*, et est donnée dans notre cas par :

$$F_s(\vec{\theta}) = E_{q(a|s;\vec{\theta})}\left[\frac{\partial \log q(a|s;\vec{\theta})}{\partial \theta_i}\frac{\partial \log q(a|s;\vec{\theta})}{\partial \theta_j}\right]. \tag{12.10}$$

La direction de plus grande pente est alors :

$$\widetilde{\nabla} f(\vec{\theta}) = F(\vec{\theta})^{-1}\vec{\nabla} f(\vec{\theta}). \tag{12.11}$$

En pratique — L'utilisation du gradient naturel ou d'une approximation de la fonction de valeur (et donc des deux ensembles) peut alourdir considérablement les calculs effectués pour chaque échantillon. Dans un problème pour lequel 1) le coût d'obtention de nouveaux échantillons est important ou 2) le temps de calcul reste négligeable par rapport au temps d'obtention des échantillons, ces méthodes plus complexes sont à préférer pour qu'une bonne solution soit trouvée avec peu d'échantillons. Si, par contre, on peut échantillonner très vite et à faible coût, par exemple quand un simulateur rapide du système est disponible, il est alors probable qu'un algorithme plus simple (tel que `OLpomdp`) s'avère plus rapide pour trouver une bonne solution (cf. chapitre 15).

Adaptation du pas d'apprentissage, critère d'arrêt

Toute amélioration qui peut être apportée à une méthode de gradient générique est candidate pour améliorer les méthodes de gradient pour MDP. On peut citer à ce titre le problème de l'adaptation du pas d'apprentissage. Le pas peut être adapté au fur et à mesure de l'apprentissage, par exemple en effectuant une recherche linéaire pour trouver un optimum local dans la direction du vecteur gradient. Le pas peut aussi être différent d'une dimension à l'autre (parce que l'on peut vouloir accélérer selon

une dimension et ralentir selon l'autre). Pour avoir un bon aperçu de ce sujet, voir [SCH 05b].

De manière générale, les questions qui se posent lors de l'utilisation d'un algorithme d'optimisation se posent ici aussi. Voici deux exemples que nous n'approfondirons pas ici :

– comment définir un critère d'arrêt, deux alternatives classiques ayant été décrites en section 12.1.2 ; et

– quel est le comportement « anytime » de l'algorithme, c'est-à-dire quelle est sa capacité à fournir une bonne solution « tôt » et à l'améliorer progressivement.

Utilisation d'une mémoire

On a vu que certains algorithmes permettent d'optimiser un contrôleur prenant des décisions en fonction de l'observation courante, ce qui s'étend naturellement à tout historique d'horizon fini. Dans le cas où un modèle du système est connu, on peut calculer l'état de croyance courant (la distribution de probabilité sur les états possibles) et prendre une décision en fonction de celui-ci, avec parfois la nécessité de travailler dans un espace d'états continu.

Une autre approche possible est d'ajouter une mémoire au contrôleur. Il peut s'agir d'une variable d'état interne, d'une boucle récurrente dans un réseau de neurones, ou d'un automate à états. L'usage de cette mémoire n'est pas prédéfini : c'est pendant l'apprentissage que cet usage prend forme (mémoriser un événement particulier...). Dans le cadre de méthodes de gradient, les deux travaux suivants emploient des automates à états finis en guise de mémoire :

– dans [MEU 99b], cet automate est un automate de contrôle, la même structure contient donc mémoire et contrôleur, cette structure (donc la quantité de mémoire disponible) étant fixée à l'avance ; et

– dans [ABE 02, ABE 03], un automate indépendant sert à contrôler une mémoire (à gérer un état interne), un second contrôleur utilisant l'état interne courant et l'observation courante pour décider de l'action à effectuer.

Si l'utilisation d'une telle mémoire peut être indispensable pour résoudre efficacement certains problèmes, ces approches restent peu courantes et d'usage difficile. En effet, non seulement ajouter une mémoire contribue à l'explosion de l'espace de recherche, mais en plus il n'y a au départ aucun indice sur le moyen d'utiliser cette mémoire (les informations utiles à retenir).

Guidage

Dans certains cas, un bon contrôleur est connu qui peut servir de guide lors de l'apprentissage. On peut alors distinguer au moins trois utilisations de ce guide :

– imitation : seules les décisions du guide sont observées ; on ne peut alors pas expérimenter et évaluer les autres décisions ; le seul apprentissage possible est une imitation simple du guide : un apprentissage supervisé ;

– imitation+exploration : les décisions du guide sont mélangées avec une politique exploratoire, de manière à expérimenter toutes les décisions possibles ; il faut alors que l'algorithme d'apprentissage tienne compte de ce que la politique apprise n'est pas celle utilisée lors de l'exploration, ce qui se fait à l'aide de méthodes d'*échantillonnage selon l'importance* [GLY 89, SHE 01, MEU 01, UCH 04] ;

– biais : la politique paramétrée peut « inclure » ce guide (qui sert donc de biais), l'optimisation ne faisant qu'apprendre à le corriger là où ses décisions ne sont pas les meilleures.

Contrôle multi-agents

Un système multi-agents (chapitre **??**) purement collaboratif (tous les agents partagent une fonction de récompense) peut être vu comme une entité prenant des décisions de manière répartie entre ses différents corps. On reste donc dans le cadre d'un problème d'optimisation classique dans lequel on cherche la meilleure politique pour les différents agents (que les agents aient une politique commune ou pas). Parce que le cadre multi-agent amène vite une explosion combinatoire, parce que chaque agent doit avoir son propre contrôleur et parce que les agents n'ont chacun qu'une observation partielle de leur environnement, il est naturel de faire une recherche directe de politique paramétrée en factorisant la politique du groupe en une politique par agent. Sur ce sujet, voir par exemple [DUT 01, PES 00].

12.5. Conclusion

Ce chapitre a présenté les méthodes de gradient pour la résolution de MDP. Cette approche est possible aussi bien avec ou sans modèle disponible, et permet de contrôler l'espace de recherche puisqu'une première étape est de choisir la forme de la politique paramétrée à optimiser. On peut réduire de manière comparable la taille de l'espace de recherche en calculant une fonction de valeur approchée, mais il n'y a pas de garantie de qualité d'une politique gourmande par rapport à cette fonction. Les recherches directes de politique ont alors l'avantage de travailler dans un espace de politiques stochastiques.

Un point important de ce chapitre est de rappeler le fait que la résolution de MDP est un problème d'optimisation qui peut être abordé par diverses techniques. La programmation dynamique est l'approche la plus courante, mais les méthodes de gradient, la programmation linéaire (chapitre 9, section **??**), les algorithmes évolutionnaires [SCH 94, MAR 07] ou les méthodes d'entropie croisée [SZI 06] peuvent être plus appropriés dans certaines situations.

Les lecteurs intéressés par la mise en pratique des méthodes de gradient pourront se référer par exemple : au chapitre 15 pour une utilisation avec grandeurs discrètes en planification, ou à [KIM 98, PET 03] pour des mises en œuvres avec grandeurs continues en robotique.

QUATRIÈME PARTIE

Exemples d'application des (PO)MDP (suite)

Chapitre 13

Recherche d'une zone d'atterrissage en environnement incertain par un hélicoptère autonome

Ce chapitre présente une utilisation des MDP sur support réel pour la recherche d'une zone d'atterrissage en environnement incertain par un hélicoptère autonome. Nous montrons quelles sont les contraintes théoriques et pratiques qui doivent être prises en compte afin d'embarquer un algorithme d'optimisation en-ligne et en temps contraint de MDP à bord de drones hélicoptères. L'application présentée dans ce chapitre est issue du projet RESSAC de l'ONERA[1] (voir [FAB 07] pour de plus amples informations sur ce projet). La figure 13.1 montre une photo prise lors d'une mission de recherche de zones d'atterrissage dans un environnement inconnu et artificiellement encombré par des cartons. La reconnaissance de zones d'atterrissage possibles et la stratégie d'action globale sont totalement autonomes : le rôle de l'opérateur humain visible sur la photo est uniquement de reprendre en main l'hélicoptère au cas où les logiciels embarqués venaient à défaillir durant les tests.

13.1. Introduction

Dans les systèmes autonomes robotiques, les processus de décision se doivent d'être « *anytime* », c'est-à-dire qu'ils doivent fournir une solution, éventuellement non optimale, à tout moment en temps borné. De nombreuses approches réactives de la planification déterministe [CHA 05, DAM 05] répondent à ce problème : un plan d'action est construit le plus rapidement possible sur un horizon borné dépendant du

---

Chapitre rédigé par Patrick FABIANI et Florent TEICHTEIL-KÖNIGSBUCH.

1. Office National d'Études et de Recherches Aérospatiales : *http ://www.onera.fr/*

Figure 13.1. *Mission de recherche de zones d'atterrissage par un hélicoptère autonome embarquant des algorithmes de décision basés sur les* MDP

temps disponible pour construire le plan et ce plan est raffiné si le temps est disponible, voire reconstruit au fur et à mesure de l'évolution de l'environnement. Cependant, ces approches ne sont pas optimales lorsque l'environnement est incertain (état partiellement observable, effets incertains des actions) et que ses incertitudes sont quantifiables, puisque la construction du plan suppose de manière optimiste que le système est dans la situation la plus probable.

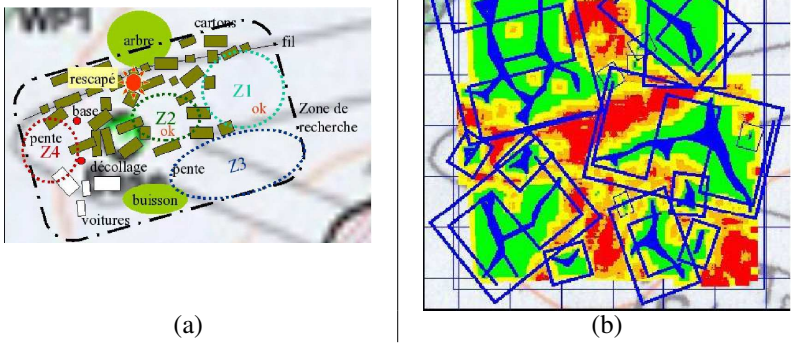
Au contraire, les approches de planification probabiliste comme les MDP construisent des plans conditionnels dépendant de l'état courant du système, et qui sont optimisés sur la moyenne de toutes les situations probables. Si ces approches ont le mérite d'être moins optimistes que les approches de planification déterministe, elles ont le désavantage d'être plus coûteuses en temps de calcul et en mémoire utilisée. De plus, l'optimisation d'un plan conditionnel sur l'ensemble des états possibles du système a orienté la recherche en planification probabiliste vers des algorithmes hors-ligne définis sur l'espace d'états tout entier. Il existe bien entendu des modèles permettant de structurer l'espace d'états (voir chapitre 9) et des algorithmes conçus pour guider la recherche d'une solution à l'aide d'heuristiques (voir chapitre 12 et [FEN 02, TEI 05a]). Cependant, ces approches modernes sont basées essentiellement sur des optimisations hors-ligne où l'implémentation sur système robotique réel n'est pas la préoccupation principale.

Dans ce chapitre, nous présentons une architecture de décision probabiliste, ainsi qu'un algorithme de planification probabiliste *anytime*, à mi-chemin entre la planification déterministe réactive et la planification probabiliste heuristique. L'architecture de décision utilise les threads du système d'exploitation embarqué afin de paralléliser la lecture de la politique et l'amélioration de la politique en tâche de fond. L'algorithme développe un sous-espace d'états atteignables à l'intérieur duquel une première politique non optimisée mais faisable est obtenue très rapidement, puis cette politique est optimisée et raffinée en tâche de fond dans un sous-espace d'états atteignables grandissant progressivement. Il s'agit ainsi d'une approche par programmation dynamique incrémentale dont la variable d'incrément est la taille du sous-espace d'états atteignables en suivant la politique courante.

Dans une première section, nous présentons le scénario de recherche de zone d'atterrissage dans un environnement mal connu et hostile. La deuxième section met l'accent sur l'architecture embarquée et, notamment, sur l'interaction entre le module décisionnel et les autres modules de l'architecture. Dans une troisième partie, nous présentons un cadre et un algorithme qui permettent d'optimiser en-ligne des MDP de manière incrémentale et locale. Enfin, avant de conclure ce chapitre, nous montrons l'intérêt d'une telle approche robotique par des résultats obtenus au cours de tests sur un drone hélicoptère autonome réel.

13.2. Présentation du scénario

Deux drones hélicoptères RMAX de marque Yamaha ont été équipés par l'ONERA d'une architecture de contrôle afin de remplir une mission autonome de recherche de zone d'atterrissage dans un environnement mal connu et hostile (cf. figure 13.1). Grâce aux capteurs embarqués et aux algorithmes développés pour traiter les données issues de ces capteurs, le vol, la navigation, l'exploration de zones, le décollage et l'atterrissage dans des zones inconnues ont été automatisés.



(a) (b)

Figure 13.2. *(a) Exemple de scénario – (b) Sous-zones extraites du traitement d'images (foncé=encombré, clair=libre)*

Le scénario qui a été expérimenté (cf. figure 13.2.a) consiste à secourir un rescapé dans une zone où on ne connaît pas a priori les sous-zones d'atterrissage possibles. En revanche, la position du rescapé et les dimensions de la zone sont connues. Une première exploration de la zone est effectuée à 50 mètres de hauteur, afin de filmer l'environnement et d'extraire par analyse de texture des sous-zones peu encombrées où il sera peut-être possible d'atterrir (cf. figure 13.2.b). Une exploration plus fine de ces sous-zones à 20 mètres permet de confirmer leurs « posabilités », c'est-à-dire le fait que le drone puisse s'y poser effectivement. Néanmoins, l'autonomie de vol est bornée, si bien qu'il n'est pas souhaitable d'explorer toutes les sous-zones dans n'importe quel ordre. Ainsi, à l'issue du traitement d'images, une planification de l'ordre d'exploration des sous-zones, de leur exploration elle-même, des actions de déplacement entre zones, d'atterrissage, de redécollage et de retour à la base est lancée.

13.2.1. *Problème de planification*

Le problème de planification commence lorsque le drone survole le rescapé à 20 mètres et que le superviseur a envoyé au planificateur une liste de sous-zones d'atterrissage possibles. Cette information est transmise sous la forme suivante (pour chaque zone : identifiant, coordonnées 2D, dimensions 2D, probabilité que la sous-zone soit atterrissable après exploration plus fine, nombre de points de passage de l'itinéraire d'exploration de la sous-zone) :

```
(zones (Z1 3153.65 -1348.34 30.9898 56.726 0.731584 6) ...)
```

À partir de cette liste de sous-zones, le planificateur génère un fichier représentant le problème de planification à résoudre. Le langage choisi est PPDDL (*Probabilistic Planning Domain Definition Language*, voir chapitre 15) [YOU 04b], où les états, les préconditions et les effets des actions sont décrits à l'aide de la logique du premier

ordre. L'intérêt principal de PPDDL réside dans la description « par intention » du problème : les formules booléennes sont paramétrées par des objets du domaine de planification. Dans le cas présent, les objets sont les sous-zones à explorer, ce qui permet de décrire une seule fois de manière générique les actions `goto(zone)`, `land` et `takeoff`. De plus, les opérateurs de la logique du premier ordre permettent de définir le domaine de manière intuitive et compacte et de le lire aisément.

13.2.2. *États et actions*

Les composantes d'état du problème de planification sont :

– `human-rescued` : booléen qui indique si le rescapé a été secouru ;

– `on-ground` : booléen qui indique si le drone est au sol ;

– `explored(zone)` : booléen (un par sous-zone) qui indique si une sous-zone a été explorée (à 20 mètres) ;

– `landable(zone)` : booléen (un par sous-zone) qui indique après exploration (à 20 mètres) si une sous-zone est atterrissable ;

– `where` : entier qui indique au-dessus de quelle zone se trouve le drone (y compris la verticale du rescapé) ;

– `flight-autonomy` : réel qui indique le temps de vol restant.

La taille de l'espace d'état est donc $2^{2(n+1)}(n+1)d$, où n est le nombre de sous-zones extraites de l'exploration globale de la zone et d est le nombre de points de discrétisation de la composante d'état représentant le temps de vol restant. Les algorithmes de planification qui ont été utilisés dans cette application (cf. section 13.4) nécessitent des variables d'environnement discrètes. Toutefois, des approches de planification relativement récentes permettent de raisonner directement avec des états hybrides, comme dans [GUE 04] par exemple, et auraient pu être implémentées.

Nous considérons $n + 6$ actions :

– `goto(zone)` : déplacement vers la sous-zone `zone` ;

– `explore` : exploration de la sous-zone survolée ;

– `land` : atterrissage dans la sous-zone (`zone`) survolée, si `landable(zone)` est vrai ;

– `takeoff` : décollage depuis la sous-zone où le drone est posé ;

– `fail-safe` : retour de sécurité à la base lorsque le temps de vol restant est inférieur à 10 minutes ;

– `end-mission` : fin de la mission, lorsque le drone est revenu à la base, soit après avoir secouru le rescapé, soit après un retour de sécurité.

13.2.3. *Incertitudes*

Parmi les incertitudes de l'environnement, trois types d'incertitudes ont été retenues, qui influencent sensiblement l'optimisation de la stratégie :

– probabilité P_a qu'une zone soit posable après exploration locale (cf. figure 13.2.b) :

$$P_a = \frac{\text{nombre de pixels clairs}}{\text{nombre de pixels foncés}} \qquad \textit{[pixels issus d'une analyse de texture]} \ ;$$

– probabilité P_s de secourir le rescapé si le drone atterrit à une distance d_z du rescapé dans la sous-zone z :

$$P_s = \frac{40}{40 + d_z} \ ;$$

– densité de probabilité f_τ qu'une action dure τ secondes (μ_a et σ_a dépendent de l'action a) :

$$f_\tau = \frac{1}{\sigma_a \sqrt{2\pi}} e^{-\frac{(\tau - \mu_a)^2}{2\sigma^2}} \ .$$

13.2.4. *Critère à optimiser*

Le planificateur doit optimiser un critère additif, égal à la somme des récompenses ou des pénalités reçues après chaque action. Une récompense de $+1000$ est associée au sauvetage du rescapé (reçue lorsque le rescapé est secouru et que l'hélicoptère est posé). Une pénalité de -1000 correspond au retour à la base sans avoir secouru le rescapé (reçue lorsque l'hélicoptère revient à la base sans que le rescapé soit secouru). La stratégie produite est ainsi censée indiquer un ordre d'exploration locale des sous-zones qui réalise un compromis entre trois types d'événements : la possibilité qu'une sous-zone soit qualifiée d'atterrissable après exploration, la possibilité de secourir le rescapé si le drone atterrit dans cette sous-zone et la possibilité de rentrer à la base (temps de vol restant) après avoir survolé cette sous-zone.

13.2.5. *Modèle formel de décision*

La spécification du problème de planification présentée ci-avant se prête tout à fait à une modélisation sous forme de FMDP (cf. chapitre 9) :

– le modèle de transition est stochastique markovien ;

– les récompenses sont rattachées aux effets des actions ;

– le critère d'optimisation est additif ;

– l'espace d'états est factorisé par variables.

De plus, le problème spécifié en langage PPDDL est automatiquement traduit en DBN, comme indiqué dans [YOU 04b]. Les DBN obtenus sont ensuite codés sous forme d'ADD pour une meilleure efficacité du traitement des données (cf. sous-section

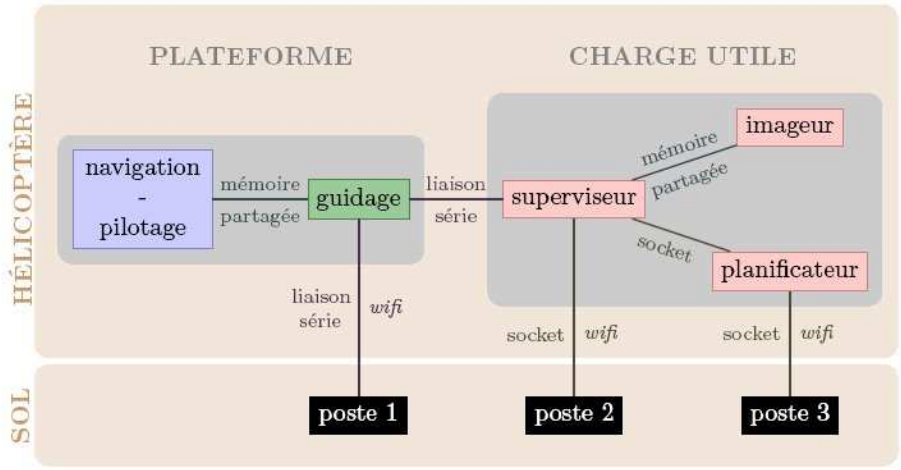


Figure 13.3. *Vue globale et synthétique de l'architecture embarquée*

9.3.2). Néanmoins, les algorithmes d'optimisation des FMDP basés sur les diagrammes de décision — y compris les plus récents — ne sont pas réellement conçus pour une application « *anytime* » dans des conditions robotiques réelles. Ainsi, nous montrons dans la suite de ce chapitre un cadre algorithmique permettant une utilisation « *anytime* » d'algorithmes d'optimisation locaux et heuristiques des FMDP.

13.3. Architecture de décision embarquée

13.3.1. *Vue globale*

Comme le montre la figure 13.3, l'architecture embarquée sur les drones est divisée en deux parties : la couche réactive pour des traitements quasiment immédiats et la couche délibérative pour des traitements plus longs qui ne sont pas nécessairement bornés dans le temps. Chaque couche logicielle est affectée à un processeur dédié. Les deux couches interagissent essentiellement par échange de données binaires.

Les fonctions de contrôle du vol sont exécutées dans la couche réactive sous des contraintes temps-réel strictes : elles ont été validées séparément et doivent pouvoir être exécutées indépendamment de la couche délibérative afin d'assurer la sécurité permanente des vols. Les fonctions de la couche délibérative sont alors autorisées à consommer plus de ressources de mémoire et de temps de calcul, sans pénaliser les fonctions temps-réel qui sont vitales pour maintenir l'hélicoptère en vol. En d'autres termes, la différence principale entre les deux couches –et la raison pour laquelle il est préférable de les séparer clairement– est le fait qu'elles n'ont pas du tout les mêmes contraintes temporelles : une telle séparation est vivement recommandée afin d'implémenter des algorithmes embarqués efficaces pour la décision autonome.

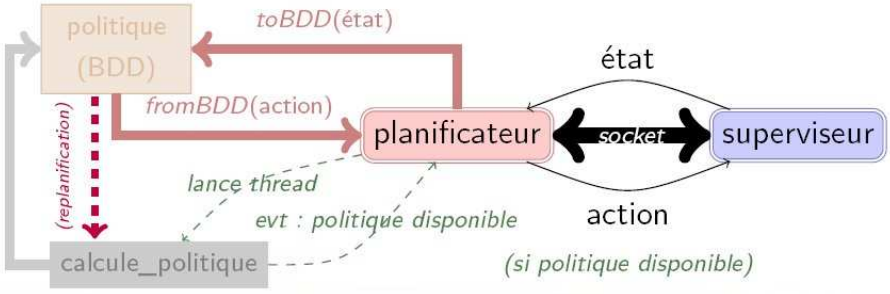


Figure 13.4. *Architecture de décision : la planification est un service multi-tâche sollicité par le superviseur*

La couche délibérative est composée de trois modules principaux permettant de réaliser le cycle classique *perception - décision - action* :

– *imageur* : réalise le traitement des images enregistrés par les capteurs ;

– *planificateur* : optimise le FMDP dont le modèle est issu du traitement d'images ;

– *superviseur* : machine à états qui déroule le scénario (préprogrammé), coordonne l'imageur et le planificateur et envoie les actions de bas niveau à la couche réactive.

L'imageur et le planificateur sont des serveurs qui sont activés à la demande par le superviseur. Ceci permet de lancer plusieurs processus de traitement d'images ou de planification en parallèle. Par exemple, un deuxième sous-problème de planification pourrait être lancé en complément du premier problème de planification (mentionné plus haut) afin d'optimiser plus finement les actions de l'hélicoptère à l'intérieur de chaque zone candidate à l'atterrissage.

13.3.2. *Planification multi-tâche sur requête du superviseur*

La figure 13.4 représente l'interaction entre le planificateur et le superviseur. Dès que le superviseur reçoit de l'imageur le résultat du traitement d'images de la zone globale (extraction de zones d'atterrissage candidates), il se connecte au serveur de planification (le *planificateur*) et lui envoie le problème à résoudre. Le planificateur lance alors deux tâches en parallèle : une tâche d'optimisation de la politique et une tâche de dialogue avec le superviseur.

13.3.2.1. *Optimisation de la politique*

La politique est générée de manière incrémentale à l'aide d'un algorithme que nous présentons dans la section suivante. À chaque incrément de l'algorithme, la politique est localement améliorée : sa valeur augmente à l'intérieur d'un sous-espace d'états qui englobe un minimum d'états atteignables, dont l'état courant et les états buts. Entre deux incréments, la politique et l'espace d'états atteignables sont copiés dans une

place mémoire sûre, c'est-à-dire protégée en lecture et écriture par mutex. Ceci permet de modifier la politique tout en ayant constamment accès à une politique applicable.

De cette façon, le planificateur fournit une politique dès la fin du premier incrément, ce qui permet d'envoyer au superviseur une première action à réaliser, sans attendre que l'algorithme d'optimisation termine complètement. Le planificateur peut ainsi être qualifié d'« *anytime* » à condition, comme nous le verrons plus loin, que le temps de calcul du premier incrément soit faible.

13.3.2.2. *Dialogue avec le superviseur*

Le dialogue avec le superviseur n'est pas aussi simple que le lecteur pourrait le croire, car les données sont échangées dans le socket en ASCII alors qu'elles sont codées sous forme de BDD et d'ADD dans le planificateur. Il est en effet inutile d'envoyer des diagrammes de décision binaires dans le socket, puisque le superviseur ne dispose pas de la connaissance suffisante pour déduire la sémantique de ces diagrammes de décision [YOU 04b].

Aussi, le processus de dialogue avec le superviseur nécessite un calcul de conversion des diagrammes de décision vers des actions représentées en ASCII (par exemple `(goto (Z0))`). Inversement, les états codés en ASCII doivent être convertis en diagrammes de décision.

Enfin, la lecture d'une action optimale dans l'état courant nécessite de bloquer le mutex associé à la lecture ou l'écriture de la politique courante. Si l'état courant n'est pas dans le sous-espace d'états atteignables, ou si ses effets en sont en-dehors, la tâche d'optimisation de la politique est interrompue puis relancée depuis le nouvel état initial. Les cas de replanification sont argumentés dans la section suivante.

13.4. Programmation dynamique stochastique, incrémentale et locale

La décision embarquée sur des hélicoptères autonomes doit être assez réactive, c'est-à-dire qu'elle doit produire des décisions en un temps comparable à la réalisation d'une action de haut niveau. Si tel n'était pas le cas, les décisions seraient déphasées avec l'environnement, au point qu'elles risqueraient de conduire l'hélicoptère dans des situations dangereuses. De plus, l'autonomie de vol est limitée, si bien qu'il n'est pas envisageable de consommer tout le temps disponible pour produire une décision.

Par conséquent, une application robotique réelle — qui plus est critique — des MDP nécessite une adaptation des algorithmes d'optimisation afin de produire des politiques en ligne « *anytime* ». Dans le contexte de l'application présentée dans ce chapitre, « *anytime* » signifie qu'une action automatiquement calculée par le planificateur est disponible en un temps comparable à la réalisation d'une action de haut niveau. Ce n'est pas, en général, le cas des algorithmes d'optimisation des MDP, y compris sous-optimaux ou heuristiques.

Le cadre algorithmique choisi dans notre application repose sur l'optimisation incrémentale d'une politique locale, connaissant l'état courant du système et une condition but à satisfaire :

$$at(base) \wedge (human_rescued \vee (flight_autonomy \leqslant 10\ mn))$$

Un sous-espace d'états atteignables est calculé en appliquant la politique courante depuis l'état courant jusqu'aux états qui satisfont la condition but. Une fois ce sous-espace d'états calculé, la politique est localement ré-optimisée à l'intérieur de ce sous-espace d'états et ainsi de suite. L'algorithme alterne donc deux phases : une phase de génération des états atteignables sur la base de la politique courante et une phase d'optimisation de la politique à l'intérieur du sous-espace d'états atteignables. La variable d'incrément de l'algorithme est la taille du sous-espace d'états atteignables. Entre chaque incrément, la politique locale obtenue peut-être appliquée par le superviseur sans attendre qu'elle soit totalement optimisée.

Dans cette application, l'algorithme incrémental choisi est `sfDP` *(Stochastic Focused Dynamic Programming)*, car il prend en compte à la fois un état initial et des états buts dans l'optimisation du MDP (voir [TEI 05a, TEI 05b]). D'autres algorithmes incrémentaux auraient pu être utilisés, comme `sLAO*` [FEN 02], ou `sRTDP` [FEN 03].

13.4.1. *Obtention d'une première politique non optimisée*

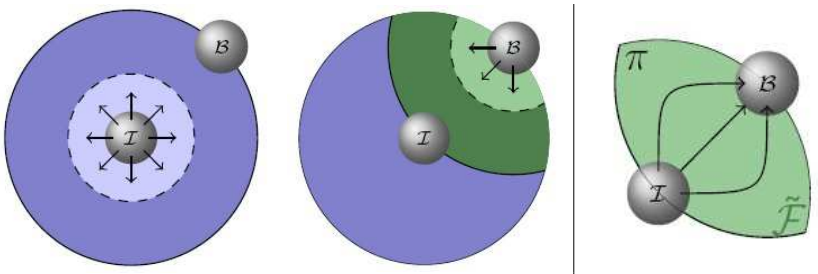
Afin d'obtenir rapidement une première politique applicable, l'algorithme `sfDP` calcule un plus court chemin « logique » menant de l'état initial courant aux états buts et ceci sans aucune optimisation numérique. Le système récursif suivant permet de calculer itérativement la politique initiale à l'aide de la logique du premier ordre :

$$\pi_0(e) = \begin{cases} \{\text{actions définies dans les états buts}\} & \text{si } e \in \text{buts}, \\ \emptyset & \text{sinon} \end{cases} ; \tag{13.1}$$

$$\pi_{i+1}(e) = \begin{cases} \{a : \exists\ e'\ ,\ T(e' \mid a, e) > 0 \text{ et } \pi_i(e') \neq \emptyset\} & \text{si } \pi_i(e) = \emptyset, \\ \pi_i(e) & \text{sinon} \end{cases} . \tag{13.2}$$

La deuxième ligne de l'équation (13.2) vérifie la propriété de plus court chemin car elle empêche, dans un état donné, de remettre en question une action qui a été déjà calculée. Comme cette action a été obtenue dans une itération précédente, elle correspond à un nombre d'étapes moindre pour atteindre un état but. L'itération s'arrête lorsque la politique est définie dans l'état initial e_0, c'est-à-dire : $\exists\ i > 0, \pi_i(e_0) \neq \emptyset$.

Remarquons enfin que l'exécution de cette politique initiale est stochastique puisque, pour tout état e et toute étape i, $\pi_i(e)$ est un *ensemble d'actions* menant à un état but en i étapes depuis e.



en avant
$\mathcal{F}_{t+1} = \mathcal{T}_{det}(\mathcal{F}_t, \pi)$

en arrière
$\tilde{\mathcal{F}}_{t-1} = \mathcal{T}_{det}^{-1}(\tilde{\mathcal{F}}_t, \pi) \cap \mathcal{F}_{t_\mathcal{B}}$

(a)

programmation dynamique dans $\tilde{\mathcal{F}}$

(b)

Figure 13.5. *SFDP : (a) expansion du sous-espace d'états atteignables $\mathcal{F}$ en suivant la politique courante π des états initiaux $\mathcal{I}$ aux états buts $\mathcal{B}$, puis (b) optimisation de π. ($\mathcal{T}$: transitions)*

13.4.2. *Génération du sous-espace d'états atteignables*

Connaissant la politique courante π, le sous-espace des états atteignables $\mathcal{F}$ est calculé en deux passes, une avant et une arrière (cf. figure 13.5). Le système récursif suivant, encodé sous forme de BDD , propage les états atteignables en avant depuis les états initiaux $\mathcal{I}$ jusqu'à ce qu'au moins un état atteignable satisfasse la condition de but $\mathcal{B}$:

$$\mathcal{F}_0 = \mathcal{I}, \tag{13.3}$$

$$\mathcal{F}_{i+1} = \mathcal{F}_i \cup \{e' : T(e' \mid \pi(e), e) > 0, e \in \mathcal{F}_i\}\,. \tag{13.4}$$

Ensuite, le système récursif suivant propage les états atteignables en arrière depuis les états buts $\mathcal{B}$ jusqu'à ce qu'au moins un état atteignable soit dans l'ensemble $\mathcal{I}$, en ne gardant que les états qui sont dans l'ensemble $\mathcal{F}$ précédemment calculé :

$$\tilde{\mathcal{F}}_0 = \mathcal{B} \cap \mathcal{F}, \tag{13.5}$$

$$\tilde{\mathcal{F}}_{i+1} = \tilde{\mathcal{F}}_i \cup \left(\left\{e : T(e' \mid \pi(e), e) > 0, e' \in \tilde{\mathcal{F}}_i\right\} \cap \mathcal{F}\right). \tag{13.6}$$

Le sous-espace d'états atteignables restreint $\tilde{\mathcal{F}}$ est finalement mis dans $\mathcal{F}$: $\mathcal{F} \leftarrow \tilde{\mathcal{F}}$.

13.4.3. *Optimisation locale de la politique*

Après chaque génération du sous-espace d'états atteignables $\mathcal{F}$, la politique courante π est mise à jour à l'aide de l'équation de Bellman classique, mais calculée

uniquement dans $\mathcal{F}$:

$$V_0(e) = 0, \tag{13.7}$$

$$V_{i+1}(e) = \mathbf{1}_{\mathcal{F}}(e) \cdot \max_a \left\{ \sum_{e'} T(e' \mid a, e) \cdot (\gamma V_i(e') + R(e' \mid a, e)) \right\}. \tag{13.8}$$

La nouvelle politique locale est alors obtenue en appliquant une fois de plus l'opérateur de Bellman, en n'évaluant que les états qui sont dans $\mathcal{F}$:

$$\pi(e) = \mathbf{1}_{\mathcal{F}}(e) \cdot \operatorname*{argmax}_a \left\{ \sum_{e'} T(e' \mid a, e) \cdot (\gamma V^*(e') + R(e' \mid a, e)) \right\}. \tag{13.9}$$

Cette nouvelle politique locale est immédiatement copiée dans un espace mémoire sûr protégé par mutex. Elle est immédiatement applicable par le superviseur, tandis que le planificateur génère de nouveau le sous-espace d'états atteignables en suivant cette nouvelle politique.

13.4.4. *Replanifications locales*

L'avantage de `sfDP` sur d'autres méthodes incrémentales est la connaissance d'états buts, ce qui réduit le nombre d'états à explorer durant la phase d'optimisation locale. Cependant, son désavantage est la perte de garantie d'optimalité car tous les états atteignables depuis les états initiaux ne sont pas explorés, mais uniquement ceux qui mènent aux états buts. En particulier, il n'y a aucune garantie que, en partant d'un état initial, chaque trajectoire possible de la politique locale courante mène aux états buts : la génération du sous-espace d'états atteignables s'arrête en effet dès qu'*au moins une* telle trajectoire est trouvée (et non toutes). Ainsi, le planificateur doit lancer un processus de replanification dans les deux cas suivants (cf. figure 13.6) :

(a) une transition de faible probabilité survient durant l'exécution de la politique, si bien que le nouvel état courant est à l'extérieur du sous-espace d'états atteignables ;

(b) aucune transition n'est définie dans l'état courant, lorsque tous les effets stochastiques de la politique courante dans cet état sont à l'extérieur du sous-espace d'états atteignables.

Le temps de replanification décroît généralement à chaque replanification, car les nouveaux états initiaux de chaque processus de replanification sont sans cesse plus proches des états buts.

13.5. Tests en vol et retour d'expérience

Dans cette section, nous présentons des résultats obtenus sur support réel (hélicoptères RMAX) sur le terrain d'aviation d'Esperce dans le Sud-Ouest de la France.

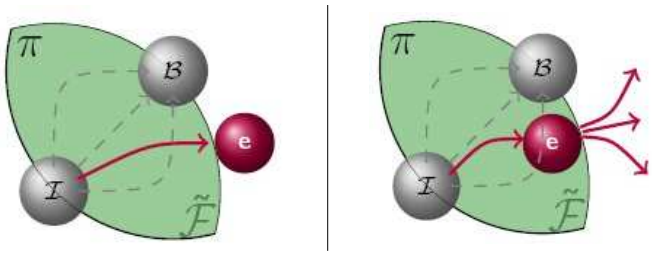


Figure 13.6. *Cas de replanification : (a) l'état courant est à l'extérieur du sous-espace atteignable ; (b) la politique n'est pas définie dans l'état courant*

Nous comparons les performances entre l'algorithme de planification « *anytime* » multi-tâche présenté dans ce chapitre et des approches mono-tâche. L'algorithme d'optimisation incrémentale est `sfDP`, tel que présenté dans la section précédente. Nous nous focalisons sur quatre critères de comparaison :

– *temps total d'optimisation* : somme des temps d'optimisation des processus de planification et de replanification ;

– *nombre de replanifications* ;

– *temps de réponse maximum* : temps maximum nécessaire pour renvoyer une action au superviseur après réception de l'état courant ;

– *zone d'atterrissage* : obtenue en appliquant la politique courante depuis la base où est stationné l'hélicoptère.

Dans le but de comparer ces critères sur une base commune, nous avons enregistré sur disque une liste de 10 zones qui ont été extraites du traitement d'images durant un vol d'essai. Afin de coder le problème de planification sous forme de BDD et d'ADD , la variable continue `flight-autonomy` a été divisée en 288 intervalles de 12,5 secondes chacun. La taille de l'espace d'états est donc : $2 \times 2 \times 2^{10} \times 2^{10} \times 11 \times 288 = 13\,287\,555\,072$ états.

Le tableau 13.1 représente la comparaison entre la version mono-tâche de `sfDP` et sa version multi-tâche. Toutes les actions ont été simulées en laboratoire après le vol d'essai, y compris leurs durées, afin de mettre en évidence le processus d'optimisation en tâche de fond (en mode multi-tâche) et les processus de replanification. Les actions durent en moyenne 50 secondes. Tous les tests ont été effectués sur le processeur embarqué à bord du drone et dédié aux processus délibératifs (Pentium 1 GHz). La dernière colonne reproduit les résultats obtenus durant le vol réel. La deuxième colonne correspond à un algorithme d'optimisation optimal comme `SPUDD`, qui travaille la politique sur l'espace d'états tout entier. Cet algorithme optimal donne une idée de la complexité du problème de planification et il permet de comparer la zone d'atterrissage optimale et celles obtenues par les différentes versions de `sfDP`. Notons que

nb de zones	5	5	5	7	7	10	10
algorithme	optimal	ST	MT	ST	MT	ST	MT
temps total optimisation	1358	2,58	2,8	13,76	13,6	308,29	258,57
nb replanifications	0	0	1	0	1	3	4
temps réponse max	1358	2,58	**0,21**	13,76	**0,29**	308,22	**5,75**
zone atterrissage	Z1	Z1	Z0	Z5	Z0	Z0	Z1

Tableau 13.1. *Comparaison entre sfDP single-thread (ST) et sfDP multi-thread (MT) – le temps est donné en secondes*

l'algorithme optimal est incapable, dès 5 zones, de calculer une politique en moins d'une heure, qui est la durée maximale d'une mission !

Ces tests n'évaluent pas l'efficacité d'algorithmes de planification stochastiques heuristiques et locaux tels `sLAO*` ou `sfDP`. Le lecteur intéressé pourra trouver une discussion sur l'intérêt de tels algorithmes dans [TEI 05b, FEN 03]. Nous souhaitons plutôt montrer dans ce chapitre qu'il est possible de modifier légèrement un algorithme développé en laboratoire sur des problèmes académiques, afin de l'adapter à des conditions réelles d'utilisation robotique. Dans le cas présent, une approche « *anytime* » multi-tâche produit des politiques applicables et non stupides en un temps très court, ce qui est une contrainte opérationnelle des algorithmes de décision embarqués à bord d'hélicoptères autonomes. Le tableau 13.1 montre que le temps de réponse maximum de la version multi-tâche de `sfDP` est négligeable par rapport à la durée moyenne des actions ($\sim$ 50 secondes). Le temps de réponse maximum de la version mono-tâche est significativement plus grand. Avec 10 zones, en mode mono-tâche, l'état courant n'est quasiment jamais inclus dans le sous-espace d'états atteignables lors des replanifications successives, car la variable d'état `flight-autonomy` décroît autant que le temps de replanification, lui-même supérieur au temps autorisé pour appliquer la politique réoptimisée dans l'état courant. Ainsi, de nombreuses replanifications sont nécessaires pour appliquer une seule action. Ce phénomène ne survient pas en mode multi-tâche, bien que cela ne soit pas visible dans le tableau présenté : dans ce mode, toutes les replanifications ont eu lieu individuellement après la réalisation d'actions différentes.

13.6. Conclusion

Dans ce chapitre, nous avons présenté une architecture de décision multi-tâche pour la planification dans l'incertain en temps réel. Nous avons montré que les algorithmes d'optimisation des MDP, y compris heuristiques, ne peuvent pas être utilisés tels quels dans des applications réalistes qui nécessitent des calculs en ligne contraints par le temps. En effet, le temps de calcul de ces algorithmes excède le temps limite entre le planificateur et les senseurs de l'agent autonome. En revanche, nous avons montré qu'il est possible de les adapter à un cadre « *anytime* » permettant de résoudre en temps réel des problèmes de planification stochastique.

Le processus de planification est divisé en deux threads communicants, synchronisés sur la politique courante applicable. Une première politique stochastique est rapidement obtenue en calculant au moins une trajectoire qui mène de l'état initial vers les états buts. Les tests sur support réel justifient les modifications apportées aux algorithmes théoriques : non seulement le temps de réponse du planificateur est significativement réduit, mais des politiques optimisées peuvent dès lors être appliquées par le superviseur en un temps compatible avec la durée des actions.

Une autre façon de produire des stratégies « *anytime* » pourrait consister à découper un algorithme d'optimisation incrémentale en de petits processus de calcul *indépendants*, lancés successivement à l'intérieur d'une seule tâche (voir [VER 07] pour un exemple d'architecture de décision réelle basée sur ce principe). Chaque nouveau processus de calcul améliorerait d'un incrément la politique locale renvoyée par le processus de calcul précédent. Néanmoins, une telle implémentation serait sans doute pénible, car elle nécessiterait de programmer des protocoles d'échanges de données entre les différents processus de calcul, afin de transmettre l'information nécessaire à la construction de la politique courante entre deux processus de calcul successifs.

Chapitre 14

Contrôle d'une mission d'exploration

Ce chapitre présente une application robotique réelle pour laquelle le contrôle se base sur les processus décisionnels de Markov. Un robot explorateur autonome est chargé de récolter des informations sur différents sites inaccessibles avant de renvoyer le résultat de ses analyses. Dans ce contexte, les ressources de ce système embarqué sont limitées. Les tâches que le robot effectuera sont constituées de plusieurs étapes. Le système de contrôle haut niveau qui est présenté dans ce chapitre permet de sélectionner quelles tâches et quelles étapes devront être effectuées en tenant compte à chaque instant des ressources disponibles (énergétiques par exemple) et de l'importance des différentes tâches de la mission.

14.1. La mission du robot explorateur

Un robot explorateur autonome a pour mission de récolter des informations sur un terrain où l'être humain ne peut pas s'aventurer. Cela peut être sur Mars, à une grande profondeur sous l'eau ou encore sous les décombres après un tremblement de terre par exemple. Ce robot dispose d'une palette d'outils embarqués qui lui permettent de faire des relevés qui pourront ensuite être analysés par des scientifiques. De plus ce robot ne dispose pas forcément d'une quantité suffisante de ressources pour effectuer toutes les opérations de la mission.

La mission du robot est découpée en plusieurs tâches indépendantes. Le robot dispose d'un plan d'exploration avant de commencer la mission, comme celui de la figure 14.1. Le robot devra alors visiter certains sites pour y effectuer des prélèvements (atmosphériques, de minerai au sol, prise de photographie, etc...).

Chapitre rédigé par Simon LE GLOANNEC et Abdel-Illah MOUADDIB.

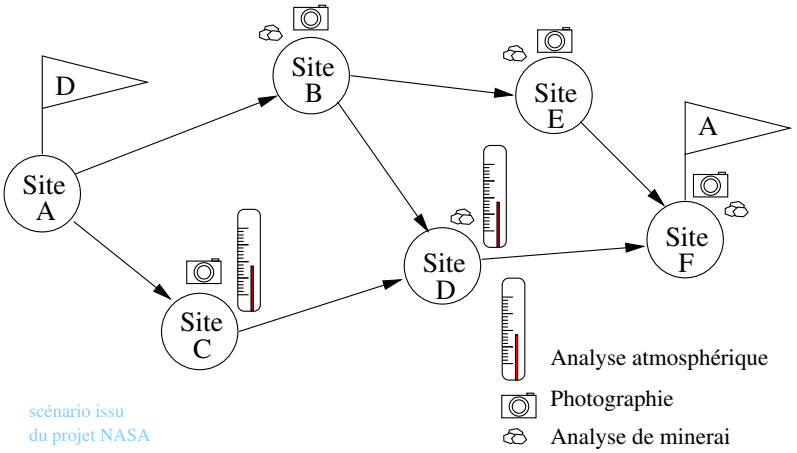


Figure 14.1. *Le robot a une mission : explorer certains sites (nœuds) de ce graphe acyclique, dans un ordre imposé. Il partira du site de départ marqué d'un **D** et arrivera, si le temps le lui permet, au site d'arrivée marqué d'un **A**. Il passera plus ou moins de temps sur chaque site pour récolter des informations. Le robot va adapter le temps passé sur chaque site en fonction de la description qu'il aura préalablement reçu pour celui-ci. Le temps disponible pour effectuer chaque mission est limité.*

Chaque opération peut prendre plus ou moins de temps, consommer plus ou moins de ressources. En effet, puisque le robot possède une palette d'outils importante, il existe plusieurs façons d'accomplir chaque tâche. Certains outils possèdent même plusieurs modes de fonctionnement : haute qualité, qualité moyenne et basse qualité. Comme la quantité de ressource embarquée est limitée, il faut que le robot puisse adapter sa consommation de façon à ce qu'il lui reste suffisamment de ressources pour effectuer les tâches les plus pertinentes. Le contrôle de la consommation des ressources consommables est le premier aspect original de ce chapitre. Nous présentons ici non pas une méthode pour planifier ou ordonner les différentes tâches, mais un système de contrôle haut niveau qui permet de gérer le mieux possible la consommation de ressources pour la mission. [BRE 02, **?**] proposent une approche pour planifier une mission en tenant compte de la consommation de ressources.

La structure des tâches qui constituent la mission est le deuxième point original de ce chapitre. Une tâche de récolte d'informations est une suite d'actions élémentaires qui sont cette fois-ci dépendantes les unes des autres : elles doivent être effectuées dans un ordre bien précis. La figure 14.2 illustre une tâche correspondant à une photographie. La construction du modèle des tâches est préalablement nécessaire pour écrire une mission.

Sur un site donné, le robot peut effectuer une ou plusieurs tâches. Ces tâches sont hiérarchiques, comme le montre la figure 14.2. Le robot va pouvoir l'exécuter étape

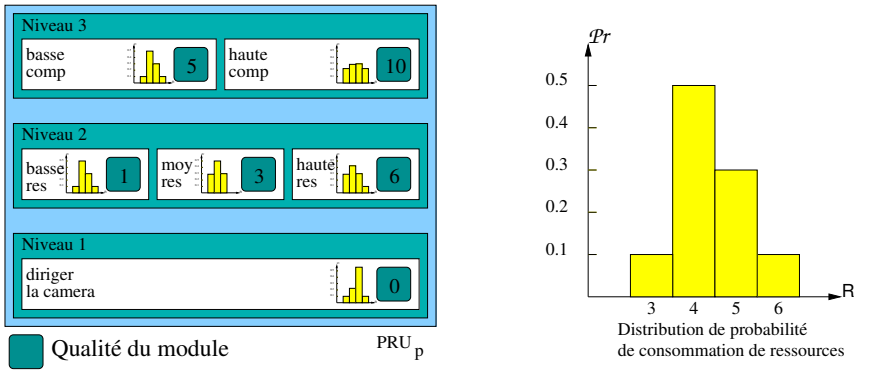


Figure 14.2. *Le robot explorateur récolte des informations un site : la partie gauche de la figure représente la modélisation d'une tâche de récolte d'information. Sur un site donné, le robot va devoir prendre une photo puis l'enregistrer. Cela se fait en trois étapes : diriger la caméra, choisir la définition de l'image, puis enregistrer cette image sur le disque. A chaque étape ou niveau, le robot peut choisir une et une seule option parmi les modules qui lui sont proposés.*

par étape. Après chaque étape, il pourra décider de l'interrompre pour passer à une autre tâche sur un autre site. Cette interruption peut faire économiser des ressources au robot pour plus tard, mais le robot ne pourra pas revenir sur cette tâche par la suite. A chaque étape, le robot dispose de plusieurs choix (comme basse résolution ou haute résolution sur la figure 14.2). Un choix implique une consommation plus ou moins forte de ressources, mais aussi un résultat plus ou moins bon pour la tâche globale. D'autres travaux sont consacrés à la décomposition hiérarchique des tâches [**?**, **?**, **?**].

Le système de contrôle présenté dans ce chapitre permet justement de trouver le meilleur compromis entre les ressources utilisées et la qualité globale des tâches effectuées. Comme les tâches de récolte d'informations sont réalisables progressivement, on parle de raisonnement progressif. Une tâche sera appelée unité de raisonnement progressif (PRU). Le raisonnement progressif a été utilisé pour modéliser des missions d'exploration dans [CAR 01, ZIL 02]. La première partie de ce chapitre constitue une introduction au formalisme. Nous présenterons ensuite comment contrôler la consommation de ressources avec un processus décisionnel de Markov en exploitant la structure hiérarchique propre au raisonnement progressif. Nous finirons par illustrer cette approche par une implémentation sur un robot autonome.

14.2. Formalisme d'une mission constituée de tâches progressives

Une **mission** est ici une suite finie de tâches représentées sous forme d'unités de raisonnement progressif (notées PRU). Nous attribuons un indice unique $\mathtt{p}$ à chaque $pru_{\mathtt{p}}$, $\mathtt{p} \in [1, \ldots, \mathbf{P}]$ (voir figure 14.1).

NOTE.– Nous pourrions étendre la définition de mission à un graphe acyclique de PRU. Une suite de PRU est un chemin possible dans un graphe acyclique de PRU (par exemple les PRU A, B, E, F de la figure 14.1). Nous simplifions le problème pour des raisons de clarté, car les indices utilisés pour décrire le modèle sont nombreux.

Dans la mission d'exploration, la réalisation d'une tâche suit l'exécution d'une PRU. Cette dernière est composée d'une suite finie de $\mathtt{N}$ **niveaux** . À la fin d'un niveau l'agent peut décider d'exécuter ou non le niveau suivant. Par contre, le robot ne peut pas sauter directement de niveau : il ne peut pas enregistrer la photo sans l'avoir prise.

Quand le robot décide d'exécuter un niveau, plusieurs méthodes s'offrent à lui. Il peut dans cet exemple prendre une photo de bonne ou de mauvaise qualité. Chaque niveau est donc représenté par un ou plusieurs **modules** qui sont des façons de l'exécuter. L'agent ne peut exécuter qu'un seul module par niveau. À chaque module est associée une qualité $\mathbf{Q}$ et une distribution de probabilité sur la consommation de ressource possible Pr. Nous notons $\mathsf{m}_{\mathtt{p},\mathtt{n},\mathtt{m}}$ le module $\mathtt{m}$ du niveau $\mathsf{N}_{\mathtt{p},\mathtt{n}}$ et $\mathbf{Q}_{\mathtt{p},\mathtt{n},\mathtt{m}}$ sa qualité.

Cette **qualité Q** est un réel positif qui représente numériquement la qualité de l'exécution du module. La qualité d'exécution d'une PRU est égale à la somme de toutes les qualités des modules exécutés dans celle-ci. Cette qualité totale n'est gagnée par le robot (par le biais d'une fonction de récompense) que si celui-ci exécute la PRU en entier. Une exécution inachevée de PRU ne rapporte rien.

La **distribution de probabilité** sur la consommation de ressource $Pr_{\mathtt{p},\mathtt{n},\mathtt{m}}$ de chaque module représente l'incertitude qui existe dans notre modèle. C'est une fonction $Pr : \mathbb{R} \rightarrow [0, 1]$ qui donne la probabilité de consommer une quantité de ressources quand l'agent exécute le module $\mathsf{m}_{\mathtt{p},\mathtt{n},\mathtt{m}}$ ($\mathbb{R}$ représente l'espace des ressources consommables).

Les **ressources consommables** sont des quantités mesurables et décroissantes. Les quantités de ces ressources consommables diminuent au fur et à mesure que la mission se poursuit, après chaque module, sans jamais augmenter pendant la mission. Parmi celles-ci on peut citer par exemple : le **temps** restant pour effectuer la mission, l'**énergie** dans les batteries ou l'**espace mémoire** restant sur le disque de données.

NOTE.– Dans ce chapitre, nous nous limitons à une seule ressource consommable, l'énergie par exemple. La gestion de plusieurs ressources consommables est possible. Ce problème a été traité dans [**?**].

Nous avons maintenant entre les mains toutes les informations nécessaires à la construction de la politique qui permettra au robot d'exécuter sa mission. Nous allons donc présenter le mécanisme de contrôle du raisonnement progressif.

14.3. Modélisation MDP / PRU

Le mécanisme de contrôle de la mission se fait en plusieurs étapes :

– la formalisation d'un processus décisionnel de Markov à partir de la mission ;

– le calcul d'une politique globale à partir du processus décisionnel de Markov obtenu ;

– l'exécution de la mission en suivant cette politique.

Le mécanisme de contrôle global prévoit un contrôle total de la mission, c'est-à-dire que la politique globale est calculée avant d'exécuter cette mission.

14.3.1. *Les états de l'agent*

L'état de l'agent dépend entre autres de la PRU qu'il exécute et du niveau qu'il a atteint dans celle-ci. On accumule dans l'état la qualité de tous les modules qui ont été exécutés dans cette PRU, c'est pourquoi la qualité accumulée $\mathbf{Q}$ est un paramètre de l'état. De plus, la décision à prendre par l'agent dépend des tâches à venir donc des ressources consommables restantes et des tâches à effectuer pour le reste de la mission.

L'état de l'agent va être décrit par quatre variables : les ressources restantes, la qualité accumulée dans la PRU courante, l'unité de raisonnement progressif courante et son indice $\mathtt{p}$ puis l'indice $\mathtt{n}$ du dernier niveau exécuté dans cette PRU. Si, par exemple, le robot vient de terminer le $2^{\text{ème}}$ niveau de la PRU numéro 5, que les modules précédemment exécutés lui ont rapporté une qualité de 15 et qu'il lui reste 153 unités de temps, l'état sera noté $\mathbf{s} = \langle \mathtt{153}, 15, \mathtt{2}, \mathtt{5} \rangle$. Il peut arriver que l'agent commence une tâche qui va consommer plus de ressources qu'il n'en avait au départ ; on regroupe donc tous les états dont la ressource restante serait négative dans un état d'échec $\mathbf{s}_{\mathtt{echec}}$.

Formellement, on écrira que : $\mathcal{S} = \{\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle, \mathtt{r} \in \mathbb{R}\} \cup \{\mathbf{s}_{\mathtt{echec}}\}$.

De plus, on crée un ensemble d'états pour un niveau 0 fictif qui correspond pour chaque PRU aux états initiaux de cette PRU. Dans cet ensemble d'états, la qualité est toujours nulle, puisque l'agent n'a encore exécuté aucun module de cette PRU.

14.3.2. *Les actions de l'agent*

Seulement deux types d'actions sont possibles $\mathcal{A} = \{\mathbf{E}_{\mathtt{m}}, \mathbf{M}\}$ (voir figure 14.3) :

– une action $\mathbf{E}_{\mathtt{m}}$ permet d'exécuter le module $\mathtt{m}$ parmi les modules du niveau suivant de la PRU courante ;

– l'action $\mathbf{M}$ permet de changer de PRU.

L'agent va donc avoir après chaque niveau le choix entre l'amélioration de la tâche courante (la PRU courante) ou l'abandon de la PRU pour la suivante. Le robot ne peut pas sauter de niveau. Cependant, si pour une raison bien définie un niveau est optionnel, on introduira un module vide (*skip*) dans les niveaux concernés. Ce module produira une qualité nulle et ne consommera pas de ressource.

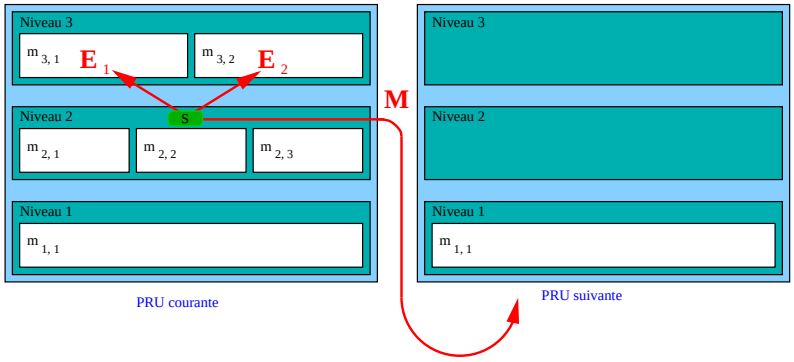


Figure 14.3. *L'agent a réalisé le niveau* 2 *de la tâche (*PRU*) courante (il a pris la photo). Deux types d'actions peuvent être entreprises : exécuter un des modules du niveau suivant (enregistrer cette photo, en effectuant un traitement ou pas),* $\mathbf{E}_1$ *ou* $\mathbf{E}_2$ *c'est-à-dire poursuivre cette tâche, ou alors commencer la tâche suivante.*

Quand l'agent choisit d'exécuter un niveau, il doit choisir parmi un des modules disponibles de ce niveau. Si le dernier niveau est atteint, plus aucun niveau n'est disponible.

Si l'agent décide d'arrêter d'améliorer la PRU courante, il choisit l'action $\mathbf{M}$. Il va alors se trouver immédiatement au début[1] de la PRU suivante où il pourra choisir, une fois de plus, entre l'une des actions $\mathbf{E}_{\mathtt{m}}$ et l'action $\mathtt{m}$.

Par exemple, notre robot peut avoir repéré une roche, mais se rendre compte que, s'il la ramasse, il va perdre un temps précieux pour plus tard. Il peut donc décider d'arrêter la tâche courante, c'est-à-dire ne pas ramasser cette roche puis passer directement à un autre site, pour repérer une autre roche, ou encore continuer vers un deuxième site sans rien faire sur le premier. Le robot ne pourra par contre jamais revenir sur ses pas pour reprendre un site abandonné.

14.3.3. *La fonction de transition*

La fonction de transition $Pr : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ modélise l'incertitude sur la dynamique du système.

L'action $\mathbf{M}$ est déterministe. Si l'agent choisit de changer d'unité de raisonnement progressif, il garde toutes ses ressources restantes. Ainsi

$$Pr(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle, \mathbf{M}, \langle \mathtt{r}, 0, \mathtt{p}+1, 0 \rangle) = 1. \tag{14.1}$$

1. C'est à dire à la fin d'un niveau 0 fictif.

La qualité accumulée dans la nouvelle PRU devient nulle. Le deuxième zéro représente le fait que l'agent est placé juste avant le premier niveau de la $pru_{\mathtt{p}+1}$.

Toute action $\mathbf{E}_{\mathtt{m}}$ est probabiliste. La distribution de probabilité est décrite dans le module à exécuter (exemple sur la figure 14.2). Ainsi

$$Pr(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle, \mathbf{E}_{\mathtt{m}}, \langle \mathtt{r} - \Delta\mathtt{r}, \mathbf{Q} + \mathbf{Q}_{\mathtt{p},\mathtt{n},\mathtt{m}}, \mathtt{p}, \mathtt{n}+1 \rangle) = Pr(\Delta\mathtt{r} | \mathtt{m}_{\mathtt{p},\mathtt{n},\mathtt{m}}). \quad (14.2)$$

14.3.4. *La fonction de récompense*

Dans notre application robotique, le mécanisme de contrôle du robot, tel qu'il a été pensé dans [ZIL 02], prévoit de récompenser le robot après chaque PRU pour ce qu'il a fait. Nous pouvons dès à présent justifier le fait de garder la qualité dans l'état : ceci permet de ne récompenser le robot que si la PRU a été entièrement effectuée.

$$\mathcal{R}(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{N}_{\mathtt{p}} \rangle) = \mathbf{Q}, \quad (14.3)$$

$$\mathcal{R}(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} < \mathtt{N}_{\mathtt{p}} \rangle) = 0. \quad (14.4)$$

Il n'y a pas à pénaliser le robot pour avoir consommé des ressources dans une PRU donnée : en effet la consommation locale entraîne une possible perte de valeur pour le futur, puisque ce qui est dépensé maintenant ne sera plus disponible plus tard dans la mission.

14.4. Calcul de la politique de contrôle

Le mécanisme de contrôle global se base sur un calcul a priori de la politique pour l'ensemble de la mission.

La valeur d'un état ne dépend que de la valeur des états des niveaux suivants et des PRU suivantes (voir l'équation 14.8). La valeur en sortie d'une PRU ne dépend que de la qualité qui y a été accumulée et du temps restant après l'obtention du résultat. Pour contrôler la $pru_{\mathtt{p}}$, il suffit de connaître la fonction de valeur liée à l'action d'exécution des modules $\mathbf{E}$.

La valeur d'un état avant le dernier niveau dépend de la valeur espérée en exécutant les niveaux supérieurs, et en choisissant le meilleur module.

$$V(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle) = \begin{cases} 0 \quad \text{si } \mathtt{r} < 0 \text{ (échec), sinon :} \\ \mathcal{R}(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{N_p} \rangle) \\ \quad + \max(Q(\mathbf{s}, \mathbf{M}), \max_{\mathtt{m}} Q(\mathbf{s}, \mathbf{E}_{\mathtt{m}})) \end{cases} \tag{14.5}$$

$$Q(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle, \mathbf{M}) = \begin{cases} 0 \quad \text{si } \mathtt{p} = \mathbf{P}, \text{ sinon :} \\ V(\langle \mathtt{r}, \mathbf{0}, \mathtt{p}+1, 0 \rangle) \end{cases} \tag{14.6}$$

$$Q(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle, \mathbf{E}_{\mathtt{m}}) = \begin{cases} 0 \quad \text{si } \mathtt{n} = \mathtt{N_p} \text{ (dernier niveau), sinon :} \\ \sum_{\Delta \mathtt{r}} Pr(\Delta \mathtt{r} | \mathtt{m}_{\mathtt{p},\mathtt{n},\mathtt{m}}) . V(\langle \mathtt{r}', \mathbf{Q}', \mathtt{p}, \mathtt{n}+1 \rangle), \\ \quad \text{où} \quad \mathbf{Q}' = \mathbf{Q} + \mathbf{Q}_{\mathtt{p},\mathtt{n},\mathtt{m}}, \\ \quad \text{et} \quad \mathtt{r}' = \mathtt{r} - \Delta \mathtt{r}. \end{cases} \tag{14.7}$$

$$V(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle) = \begin{cases} 0 \quad \text{si } \mathtt{r} < 0 \text{ (échec) ou } \mathtt{p} = \mathbf{P}+1, \quad \text{sinon :} \\ \max(Q(\mathbf{s}, \mathbf{M}), \max_{\mathtt{m}} Q(\mathbf{s}, \mathbf{E}_{\mathtt{m}})); \end{cases} \tag{14.8}$$

$$Q(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle, \mathbf{M}) = \mathcal{R}(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle) + V(\langle \mathtt{r}, \mathbf{0}, \mathtt{p}+1, 0 \rangle); \tag{14.9}$$

$$\begin{aligned} Q(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle, \mathbf{E}_{\mathtt{m}}) &= \mathcal{R}(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle) + \\ &\quad \sum_{\Delta \mathtt{r}} Pr(\Delta \mathtt{r} | \mathtt{m}_{\mathtt{p},\mathtt{n},\mathtt{m}}) \cdot V(\langle \mathtt{r}', \mathbf{Q}', \mathtt{p}, \mathtt{n}+1 \rangle), \\ &\quad \text{où} \quad \mathbf{Q}' = \mathbf{Q} + \mathbf{Q}_{\mathtt{p},\mathtt{n},\mathtt{m}} \quad \text{et} \quad \mathtt{r}' = \mathtt{r} - \Delta \mathtt{r}. \end{aligned} \tag{14.10}$$

NOTE.– attention à ne pas confondre la qualité $\mathbf{Q}$ du module exécuté (propre à ce chapitre) avec la fonction de Q-valeur $Q(\mathbf{s}, \mathbf{a})$ introduite dans le chapitre 1.

On détermine une politique optimale avec l'équation suivante :

$$\pi(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle) = \underset{\mathbf{E}_{\mathtt{m}}, \mathbf{M}}{\operatorname{argmax}}(Q(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle, \mathbf{M}), Q(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \mathtt{n} \rangle, \mathbf{E}_{\mathtt{m}})) \tag{14.11}$$

Ce MDP est sans cycle et à horizon fini. L'algorithme qui permet de calculer la fonction de valeur est donc linéaire en la taille de l'espace d'états. Un état ne devra être évalué qu'une seule fois, et il suffit d'appliquer un algorithme de chaînage avant arrière (en partant d'un état initial) ou par chaînage arrière (en partant de tous les états terminaux possibles). Un algorithme de chaînage avant-arrière fonctionne sur le principe suivant : à partir d'un état initial, on génère les états que l'on peut atteindre avec les actions disponibles à cet instant. On procède ainsi récursivement jusqu'à ce que plus aucun état n'ait de successeur possible. Ces états terminaux sont l'état d'échec et les états de fin de la dernière PRU. On évalue ensuite chaque état par chainage arrière, en calculant la valeur du dernier état puis en propageant cette valeur en arrière. On

obtient finalement une valeur pour l'état initial. La valeur de chaque état est définitive, donc la politique est déterminée en même temps que l'évaluation.

Une fois la politique calculée, on charge celle-ci sur le robot qui peut ensuite effectuer la mission.

14.5. Modéliser concrètement une mission

Ce contrôle de consommation de ressources pour un robot explorateur autonome suppose que

– le graphe acyclique de la mission est connu ;

– les PRU sont déjà modélisées.

Pour cela, il faut connaître préalablement la distribution de probabilité sur la consommation d'une ressource possible pour chaque module qui conpose les PRU. Pour l'instant, ces distributions sont évaluées expérimentalement avant la conception de la mission. On répète une série des tests pour chaque module exécutable par le robot pour obtenir une estimation du montant de ressources qui seront consommées dans chaque module. La figure 14.4 est un exemple d'application qui utilise le raisonnement progressif.

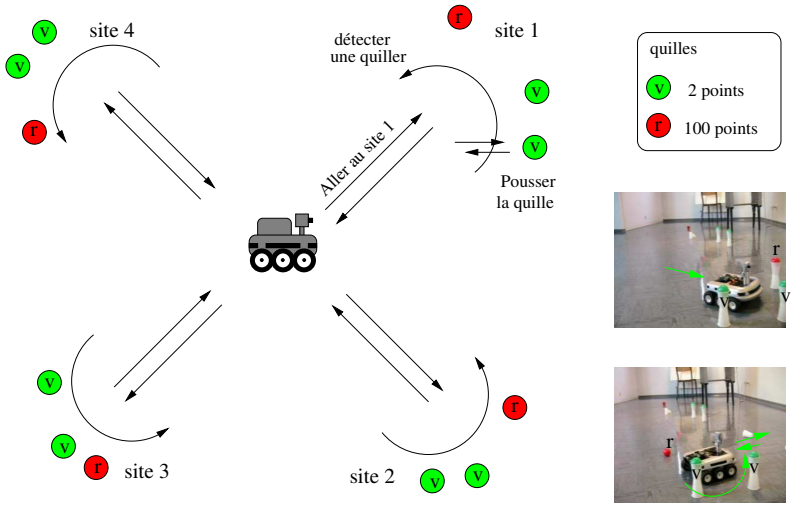


Figure 14.4. *Exemple d'application illustré : un robot est chargé de renverser un ensemble de quilles disposées sur le sol. Le temps imparti pour la mission est inférieur à celui nécessaire pour renverser toutes les quilles. Chaque quille rapporte un nombre de points différents, le robot doit décider à chaque instant s'il renverse la quille ou s'il passe à la suivante. La tâche "renverser une quille" a été modélisée sous forme d'une unité de raisonnement progressif. Les deux photos montrent le robot en action : il détecte les quilles sur le site, puis renverse la quille rouge, qui rapporte plus de points.*

14.6. Extensions possibles

Une extension du contrôle de la consommation d'une ressource unique à plusieurs ressources est proposée [GLO 07]. Une structure agrégée de l'espace d'états permet de limiter sa taille, qui augmente exponentiellement avec le nombre de ressources à gérer. Une autre extension permet de calculer ou de recalculer une politique de contrôle pendant la mission [GLO 07]. Ceci est nécessaire quand l'environnement dans lequel évolue le robot change. En effet, si des tâches apparaissent au cours de la mission, il faut pouvoir décider rapidement combien de ressources doivent être allouées pour la tâche courante, toujours dans l'optique de maximiser la somme des qualités obtenues sur le long terme. La politique de contrôle ainsi calculée n'est plus optimale, mais permet au robot de prendre des décisions dans la PRU courante qui sont souvent optimales.

14.7. Conclusion

Nous venons de présenter une méthode pour contrôler la consommation de ressources d'un robot explorateur autonome. Cette méthode se base sur le raisonnement progressif, qui permet d'effectuer pas à pas une mission composée de tâches hiérarchisées. Il permet également d'adapter la consommation de ressources pour chaque tâche en choisissant le bon module à chaque niveau d'exécution ou en interrompant directement la tâche en cours. Le contrôle de la mission est obtenu en résolvant un MDP avant celle-ci. La modélisation des distributions de probabilité sur la consommation de ressources se fait expérimentalement avant la conception de la mission. Deux extensions ont été mentionnées, l'une pour rajouter différents types de ressources consommables, l'autre pour calculer une politique de contrôle pendant la mission, au détriment de l'optimalité.

Chapitre 15

Planification d'opérations

15.1. Planification d'opérations

Ce chapitre aborde l'application des MDP aux problèmes de planification d'opérations[1] tels qu'on les trouve dans des domaines aussi variés que l'exploration spatiale (rovers, satellites, télescopes), les opérations militaires et la gestion de projets. La planification automatisée [GHA 04] est une branche de l'intelligence artificielle ; elle vise à construire des systèmes génériques capables de choisir et d'organiser les opérations à entreprendre, de manière à atteindre des objectifs donnés à moindre coût. Ici, nous examinons des problèmes de planification complexes qui requièrent non seulement l'exécution parallèle d'opérations, la prise en compte explicite du temps (durée des opérations, instants auxquels les opérations affectent l'état de l'environnement), mais aussi la gestion de l'incertitude liée aux effets des opérations, aux instants auxquels ils se produisent et à la durée des opérations. Il s'agit de problèmes de *planification temporelle probabiliste* [LIT 05, MAU 05, ABE 05]. Nous définissons ces problèmes de façon intuitive puis formelle avant de montrer dans la section 15.2 comment ils se modélisent par des MDP et dans les sections 15.3 et 15.4 comment ils se résolvent en adaptant des algorithmes discutés au cours des chapitres précédents.

15.1.1. *Intuition*

Nous donnons tout d'abord une présentation intuitive du problème, illustrée par un exemple d'exploration spatiale, scénario dans lequel un rover évoluant sur Mars

---

Chapitre rédigé par Sylvie THIÉBAUX et Olivier BUFFET.

1. Nous utilisons le terme « opération » de préférence à « tâche » ou « action ». Nous évitons en particulier ce dernier pour distinguer une « action » d'un problème de planification d'une « action » au sein d'un MDP.

doit effectuer plusieurs expériences sur certains sites et acquérir ainsi des données scientifiques qu'il transmet à la Terre [BRE 02].

15.1.1.1. *Caractéristiques du problème*

Un problème de planification d'opérations se caractérise par :

– **Un environnement/**système décrit par un ensemble de variables d'état (binaires ou multivaluées), par exemple la position et l'orientation du rover, l'énergie et la mémoire disponibles, l'état de ses instruments (calibrés, en marche, etc.).

– La donnée de **l'état initial** du système.

– **Des opérations**, sous le contrôle du planificateur, qui permettent d'agir sur le système – par exemple naviguer d'un point à un autre, effectuer une expérience, transmettre des données, initialiser un instrument. Plusieurs opérations peuvent être déclenchées en même temps, ou encore, une opération peut être déclenchée alors que d'autres sont en cours d'exécution – par exemple, le rover peut devoir utiliser certains instruments en parallèle pour mener à bien sa mission. Une opération ne peut être déclenchée que si certaines *préconditions* sont satisfaites. On dira que l'opération est alors *éligible* – par exemple, pour qu'une expérience soit réalisable, certains instruments doivent être en marche, initialisés et calibrés. Certaines conditions doivent aussi être maintenues *invariantes* pendant un intervalle donné pour garantir la bonne exécution de l'opération.

Une opération a un ou plusieurs *effets* qui se caractérisent par des changements de valeur des variables d'état. Par exemple, les effets d'une opération de navigation incluent le changement de position du robot et la quantité d'énergie consommée. Ici, nous considérons que l'occurrence de ces effets et les durées après lesquelles ils se produisent sont régies par un modèle probabiliste. Par exemple, les effets d'une opération de navigation ainsi que la durée de l'opération dépendent de caractéristiques très précises du terrain qui ne peuvent généralement pas être modélisées explicitement, de sorte que l'on peut considérer que l'opération résulte en deux positions possibles avec une certaine probabilité : soit le rover est arrivé à destination, soit il n'a pu y parvenir et est retourné à sa position d'origine ; dans les deux cas, on peut par exemple considérer que la consommation d'énergie et la durée du déplacement suivent une distribution normale.

– **Des objectifs** : en planification, on cherche classiquement à atteindre un état but, décrit par une affectation de valeurs à certaines des variables d'états – dans notre exemple, le but du rover est d'avoir acquis et transmis certaines données. Plusieurs critères de qualité d'un plan solution peuvent être utilisés. Dans ce chapitre nous considérons essentiellement :

- maximiser la probabilité d'atteindre le but et,
- si le but est toujours atteignable, minimiser le temps ou les ressources utilisés.

Des situations plus complexes pourront par exemple affecter des récompenses différentes à chacun des sous-buts – typiquement, dans notre scénario, les données à acquérir par le rover ont des priorités différentes et le rover n'a pas les ressources nécessaires pour les acquérir toutes.

Un tel problème peut être spécifié dans une variante probabiliste du langage PDDL (*Planning Domain Definition Language*, [FOX 03, YOU 04b]), voir par exemple [LIT 05], ou par le biais d'une interface graphique telle que Brazil (voir figure 15.1).

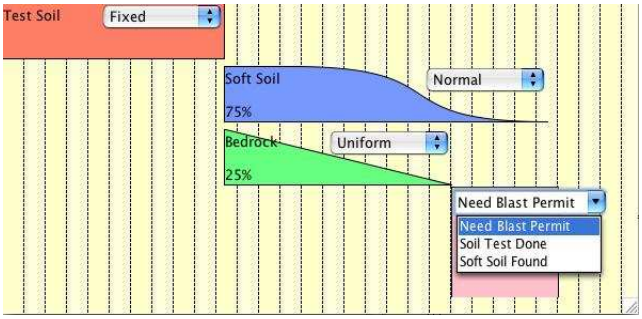


Figure 15.1. *Spécification, à l'aide de l'interface Brazil, d'une opération d'analyse d'un sous-sol avant travaux de construction. Après environ 10 unités de temps, on sait si le bâtiment reposera sur un sous-sol rigide (« bedrock ») ou pas (« soft-soil »). Selon le cas, la durée du reste de l'opération suit une distribution uniforme ou normale.*

15.1.1.2. *Plans*

Il existe plusieurs attitudes possibles face à l'incertitude, attitudes qui déterminent le type de plan solution généré pour résoudre un problème de planification.

– **Plans temporels classiques.** Face à des problèmes de planification pour lesquels les risques liés à l'incertitude sont minimes, il est courant d'utiliser un planificateur déterministe qui ignore l'incertitude du modèle. On génère alors un simple plan temporel, représentable sous la forme d'un diagramme de Gantt (comme au bas de la figure 15.2) qui spécifie les intervalles d'exécution des opérations choisies par le planificateur pour atteindre le but. Si, à l'exécution, un événement inattendu survient, on doit *replanifier*. Une telle approche ne garantit évidemment pas l'optimalité du résultat ni que le but soit atteint, mais permet de bénéficier d'algorithmes de planification très efficaces. Supposons que notre rover doive naviguer jusqu'à un site, y prendre un panorama complet du paysage, puis déterminer la composition du sol. Si la planification ignore l'incertitude quant à la durée et la consommation d'énergie de l'opération de navigation, il est possible que, suite à la prise de panorama, le rover constate qu'il n'a plus assez de temps ou d'énergie pour compléter l'exécution du plan et doive replanifier ou même ajourner la mission.

– **Plans temporels robustes.** Si les risques sont plus importants, il est préférable de tenir compte de l'incertitude. Si les coûts liés à une attitude conservatrice (pertes d'opportunités) sont acceptables, on peut se contenter de générer des plans temporels *robustes* [SCH 05a]. Il peut s'agir de plans *conformants* dont la bonne exécution est garantie quelles que soient les circonstances ou, de façon plus réaliste, de plans flexibles temporellement ou robustes en terme de consommation de ressources. Un

plan robuste pour l'exemple ci-dessus omettrait l'opération de prise de panorama au profit de l'analyse (prioritaire) de la composition du sol si la probabilité d'une insuffisance des ressources de temps ou d'énergie n'est pas négligeable. Ici encore, cette approche n'est pas optimale.

– **Plans temporels contingents.** Enfin, dans le cas général, on génère des plans temporels *contingents* [ABE 04, MAU 05, LIT 05, ABE 05, DEA 03] qui prescrivent, à chaque instant d'intérêt, différentes opérations à déclencher selon l'état de l'exécution. Pour notre exemple, un plan contingent pourrait examiner les ressources disponibles après l'opération de navigation et, selon le cas, prescrire un panorama complet, un demi panorama, ou passer directement à l'analyse de la composition du sol. La figure 15.2 illustre le concept de plan temporel contingent à travers une interface de visualisation. Le haut de la figure montre l'arbre[2] des contingences (exécutions possibles) couvertes par le plan. A la racine de l'arbre, on déclenche une ou plusieurs opérations. Puis le prochain événement qui se produit (un effet d'une opération en cours d'exécution) détermine la branche de l'arbre que l'on prend. Arrivé au nœud suivant, le plan peut encore prescrire le déclenchement de nouvelles opérations et ainsi de suite. Une feuille de l'arbre représente soit une situation de succès (le but est atteint), soit un échec du plan (situation à partir de laquelle le but ne peut plus être atteint, par manque de ressource par exemple). Chaque branche de l'arbre correspond à un plan temporel représentable par un diagramme de Gantt comme celui en bas de la figure.

Dans ce chapitre, nous nous intéressons à la génération de plans temporels contingents.

15.1.2. *Définitions formelles*

Nous définissons maintenant les notions de problème et de plan de manière formelle, mais dans un cadre qui n'est pas celui des MDP.[3] Nous effectuons plusieurs simplifications par rapport à la présentation intuitive. La première est de ne considérer que des variables booléennes. La seconde est l'hypothèse que les opérations n'ont pas de conditions invariantes –des conditions devant être vérifiées pendant toute l'exécution de l'action–, mais seulement des préconditions. La plupart des travaux que nous décrivons se généralisent trivialement au cas multivalué et aux conditions invariantes. La troisième hypothèse concerne les instants de prise de décision des opérations à déclencher, que nous limitons aux instants où un effet d'une opération en cours d'exécution peut se produire, ceci de manière à limiter la combinatoire. Bien que des travaux soient actuellement en cours pour lever cette restriction, celle-ci est

2. Pour des problèmes à horizon infini, on pourra considérer un graphe cyclique plutôt qu'un arbre. Il devient particulièrement difficile de visualiser un tel plan s'il y a une incertitude sur des quantités continues comme la durée des opérations ou les ressources consommées.

3. Nous reviendrons au cadre des MDP en section 15.2

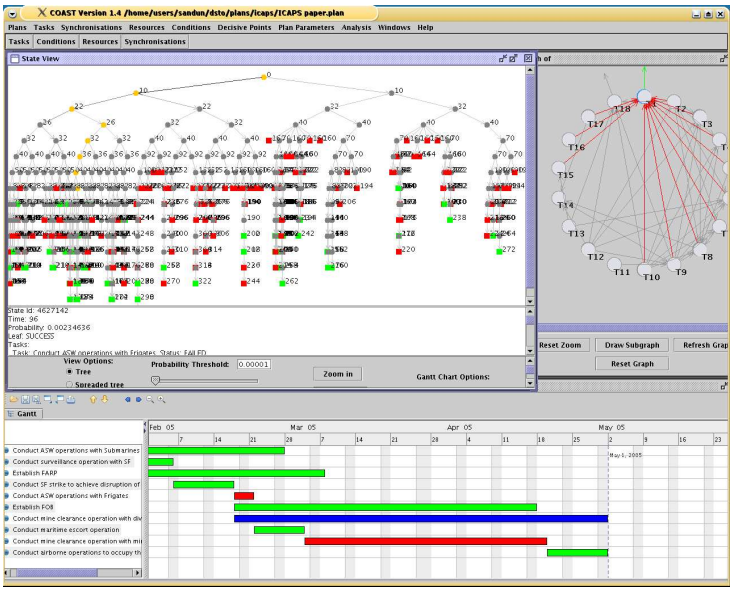


Figure 15.2. *Une interface de visualisation de plans temporels contingents. Dans ce cas particulier, les événements aléatoires sont des fins de tâches qui peuvent déboucher soit sur un échec (à gauche), soit sur un succès (à droite).*

à notre connaissance utilisée dans tous les planificateurs temporels probabilistes existants. La dernière hypothèse est l'absence de distributions continues. Elle sera levée en section 15.4.

15.1.2.1. *Problème de planification, opérations*

DÉFINITION.– *Un* problème de planification d'opérations *est ici défini par un quadruplet* $\langle B, m_0, Op, \Phi \rangle$ *où :*

– $B = \{b_1, \ldots, b_{|B|}\}$ *est un ensemble de* variables booléennes *;*

– m_0 *est l'*état initial *du système, défini par une affectation des variables de* B *;*

– $Op = \{op_1, \ldots, op_{|Op|}\}$ *est un ensemble d'*opérations *(description ci-dessous) ;*

– Φ *est une formule booléenne sur* B *décrivant les situations de succès.*

Cette définition est incomplète, non seulement parce que la notion d'opération n'est pas décrite (voir ci-dessous), mais aussi parce qu'elle ne spécifie pas précisément l'objectif de ce problème de planification (le critère d'optimisation). Ce point est discuté plus loin.

DÉFINITION.– *Une* opération *op est définie par un triplet* $\langle Pre(op), c(op), A(op) \rangle$ *:*

– $Pre(op)$ *: une formule booléenne sur* B *décrivant les préconditions de op,*

– $c(op) \in \mathbb{R}$ *: le coût de l'exécution de op, et*

– $A(op)$ *: un arbre définissant ses effets possibles, et décrit par :*

- des arcs représentant une progression temporelle : à chaque arc est associée une distribution de probabilité P_d *(sur* $\mathbb{R}$*) décrivant la* durée *incertaine entre un nœud et son successeur ;*

- des nœuds internes « conjonctions » *(notés par un « & ») : si un tel nœud est atteint, tous ses descendants doivent être exécutés ;*

- des nœuds internes « hasards » *(notés par un « ? ») et accompagnés chacun d'une distribution de probabilité* $P_?$ *sur ses descendants : si un tel nœud est atteint, un seul de ses descendants doit être exécuté, lequel est choisi au hasard en utilisant la distribution* $P_?$ *;*

- des feuilles « effets » *chacune accompagnée d'un couple* $(b, v_b) \in B \times \{0, 1\}$ *: si une telle feuille est atteinte, la variable* b *prend la valeur* v_b*.*

Quand une opération est déclenchée, ce qui n'est possible que si $Pre(op)$ est vérifiée, son arbre $A(op)$ est exécuté en partant de sa racine. Cet arbre décrit les effets de cette opération, en modélisant aussi bien leur incertitude que les possibles réactions en chaîne et l'écoulement du temps entre deux événements. La figure 15.3 montre l'arbre associé à l'opération d'analyse du sous-sol.

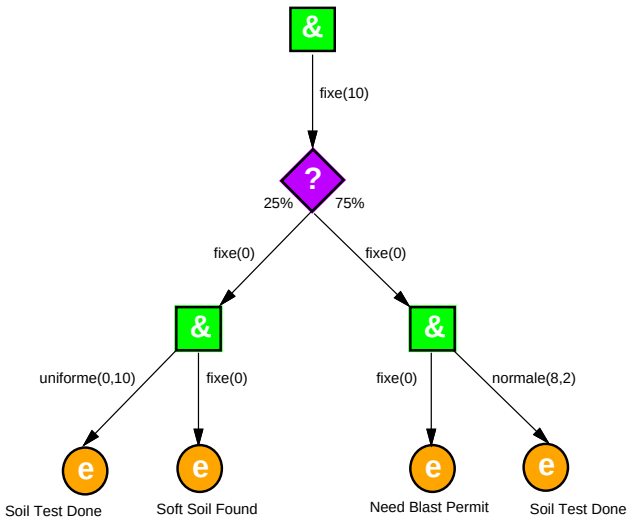


Figure 15.3. *Arbre spécifiant les effets de l'opération d'analyse du sous-sol de la figure 15.1*

NOTE.– Sauf indication contraire, on va considérer uniquement des durées déterministes, la distribution de probabilité P_d étant remplacée par une durée fixe d.

15.1.2.2. *Exécution*

Plusieurs opérations peuvent être en cours d'exécution simultanément. Cette exécution peut être gérée à l'aide d'une file d'événements dans laquelle chaque événement correspond à un nœud d'un arbre d'une des opérations en cours. Cette file est ordonnée selon la date des événements. Le processus d'exécution prend ainsi tous les événements à venir au prochain pas de temps t : $e_1, \ldots, e_n$, les groupes en $E_{\&/?}$ et E_e (ensembles regroupant les nœuds conjonction et hasard d'un côté, et les feuilles (effets) de l'autre), puis :

1) Pour tout $e \in E_{\&/?}$, on retire e de $E_{\&/?}$ et :

- si e est un nœud conjonction, on ajoute tous ses fils à la file ou (pour ceux qui doivent être exécutés immédiatement) dans l'un des deux ensembles $E_{\&/?}$ ou E_e ;

- si e est un nœud hasard, on tire au sort l'un de ses fils conformément à la distribution associée au nœud et ajoute ce fils soit à la file, soit à $E_{\&/?}$ ou E_e.

2) Une fois $E_{\&/?}$ vide, on teste si les effets présents dans E_e ne sont pas en conflits, i.e. ne disent pas de mettre la même variable à la fois à vrai et à faux. Un tel conflit entraîne un échec de l'exécution.

3) S'il n'y a pas de conflit, on exécute chacun de ces effets (on met les variables booléennes aux valeurs spécifiées).

Ce processus est incomplet sans décrire le déclenchement de nouvelles opérations à un instant donné. Chaque opération ne peut être déclenchée que si ses préconditions sont vérifiées. Pour que ces opérations puissent être initiées en parallèle, leurs effets immédiats doivent être compatibles et ne pas violer une des préconditions. Ce modèle du parallélisme est basé sur la notion d'*indépendance* [GHA 04, pp. 119–120] par laquelle plusieurs opérations peuvent avoir lieu en parallèle si et seulement si leur exécution dans n'importe quel ordre est possible et donne lieu à un résultat (ici immédiat) unique. Le déclenchement d'un ensemble d'opérations peut donc être effectué comme suit :

1) On teste si les préconditions de toutes les opérations à déclencher sont vérifiées.

2) On ajoute les racines des arbres de ces opérations à la file d'événements et on traite les événements immédiats comme décrit précédemment.

3) On teste à nouveau si les préconditions des opérations ci-dessus sont vérifiées.

On dira que deux opérations sont *mutuellement exclusives (mutex)* ssi elles ne sont pas indépendantes, c'est-à-dire, ssi leurs préconditions sont incompatibles ou aucune exécution de leurs événements immédiats ne peut réussir.

D'autres définitions de l'exclusion mutuelle proposées dans la littérature tiennent compte de l'exécution complète des opérations et interdisent tout conflit potentiel avec l'un des effets (pas seulement les effets immédiats) des opérations à déclencher ou en cours d'exécution [MAU 05]. On ne peut alors pas combiner des opérations ayant une probabilité extrêmement faible d'entrer en conflit, même si c'est la seule façon d'obtenir un plan adéquat.

15.1.2.3. *Instants de prise de décision, états, plans*

Dans notre cadre, une décision ne peut être prise qu'aux instants où des événements de la file doivent être traités. On déclenche alors un ensemble d'opérations (éventuellement vide) et il faut attendre un instant de prise de décision futur pour tout nouveau déclenchement. Comme nous l'avons dit plus haut, cette hypothèse est restrictive. Même en l'absence d'événement probabiliste, elle mène en général à une perte d'optimalité et de complétude du planificateur [CUS 07]. Il existe cependant certains cas intéressants où l'optimalité est préservée, par exemple pour des opérations dont les préconditions sont aussi des invariants sur l'intervalle d'exécution de l'opération et dont les effets ne se produisent qu'à la fin de l'intervalle [MAU 06].

DÉFINITION.– *Un* état $s \in S$ *est décrit non seulement par l'affectation des variables booléennes, mais aussi par la file d'événements à venir aux instants de prise de décision. L'état initial est* $s_0 = \langle m_0, \emptyset \rangle$.

On peut se passer d'inclure le temps dans la description des états s'il n'y a pas de référence dans le problème à une date absolue (temps limite pour atteindre le but, événement extérieur devant arriver à un instant précis). De la sorte, on va pouvoir assimiler les états qui sont identiques à cette donnée près, ce qui peut réduire la taille de l'espace d'états de manière dramatique (en introduisant des boucles). C'est ce qui peut amener à une représentation d'un plan sous la forme d'un graphe avec cycles.

DÉFINITION.– *Un* plan $\pi : S \rightarrow 2^{Op}$ *est une application qui, à chaque état rencontré lors de son exécution, associe un ensemble d'opérations.*

Il est très courant que le nombre d'états qui peuvent être rencontrés soit bien plus petit que le nombre d'états possibles. Le domaine de définition d'un plan dépend donc du plan lui-même, et se construit en partant de s_0.

15.1.2.4. *Objectif*

La planification consiste à chercher un plan optimisant un critère donné. Différents critères d'optimisation peuvent être employés, utilisant diverses quantités apparaissant dans les définitions qui viennent d'être données.

Probabilité de succès : un premier problème est de *maximiser la probabilité d'atteindre un état but* (état vérifiant Φ et, si on le souhaite, avec une file d'événements vide), c'est-à-dire éviter les échecs d'exécution et les boucles infinies.

A probabilité de succès égale, on va pouvoir distinguer deux plans suivant d'autres critères tels que ceux décrits ci-dessous.

Coût d'exécution : un critère supplémentaire est la *minimisation de l'espérance du coût d'exécution des tâches*.

Durée du plan : un dernier critère classique est la *minimisation de l'espérance de la durée du plan*, laquelle dépend de la durée d'exécution des opérations utilisées. Ce critère incite à exploiter le parallélisme possible lors de l'exécution des opérations.

On peut aussi combiner coût d'exécution et durée du plan, voire probabilité de succès. Il faut pour cela que ces différentes grandeurs soient compatibles, ce qui requiert :

– de remplacer la probabilité de succès par une probabilité d'échec ($P(\text{Échec}) = 1 - P(\text{Succès})$) pour ne considérer que des problèmes de minimisation, et

– de faire en sorte que toutes les unités soient homogènes, ce qui requiert de choisir un « taux de change » entre coût des opérations, temps et probabilité d'échec.

Dans certains problèmes, il n'est pas possible d'effectuer une telle opération. Mais nous ne considérerons ici que le cas simple dans lequel c'est possible.

15.2. MDP

Le problème de planification peut être modélisé sous la forme d'un MDP dont les états sont ceux du problème et dont les actions correspondent aux décisions des ensembles d'opérations à déclencher. Puisque les nombres de points de décision et de décisions possibles croissent tous deux de manière exponentielle avec le nombre d'opérations éligibles, la difficulté principale est de maîtriser la taille du MDP obtenu. Pour ce faire, les travaux se sont orientés vers l'utilisation et l'extension d'algorithmes de recherche heuristique qui explorent une fraction du MDP par essais successifs, tels que l'algorithme (L)RTDP présenté au chapitre 10 et vers la création d'heuristiques appropriées. Ces algorithmes et heuristiques visent à résoudre le MDP de manière optimale, ou proche de l'optimale. Cette section détaille tour à tour ces trois points : modélisation, algorithmes et heuristiques. Nous terminons par un aperçu de travaux en planification automatisée, qui, bien que connexes, ne s'appuient pas directement sur un modèle en termes de MDP.

15.2.1. *Modélisation sous la forme d'un* CoMDP

15.2.1.1. *Etats, actions, transitions*

Pour transformer un problème de planification d'opérations en un MDP, nous commençons par assimiler les *états* de l'un à ceux de l'autre ; un état du MDP consistera donc en une affectation des variables booléennes, une file d'événements, et éventuellement un instant courant. Une *action* est simplement la décision, dans un état donné, de déclencher un *ensemble* (éventuellement vide) d'opérations éligibles et non mutuellement exclusives. Pour cette raison, on parle souvent de CoMDP (MDP concurrents) [MAU 04, MAU 05]. Connaissant états et actions de notre problème, on peut calculer les *probabilités de transition* associées [ABE 04, MAU 05]. En pratique, étant donnés un état et une action, ceci nécessite :

1) d'incrémenter le temps jusqu'à la date du prochain événement de la file ;

2) de simuler le déclenchement de l'action comme décrit au paragraphe 15.1.2.2, de manière à énumérer tous les états successeurs possibles ;

3) de calculer la probabilité d'atteindre chaque état successeur, qui, pour un état donné, est le produit des probabilités des fils des nœuds hasard que l'on insère dans

la file de cet état ; il convient aussi de fusionner les états identiques et de sommer leur probabilités si plusieurs exécutions peuvent y aboutir.

15.2.1.2. *Récompenses, coûts*

La fonction de récompense dépend du critère choisi. Nous en discutons ici en même temps que du type d'algorithme de résolution approprié :

– Probabilité de succès : on donne une récompense nulle par défaut, et unitaire quand un état succès est atteint. La valeur d'un couple état-action $s \times a$ est la probabilité de succès si l'on effectue l'action a dans l'état s, puis on suit la politique optimale. En considérant tout état succès comme absorbant (sans récompense une fois dans un tel état), on obtient un MDP à horizon infini pour lequel l'algorithme d'itération sur les valeurs peut être employé avec le critère total ($\gamma = 1$). On peut toutefois restreindre les états considérés aux états atteignables depuis s_0.

Plutôt que d'utiliser une fonction de récompense, il peut être avantageux de se ramener à un problème de plus court chemin stochastique, où l'on cherche à minimiser l'espérance du coût pour atteindre un état terminal (succès ou échec). On affecte alors un coût nul par défaut et unitaire lorsqu'un état échec est atteint. Si l'on inclut le temps dans l'état, il est toujours possible de garantir que l'on atteindra un état terminal en imposant une borne supérieure sur la durée du plan et en considérant tout état la dépassant comme un échec. On peut donc appliquer les algorithmes classiques de résolution des problèmes de plus court chemin stochastique tels que (L)RTDP [BAR 95, BON 03] et LAO* [HAN 01].

– Coût d'exécution, durée du plan : avec ces deux critères, on se situe aussi dans le cadre d'un problème de minimisation des coûts. Pour une transition $s \times a \rightarrow s'$, le coût $c(s, a, s')$ est dans un cas la somme des coûts des opérations déclenchées par l'action a et dans l'autre cas le temps écoulé entre s et s'. Si, quel que soit l'état atteint, il est toujours possible d'atteindre un état terminal, le MDP obtenu est encore un problème de plus court chemin stochastique, auquel les algorithmes mentionnés ci-dessus s'appliquent.

– Combinaison de critères : idéalement, on souhaite combiner ces critères en privilégiant la minimisation de la probabilité d'échouer. Les algorithmes adverses aux risques [GEI 01], qui optimisent plusieurs critères ordonnés pourraient être étendus dans cette direction. Une alternative beaucoup moins coûteuse mais qui n'offre pas de garantie en général est de combiner les critères via une fonction de coût donnant un poids exponentiellement plus important aux critères privilégiés : s'il y a n critères et $c_i(s, a, s')$ est la fonction de coût relative au critère i, la fonction de coût globale est $c(s, a, s') = \sum_{i=1}^{n} c_i(s, a, s')\ \alpha^i$, avec α suffisamment large [ABE 04].

15.3. Algorithmes

15.3.1. *Résolution exacte*

15.3.1.1. *(L)RTDP*

Les approches présentées dans la littérature se situent toutes dans le cadre de problèmes de plus court chemin stochastique qu'elles résolvent en utilisant des variantes de (L)RTDP. Ceci contribue à gérer l'explosion de la taille du COMDP en le construisant et l'explorant à la volée, guidé par une heuristique.

On a vu au chapitre 10 que RTDP est une version asynchrone de l'algorithme d'itération sur les valeurs qui met à jour les valeurs des états à la fréquence à laquelle ils sont visités par la politique gloutonne [BAR 95]. Ceci permet de se concentrer en priorité sur les états probables et d'obtenir rapidement une politique de qualité, ainsi que d'ignorer totalement les états non-atteignables à partir de s_0. La valeur de chaque état est initialisée de façon heuristique à la première rencontre. RTDP parcourt l'espace d'états par essai successifs, explorant à chaque essai le chemin dicté par la politique gloutonne, et mettant à jour les valeurs des états rencontrés sur le chemin. Le chemin s'arrête lorsqu'un état terminal est atteint (ou lorsque le nombre de mises à jours excède un seuil donné). Le parcours s'arrête lorsque la valeur des états explorés a convergé à un ϵ près. La convergence en temps fini n'est pas garantie et peut être lente si l'heuristique n'est pas suffisamment informative, car il est possible que des états importants mais relativement peu probables ne soient pas visités suffisamment souvent. LRTDP [BON 03] remédie à ces problèmes de convergence en étiquetant les états comme ayant convergé ou non et en concentrant ses efforts sur les régions du MDP n'ayant pas déjà convergé. Si l'heuristique utilisée est admissible (ses valeurs sous-estiment le coût des états), LRTDP converge en temps fini vers la politique optimale.

15.3.1.2. *Gestion de la mémoire*

La taille des COMDP obtenus pour des problèmes de planification d'opérations réels est telle qu'il est nécessaire de modifier (L)RTDP pour le rendre plus efficace, au prix d'une perte d'optimalité et de convergence. En effet, même si la majorité des états n'est pas visitée, des scénarios assez bénins n'impliquant qu'une vingtaine d'opérations nécessitent déjà l'exploration de millions d'états.

En particulier, des mesures supplémentaires sont nécessaires pour limiter la consommation de mémoire de l'algorithme. Lorsqu'une valeur $Q(s, a)$ indique qu'une action est trop coûteuse, les états correspondants ne feront pas partie de la solution finale. De tels états constituent la majorité des états en mémoire et sont obsolètes, même s'ils peuvent être ponctuellement revisités par la politique courante. Ceci motive donc la politique de gestion de la mémoire suivante : on enlève de la table mémorisant les états tout état n'étant pas atteignable via la politique courante et dont la fréquence d'apparition récente est en dessous d'un certain seuil [ABE 04]. Si un tel état est revisité par la suite, il conviendra alors de réapprendre sa valeur.

15.3.1.3. *Réduction du nombre de mises à jour*

Il convient aussi de limiter le temps de calcul qui est dominé par le fait que le nombre de mises à jour à effectuer[4] pour chaque état visité est linéaire en le nombre d'actions du MDP et donc exponentiel en le nombre d'opérations du problème de planification. Pour réduire le temps de calcul, on n'effectue pour chaque état qu'un ensemble aléatoire de mises à jour, à partir d'une distribution qui favorise les combinaisons d'opérations les plus « importantes » [MAU 04]. Cette distribution est générée : (1) en mémorisant et utilisant les combinaisons pour lesquelles on a déjà découvert des Q-valeurs faibles, et (2) en calculant les Q-valeurs de toutes les actions correspondant à une opération unique et en biaisant la distribution en faveur des combinaisons contenant des opérations à faible Q-valeurs.

15.3.1.4. *Algorithmes hybrides*

La résolution de problèmes de planification d'opérations a aussi motivé l'utilisation d'algorithmes hybrides qui emploient un algorithme optimal pour les états fréquemment visités et un algorithme rapide mais sous-optimal pour les autres états [MAU 05]. Par exemple, on peut hybrider l'algorithme RTDP opérant dans le CoMDP définit ci-dessus et une version de RTDP opérant dans un CoMDP « aligné » dans lequel on ne peut prendre de nouvelle décision qu'une fois que toutes les opérations dernièrement déclenchées ont terminé. La figure 15.4 illustre ce concept de CoMDP aligné. Celui-ci est beaucoup plus simple que notre CoMDP « imbriqué » car les états n'ont plus besoin de maintenir une file d'événements.

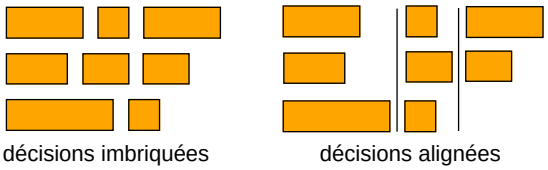


Figure 15.4. *Décisions imbriquées et alignées. Dans le cas aligné, les décisions n'ont lieu qu'aux instants matérialisés par les lignes verticales.*

Avec le CoMDP aligné, RTDP converge rapidement car l'espace d'états est plus petit, mais il produit une politique sous-optimale pour le problème initial. Avec le CoMDP imbriqué, RTDP génère la politique optimale, mais la convergence est très lente. L'algorithme hybride utilise donc RTDP dans le CoMDP imbriqué assez longtemps pour qu'il génère une « bonne » politique pour les états fréquents mais l'interrompt bien avant qu'il converge pour les états moins fréquents ; à partir de chaque état dont la fréquence des visites par la politique imbriquée est insuffisante, on crée une politique alignée en laissant RTDP converger sur le CoMDP aligné. La politique hybride retournée consiste en la politique imbriquée pour les états fréquents et la politique alignée

4. Le calcul du $\min$, dans la ligne $L(s) \leftarrow \min_{a \in A} Q(s,a)$ de l'algorithme RTDP.

pour les autres. De plus, pour garantir que l'on atteindra toujours un état terminal, on crée aussi une politique alignée pour tout état précédant un état puits atteint par la politique imbriquée, ainsi que pour tout état faisant partie d'un cycle de la politique imbriquée.

L'algorithme hybride alterne donc k essais RTDP sur le CoMDP imbriqué et des exécutions de RTDP jusqu'à convergence sur le CoMDP aligné pour les états peu fréquents, les états puits et les cycles. L'alternance s'arrête lorsque le coût de la politique hybride approche celui de la politique optimale. Ceci peut être implémenté de la façon suivante [MAU 05, figure 4]. On remarque que l'estimation $C_i(s_0)$ du coût de l'état initial fourni par RTDP dans le CoMDP imbriqué est toujours inférieure au coût de la politique optimale (moyennant l'utilisation d'heuristiques admissibles) et que le coût réel $C_h(s_0)$ de la politique hybride (que l'on peut évaluer par simulation), est toujours supérieur à celui de la politique optimale. On peut donc s'arrêter lorsque $\frac{C_h(s_0)-C_i(s_0)}{C_i(s_0)} < r$ pour un r donné.

15.3.2. *Résolution heuristique*

Les algorithmes de recherche tels que (L)RTDP initialisent de manière heuristique l'estimation du coût d'un état à leur première rencontre. Une heuristique est une fonction $h(s) \geq 0$ estimant le coût optimal $C^*(s)$ pour atteindre un état terminal à partir de l'état s. Pour garantir la convergence de LRTDP vers la politique optimale, il suffit que h soit admissible : $h(s) \leq C^*(s)$ pour tout état s. Nous détaillons maintenant certaines des heuristiques utilisées dans le cadre de la planification d'opérations. Les deux premières sous-sections font l'hypothèse que tous les effets des opérations se produisent à la fin de leurs intervalles d'exécution respectifs. La durée de l'opération op est notée $\Delta(op)$. Toutes les heuristiques présentées font l'hypothèse que le but Φ du problème est une conjonction, c'est-à-dire un ensemble de sous-buts.

15.3.2.1. *Heuristiques élémentaires*

Le système MOP [ABE 04] implémente plusieurs heuristiques admissibles simples et extrêmement rapides à calculer, estimant respectivement la probabilité d'échec du plan, l'espérance de sa durée totale et l'espérance du coût des opérations qu'il prescrit. Nous décrivons les deux dernières ci-dessous. Ces heuristiques sont ensuite combinées pour estimer le coût total comme décrit au paragraphe 15.2.1.2 – via une pondération exponentielle. Dans la suite, on note $buts(\Phi, s)$ l'ensemble des sous-buts de Φ qui ne sont pas encore satisfaits dans l'état s et $prod(\phi)$ l'ensemble des opérations ayant le sous-but ϕ parmi leurs effets possibles.

Nous commençons par exprimer une borne inférieure de la durée du plan : le maximum de toutes les durées requises pour établir chacun des sous-buts. La durée requise pour établir un sous-but donné est le minimum des durées des opérations pouvant produire le sous-but. Ceci donne :

$$h^{el}_{\Delta}(s) = \max_{\phi \in buts(\Phi,s)} \min_{op \in prod(\phi)} \Delta(op) \leq C^*_{\Delta}(s)$$

Il s'agit bien d'une borne inférieure puisque 1) on ne considère que le coût de la production du sous-but le plus critique, 2) on ignore le fait que produire un sous-but peut en détruire un autre et 3) on fait l'hypothèse que l'on contrôle le résultat probabiliste des opérations.

On peut construire une borne inférieure du coût du plan (e.g. sa consommation de ressources) de façon similaire, en sommant les coûts minimaux nécessaires à la production de chaque sous-but, mais il faut faire attention à ne pas sommer plusieurs fois le coût d'une opération pouvant produire plusieurs sous-buts. Le calcul exact de l'ensemble d'opérations de moindre coût produisant tous les sous-buts est un problème NP-difficile, mais l'on peut construire une sous-approximation en divisant le coût d'une opération par le nombre de sous-buts qu'elle produit. Ainsi, si une opération a le coût le plus faible pour tous les sous-buts qu'elle produit, l'opération contribuera pour son coût exact à la somme. Si elle n'a le coût le plus faible que pour un sous-ensemble des sous-buts produits, alors sa contribution sera inférieure au coût exact :

$$h_c^{el}(s) = \sum_{\phi \in buts(\Phi,s)} \min_{op \in prod(\phi)} \frac{c(op)}{|\{\phi' \in buts(\Phi,s) : op \in prod(\phi')\}|} \leq C_c^*(s)$$

15.3.2.2. *Heuristiques obtenues par relaxation du* CoMDP

Des heuristiques plus informatives peuvent être obtenues en résolvant de façon optimale une relaxation du CoMDP initial, i.e. un problème plus simple. Une telle relaxation est plus facile à résoudre et produit une politique optimale moins coûteuse que l'originale qui peut donc servir d'heuristique admissible.

Le système DUR [MAU 05] met en œuvre par exemple deux heuristiques obtenues par relaxation pour estimer l'espérance de la durée du plan. La première, l'heuristique de « concurrence maximale », est basée sur l'observation que la durée moyenne de la politique optimale pour le MDP séquentiel (aucun parallélisme permis), divisée par le nombre maximal d'opérations pouvant avoir lieu en parallèle à tout instant, est une borne inférieure de la durée moyenne optimale du CoMDP. On calcule donc cette heuristique de la façon suivante. On résout le MDP séquentiel dont les états consistent seulement en une affectation des variables booléennes, dont les actions sont les opérations du problème et pour lequel le coût d'une transition est la durée de l'opération impliquée. On obtient une politique optimale de coût $C^*_{seq}(m)$ pour chaque état m du MDP séquentiel. On résout alors le CoMDP initial avec l'heuristique :

$$h^{mc}_{\Delta}(s) = \frac{C^*_{seq}(m(s))}{maxconc} \leq C^*_{\Delta}(s)$$

où $m(s)$ est l'affectation des variables booléennes de l'état s et $maxconc$ est le nombre maximal d'opérations pouvant s'exécuter en parallèle à tout instant.

La deuxième heuristique mise en œuvre dans le système DUR est l'heuristique dite de « l'anticipation des effets ». L'idée est de prétendre que l'on connaît tous les effets des opérations dernièrement déclenchées sans attendre, c'est-à-dire dès le prochain

instant de décision. L'état du CoMDP résultant est représenté par un couple (m, δ), ou m est l'état du système qui tient compte des effets que l'on connaît par anticipation et δ est la durée jusqu'à la date de terminaison la plus tardive des opérations en cours d'exécution. Intuitivement, ceci signifie que l'on atteindra l'état m après une durée δ. La figure 15.5 donne un exemple dans lequel, à partir de l'état m du système, on déclenche 3 opérations se terminant respectivement après 2, 4 et 8 unités de temps et dont les effets cumulés nous mènent à l'état m' du système. L'état résultant du CoMDP relaxé est $(m', 6)$, car les effets sont connus dès la terminaison de l'opération b après deux unités de temps et il reste 6 unités jusqu'à terminaison de la dernière opération (a). Le coût d'une transition du CoMDP relaxé représente la durée de laquelle le temps avance entre deux états. Elle correspond à la durée jusqu'à la prochaine terminaison d'une action en cours d'exécution.

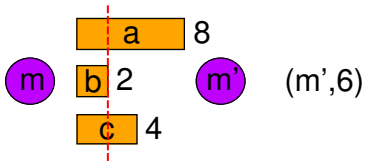


Figure 15.5. *Anticipation d'effets.*

Le CoMDP résultant est bien d'une relaxation car (1) on a d'avantage d'information plus tôt sur ce qui va se produire et (2) on a perdu la trace des instants auxquels les différentes actions se terminent et on autorisera par conséquent le déclenchement d'opérations dont les préconditions ne sont pas forcément vérifiées. A partir d'une solution optimale, de coût $C^*_{ee}(\sigma)$ pour chaque état $\sigma = \langle m, \delta \rangle$ du CoMDP relaxé, on obtient une heuristique pour le MDP original de la façon suivante :

$$h^{ee}_{\Delta}(s) = \sum_{m' \in 2^B} p(m' \mid m(s), ex(s)) C^*_{ee}(\langle m', last(s) \rangle) \leq C^*_{\Delta}(s)$$

où $ex(s)$ est l'ensemble des opérations en cours d'exécution dans l'état s et $last(s)$ est la durée jusqu'à la terminaison la plus tardive d'une action de $ex(s)$.

On se référera à [MAU 06] pour une extension de ce type d'heuristiques et des algorithmes hybrides au cas de durées stochastiques.

15.3.2.3. *Heuristiques basées sur le graphe de planification*

Le graphe de planification est une structure de données souvent utilisée pour construire des heuristiques [BRY 06]. Il peut être vu comme une approximation de l'espace d'état du problème. L'origine de ce graphe est le planificateur atemporel et déterministe Graphplan [BLU 97], qui l'utilise pour déterminer s'il est possible, à partir de l'état initial, d'atteindre un état but en moins de n étapes — à chaque étape, on autorise l'exécution parallèle d'un ensemble d'opérations non-mutex qui sont toutes supposées se terminer avant l'étape suivante. Le graphe fournit, en temps polynomial,

une condition nécessaire mais insuffisante à l'atteignabilité. Une extension du graphe de planification permet d'obtenir une borne inférieure de la probabilité d'échec pour des problèmes de planification temporels probabilistes tels que nous les avons définis [LIT 05]. Dans ce cadre, l'estimation consiste en trois étapes : (1) construction du graphe, (2) association de bornes inférieures aux nœuds du graphe, puis (3) combinaison de ces bornes pour arriver à une estimation heuristique de la probabilité d'échec d'un état donné du CoMDP. Nous considérons chacune de ces étapes tour à tour.

Étape 1 : le graphe et sa construction

Le graphe de planification traditionnel (déterministe, a-temporel) consiste en une alternance de niveaux dont chacun est constitué soit de nœuds représentant des propositions (variables booléennes), soit de nœuds représentant des opérations. Il se construit à partir de la description des opérations, en partant de l'état initial du problème de planification. Les successeurs d'un nœud opération d'un niveau donné sont les propositions du niveau suivant représentant les effets positifs de l'opération (les variables que l'opération met à la valeur 1) et ses prédécesseurs sont les propositions du niveau précédent représentant ses préconditions.[5] On se référera à [GHA 04, chap. 5] pour les détails de la construction du graphe et de ses propriétés. Nous mentionnons simplement ici que le graphe est de taille polynomiale en la taille du problème et que la présence d'un fait ou d'une opération à un niveau n donné indique qu'il n'est *pas impossible* que le fait soit atteignable ou que l'opération soit exécutable après n étapes.

Par rapport au graphe traditionnel, le graphe de planification temporel probabiliste contient un type de nœud supplémentaire : les nœuds *hasard*. Ceux-ci correspondent aux nœuds hasard des arbres définissant les effets des opérations. Avec cette extension, les nœuds opération sont maintenant reliés à des nœuds hasard, les nœuds hasard à des nœuds proposition ou à d'autres nœuds hasard et les nœuds proposition à des nœuds opération. Afin de traiter les aspects temporels du problème, chaque niveau est associé à une date. Les successeurs d'un nœud hasard ne se trouvent pas nécessairement au niveau « suivant », mais au niveau dont la date est appropriée.

Étape 2 : Calcul du coût des nœuds du graphe

On génère le graphe à partir de l'état initial du problème jusqu'à un horizon temporel (une date donnée). On calcule ensuite, par propagation arrière, pour chaque nœud n du graphe, un vecteur de coûts $c_n[i]$, qui reflète la capacité du nœud à contribuer à l'atteignabilité du $i^{\text{ième}}$ sous-but Φ_i. Un coût nul signifie que le nœud (éventuellement en combinaison avec d'autres) est capable de rendre le sous-but inévitable, alors qu'un coût de 1 signifie que le nœud n'est pas pertinent à l'atteignabilité du sous-but. Les vecteurs de coûts des nœuds du dernier niveau du graphe, qui sont des nœuds proposition, sont initialisés de la façon suivante : $c_n[i] = 0$ si $n = \Phi_i$ et $c_n[i] = 0$ sinon. Puis les coûts sont propagés en arrière, selon des règles qui garantissent l'admissibilité et

5. On suppose que $Pre(op)$ est une conjonction de propositions atomiques.

qui dépendent du type de nœud, hasard (h), opérations (o), ou proposition (p) :

$$\begin{array}{lcl} c_n^h[i] & := & \prod_{n' \in Succ(n)} c_{n'}^{p,h}[i] \\ c_n^o[i] & := & \sum_{n' \in Succ(n)} \Pr(n')\, c_{n'}^h[i] \\ c_n^p[i] & := & \prod_{n' \in Succ(n)} c_{n'}^o[i] \end{array}$$

où $Succ(n)$ est l'ensemble des successeurs du nœud n du graphe.

Étape 3 : Estimation du coût d'un état du CoMDP

Enfin, les composants des vecteurs de coût sont combinés, toujours de manière admissible, pour estimer le coût d'un état donné du CoMDP. On considère pour cela les nœuds du graphe qui sont pertinents pour l'état. Il s'agit des nœuds propositions ou hasard du graphe (au niveau dont la date est appropriée) qui représentent les variables booléennes à vrai dans l'état, ainsi que les effets et les nœuds hasard de la file d'événements [LIT 05]. On fait ensuite le produit des vecteurs de coûts des nœuds pertinents et on retient comme estimation finale la composante maximale du produit. Celle-ci représente la valeur associée au sous-but le plus difficile à atteindre :

$$h_{\Pr}^{gp}(s) = \max_{i \in |\Phi|} \Pi_{n \in \text{pertinents}(s)} c_n[i] \leq C_{\Pr}^*(s)$$

Pour finir, notons que le graphe de planification n'est pas limité à l'estimation de l'atteignabilité et de la probabilité d'échec, mais est souvent utilisé pour estimer la durée ou le coût d'un plan [BRY 06].

15.3.3. *Autres approches à base de modèles*

Outre l'approche MDP, les recherches en planification automatisée étudient une multitude d'autres modélisations et algorithmes. Ceux-ci vont de représentations à base de systèmes de transitions et d'algorithmes de recherche heuristique, aux représentations logiques ou à base de contraintes et d'algorithmes de statisfiabilité, *model-checking*, de démonstration de théorème et de propagation de contraintes, en passant par des représentations et algorithmes basés sur les graphes [GHA 04]. Nous donnons ici un aperçu des travaux les plus pertinents en planification temporelle probabiliste.

15.3.3.1. *Recherche heuristique dans un graphe ET-OU*

Prottle [LIT 05] est un système de planification temporelle probabiliste utilisant une modélisation à base de systèmes de transition représenté par un graphe ET-OU et un algorithme de recherche heuristique inspiré à la fois par LRTDP et AO*. Les états du systèmes de transition sont les états du problème de planification comme définis précédemment. Les transitions correspondent au choix d'une opération à déclencher à l'instant courant, à l'avancement du temps jusqu'à l'instant du prochain événement de la file et au traitement d'un événement de la file. Une différence, par rapport à la modélisation en terme de CoMDP, est donc que l'on décompose une transition du MDP en autant de transitions que d'opérations déclenchées ou d'événements traités. De plus, un état du système est un état OU si l'on y choisit une opération à déclencher

ou un événement déterministe à traiter et un état ET si l'on y traite un événement probabiliste.

L'algorithme de recherche opère par essais successifs comme (L)RTDP, et utilise non seulement une borne inférieure $L(s)$ sur le coût de chaque état s, mais aussi une borne supérieure $U(s)$. La borne inférieure est initialisée par l'heuristique basée sur le graphe de planification discutée à la sous-section précédente. Le coût d'un état converge lorsque $U(s) - L(s) \leq \epsilon$. L'algorithme met à jour les bornes et les labels (« convergé ») des états du chemin uniquement lorsque la fin du chemin (un état terminal succès ou échec) est atteint. Les formules de mise à jour des bornes d'un état en fonction de celles de ses successeurs $Succ(s)$ dépendent du type de borne (L-U) et du type d'état (ET-OU) considérés :

$$\begin{array}{lcl} L_{OU}(s) & := & \max(L(s), \min_{s' \in Succ(s)} L(s')) \\ U_{OU}(s) & := & \min(U(s), \min_{s' \in Succ(s)} U(s')) \\ L_{ET}(s) & := & \max(L(s), \sum_{s' \in Succ(s)} \Pr(s')\, L(s')) \\ U_{ET}(s) & := & \min(U(s), \sum_{s' \in Succ(s)} \Pr(s')\, U(s')) \end{array}$$

LRTDP sélectionne le prochain état du chemin à explorer aléatoirement parmi ceux résultant de l'exécution de la politique gloutonne. L'algorithme de recherche de Prottle sélectionne le prochain état de façon déterministe, implémentant une stratégie qui vise à générer le plus rapidement possible un chemin vers un état succès, puis à rendre robuste les chemins connus vers le but. Formellement, l'état sélectionné est le successeur s (parmi ceux n'ayant pas encore convergé) maximisant $\Pr(s)U(s)$ et à résultat égal, celui maximisant $\Pr(s)L(s)$.

De part son algorithme de recherche qui implémente une stratégie de planification intéressante et qui exploite à la fois borne inférieure et supérieure, Prottle constitue une alternative intéressante aux modélisations et algorithmes basés sur les MDP.

15.3.3.2. *Algorithmes basés sur le graphe de planification*

Le graphe de planification évoqué dans la sous-section précédente n'est pas seulement une source de génération d'heuristiques, mais aussi un espace qui peut être exploré (traditionnellement par recherche arrière) pour générer un plan concurrent [GHA 04, pp. 125-129]. Graphplan [BLU 97] a été le premier à exploiter cette idée qui est maintenant présente dans de nombreux planificateurs. Paragraph [LIT 06] est une extension de Graphplan pour les systèmes probabilistes équivalents aux CoMDP. Paragraph forme un plan contingent optimal en concaténant des sous-trajectoires générées par Graphplan.

Cette idée de concaténer des trajectoires est aussi présente dans des travaux sur la génération incrémentale de plans contingents [DEA 03]. Ici, le but n'est pas de générer un plan optimal, mais un plan couvrant les contingences les plus utiles. L'identification de ces contingences, i.e., l'estimation de leur utilité, s'appuie sur une propagation arrière des fonctions d'utilité dans le graphe de planification.

Paragraph est un planificateur probabiliste concurrent, mais non-temporel. Des travaux sont en cours pour l'étendre au cas temporel, en compilant les opérations temporelles en opérations instantanées que Paragraph sait déjà traiter et en gérant les contraintes de temps entre ces dernières opérations via un algorithme de programmation linéaire. Cette façon de combiner graphe de planification et programmation linéaire est inspirée par le planificateur temporel déterministe LPGP [LON 03] et constitue une voie extrêmement prometteuse pour la planification temporelle probabiliste.

15.3.3.3. GSMDP

Le processus de décision semi-Markovien généralisé (GSMDP) [MAT 62] est un formalisme puissant pour décrire des systèmes à événements discrets composés de processus asynchrones opérant en temps continu ou discret et en présence d'incertitude. Comme son nom l'indique, un GSMDP est une généralisation des processus de décision semi-Markovien (SMDP), qui permet de modéliser des durées dont les distributions de probabilité (qui peuvent avoir recours à une mémoire, comme dans les SMDP) ne dépendent pas seulement de l'état courant, mais de la trajectoire suivie par le système. Un GSMDP peut être vu comme la composition de SMDP concurrents.

Les GSMDP permettent de modéliser des problèmes de planification d'opérations plus généraux que ceux que nous avons considérés dans cette section [YOU 03a]. En particulier, le temps continu, les événements hors du contrôle du planificateur, les distributions généralisées qui requièrent une mémoire (par exemple Weibul) sont facilement modélisés, tout en permettant des événements et actions concurrents.

Les techniques pour résoudre ces problèmes de planification très généraux incluent des méthodes génère-teste-répare, fondées sur une évaluation par échantillonnage probabiliste de la politique courante [YOU 03b], ou encore l'approximation du GSMDP par un MDP s'appuyant sur l'approximation des distributions généralisées par des lois phase type [YOU 04a]. Bien que ces techniques n'offrent pas de garantie d'optimalité, le formalisme GSMDP constitue une avenue de recherche prometteuse pour la planification temporelle probabiliste.

15.4. Apprentissage par renforcement : FPG

15.4.1. *Employer des méthodes approchées*

Tous les algorithmes présentés jusqu'ici pour résoudre des problèmes de planification d'opérations passent par l'estimation de l'utilité de couples état-action. Comme dans la majorité des approches de résolution de MDP, cela permet de déterminer la meilleure action à effectuer dans chaque état, ou au moins dans chaque état visité par la politique optimale trouvée.

Mais le nombre d'états concernés est souvent très grand, ce qui rend de tels algorithmes très gourmands en mémoire, même si de bonnes heuristisques sont employées

pour limiter leur exploration. Certains pallient la croissance exponentielle de ces besoins en mémoire en faisant des approximations, comme on l'a vu dans le cas d'algorithmes hybrides en section 15.3.1.4. Mais une approche qui n'a été proposée que récemment est d'utiliser des fonctions d'approximation, que ce soit pour approcher une fonction de valeur (chapitre 11) ou pour définir une politique (chapitre 12).

L'algorithme FPG (*Factored Policy-Gradient*) [ABE 05] propose ainsi d'employer une des méthodes de gradient stochastiques vues au chapitre 12 pour résoudre des problèmes de planification d'opérations. Le principe est de faire fonctionner un de ces algorithmes d'apprentissage par renforcement en interaction avec un simulateur du problème de planification (on suppose un tel simulateur disponible). Pour rappel, dans ce cadre une politique est vue comme une fonction paramétrée, l'apprentissage correspondant à l'optimisation des paramètres. Pour présenter FPG plus en détails, nous allons principalement discuter de deux points importants dans sa conception :

1) le choix de la forme de la politique paramétrée, et

2) le choix de l'algorithme d'optimisation le plus approprié.

15.4.2. *Politique paramétrée*

Pour faire le choix de la forme de la politique paramétrée, nous partons des contraintes que posent les entrées et sorties requises. L'objectif est de trouver un bon compromis pour que la forme dépende d'un petit nombre de paramètres mais puisse toujours représenter des politiques efficaces.

15.4.2.1. *Entrées*

Parce qu'un problème de planification d'opérations est doté au départ d'une certaine structure, il est naturel de prendre comme entrées de l'approximateur de fonction non un simple numéro identifiant l'état courant, mais un vecteur dépendant de cet état courant. Dans notre cadre de travail, une information complète est donnée par 1) les variables booléennes du modèle et 2) la file des événements à venir prévus. Du fait de sa longueur variable, une file d'événements se prête mal à une représentation sous la forme d'un vecteur, à moins de ne garder qu'un nombre réduit d'événements (les plus proches dans le temps par exemple). Dans le planificateur FPG, le choix fait est de restreindre le vecteur d'entrée $\vec{o}$ aux variables booléennes.

15.4.2.2. *Sorties*

A chaque instant de prise de décision, il faut déterminer pour chaque opération éligible si elle doit être exécutée ou pas. Mais, idéalement, une politique paramétrée renvoie une distribution de probabilité sur les actions possibles, une action correspondant au déclenchement d'un ensemble d'opérations éligibles. Cela pose un problème parce que :

– le nombre d'actions possibles croît de manière exponentielle avec le nombre d'opérations éligibles et

– une distribution de probabilité sur un nombre d'actions variable paraît difficile à représenter avec une fonction d'approximation.

Pour résoudre ce problème, FPG emploie un contrôleur *factorisé* en un sous-contrôleur par opération. Pour l'entrée $\vec{o}$ courante, FPG calcule ainsi pour chaque opération éligible op une probabilité de l'exécuter $P(op|\vec{o};\vec{\theta}_{op})$, puis échantillonne un sous-ensemble de ces opérations. Mais encore faut-il ne pas déclencher des opérations mutex. Pour cela, la solution employée par FPG consiste simplement à identifier les opérations en conflit et à en retirer aléatoirement jusqu'à ce que tous les conflits soient résolus.

15.4.2.3. *Fonction d'approximation*

Ayant défini entrées et sorties, divers approximateurs de fonction peuvent encore être employés. En pratique, les implémentations de FPG ont utilisé jusqu'ici des arbres de décision et, surtout, des réseaux de neurones de type perceptron. Les meilleurs résultats ont été obtenus avec des perceptrons sans couche cachée, aussi appelés réseaux linéaires (voir équation (12.4)).

La figure 15.6 montre un tel contrôleur basé sur des réseaux linéaires. Dans ce cas particulier, un état contient non seulement une file d'événements et des variables booléennes (appelées « prédicats »), mais aussi le temps et des ressources. Le contrôleur est ici vu en interaction avec un simulateur via la fonction $chercheSuccesseur(s_t, \vec{a}_t)$ qui échantillonne l'état suivant s_{t+1} en fonction de l'état courant s_t et du vecteur action choisi $\vec{a}_t$.

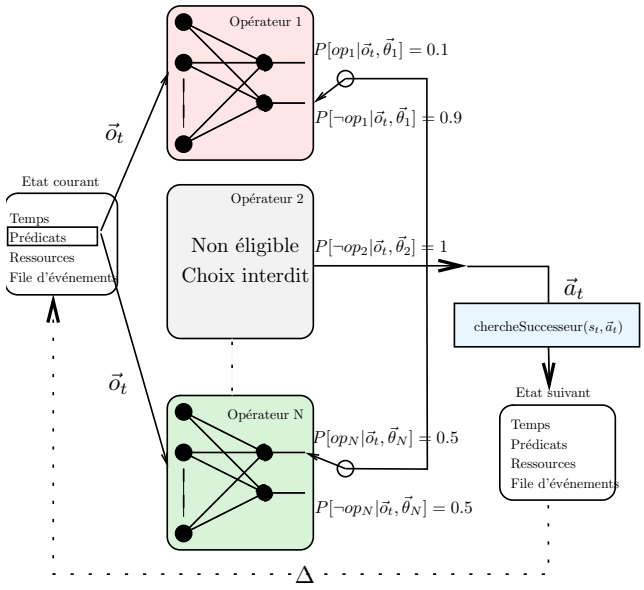


Figure 15.6. *Schéma de principe du contrôleur FPG*

On notera que les méthodes de gradient peuvent au besoin gérer des entrées et des sorties continues. On peut donc en théorie aborder des problèmes plus complexes que ceux présentés ici. Un exemple type est la gestion de projets dans lesquels les durées des tâches sont incertaines.

15.4.3. *Méthodes de gradient*

15.4.3.1. *Terminaison d'une exécution*

De tels problèmes de planification d'opérations se terminent quand un état terminal est atteint, qu'il s'agisse d'un succès ou d'un échec. Mais il est aussi possible que des exécutions de politique ne se terminent jamais. En particulier, il est possible de se trouver dans une partie de l'espace d'états sans issue, quelle que soit la politique. Pour éviter qu'un algorithme d'apprentissage se retrouve ainsi bloqué, il faut définir une durée maximale $T_{\max}$ au bout de laquelle tout état est considéré comme un échec. Cette durée peut être mesurée en unités de temps (secondes, heures, jours...) ou en nombre de points de décisions.

15.4.3.2. *Choix d'*`OLpomdp`

Parce que l'on garantit ici la terminaison de toute exécution, on peut adopter une méthode de gradient pour processus régénératif (voir section 12.2.3.1). Les implémentations actuelles de FPG utilisent toutefois l'algorithme `OLpomdp` (algorithme 12.3, page 369) pour bénéficier de l'efficacité de son apprentissage en ligne. On préfère aussi `OLpomdp` à d'autres algorithmes plus coûteux en temps de calcul, telles que les méthodes « Acteur-Critique » vues en section 12.3, parce que l'on bénéficie ici d'échantillons à très faible coût si un simulateur rapide existe.

15.4.3.3. *Critère optimisé*

Tel qu'il a été décrit, `OLpomdp` optimise la récompense moyenne par pas de simulation. Or, dans le cas de la planification d'opérations, le critère qu'il faudrait employer est la récompense moyenne par exécution complète (par chemin d'un état initial à un état terminal). Au lieu de maximiser par exemple la probabilité de succès, `OLpomdp` va maximiser la fréquence à laquelle des succès sont rencontrés, faisant un compromis entre probabilité de succès et exécutions courtes. Pour corriger ce phénomène, on peut modifier `OLpomdp` comme suit :

– effacer la trace d'éligibilité à chaque fois que l'on repart de l'état initial ; et

– ajouter des étapes de décision fictives afin que toutes les exécutions soient de même longueur ; et

– accumuler les récompenses pendant une exécution pour ne les consommmer qu'une fois un état terminal atteint.

En pratique, `OLpomdp` a principalement été utilisé sans effectuer ces corrections. Une raison à cela est que l'apprentissage est plus difficile quand les exécutions sont longues. On préfère ne pas ajouter des pas de décision fictifs si cela permet d'apprendre des politiques favorisant des exécutions courtes.

15.4.4. *Améliorations de FPG*

Parce qu'il est basé sur une méthode de gradient, FPG peut bénéficier de quelques améliorations.

D'abord, FPG souffre du fait que son exploration initiale est aléatoire, assimilable à un mouvement brownien. Prenons le célèbre monde des blocs [SLA 01] dans lequel des blocs numérotés doivent être empilés selon une configuration donnée. On observe que, si une politique aléatoire est suivie, la fréquence à laquelle la configuration but est rencontrée décroît de manière exponentielle quand le nombre de blocs considérés augmente. Il serait donc utile d'orienter les recherches de FPG. Deux approches envisageables sont :

– utiliser un estimateur de progrès : il s'agit de donner à chaque décision une récompense indiquant si l'on semble s'être rapproché ou éloigné du but ; une difficulté est d'estimer la distance au but ; mais un estimateur simple s'est déjà montré très efficace [BUF 06] ; et

– suivre les décisions d'une heuristique : au lieu de commencer avec une politique aléatoire, il est envisageable de bénéficier de règles de décisions réputées efficaces, telles que celles développées dans le domaine de la planification classique (déterministe).

D'autre part, les calculs effectués par `OLpomdp` restent en général assez coûteux. Or dans de nombreux problèmes de planification d'opérations, la récompense reçue est la plupart du temps nulle. Il est alors possible 1) de ne modifier le vecteur $\vec{\theta}$ que quand une récompense est présente et 2) de ne faire décroître la trace d'éligibilité liée à une opération que quand une récompense est reçue ou quand cette opération est utilisée.

15.5. Expérimentations

Ces expérimentations, reprises de [ABE 07a], comparent MOP, Prottle et FPG. Nous présentons des résultats selon trois critères : la probabilité d'atteindre un état but, la longueur moyenne d'exécution (que le résultat soit un succès ou un échec), et la récompense moyenne à long terme (pour FPG). Les problèmes considérés sont :

– *Probabilistic Machine Shop (MS)* (un problème de gestion de machines dans un atelier) [MAU 05],

– *Maze (MZ)* (un labyrinthe),

– *Teleport (TP)* (un scénario de science-fiction dans lequel la téléportation lente est plus sûre que la rapide) [LIT 05], et

– *PitStop* (une course de voitures avec gestion des arrêts au stand et durées d'action incertaines) [ABE 07a]

Pour les trois premiers problèmes, on utilise ici les versions données dans [LIT 05]. Les expérimentations utilisent : FPG avec des réseaux linéaires, MOP, Prottle, une

politique aléatoire déclenchant les actions au hasard et une politique naïve qui tente d'exécuter *toutes* les actions éligibles. Ces deux derniers algorithmes permettent de vérifier qu'une optimisation est nécessaire pour obtenir de bons résultats.

Toutes les expérimentations ont une durée maximale de 600 secondes. D'autres paramètres sont décrits dans le tableau 15.2. En particulier, le pas de gradient constant α a été choisi comme la plus grande valeur garantissant une convergence sûre sur 100 exécutions sur tous les domaines. Les expérimentations ont été effectuées sur un Pentium IV 2.4GHz avec 1Go de mémoire vive. Les résultats sont résumés dans le tableau 15.1. A part pour Prottle, les probabilités d'échec et longueurs moyennes d'exécution ont été estimées à partir de 100 000 exécutions simulées du plan optimisé. Les résultats de Prottle viennent de [LIT 05], en citant les plus petits résultats de probabilité d'échec. Les expérimentations avec FPG et MOP ont été répétées 100 fois pour tenir compte de la nature stochastique de l'optimisation. Les expérimentations répétées avec FPG sont importantes pour mesurer l'effet des minima locaux. Pour FPG et MOP sont présentés les résultats moyens sur 100 optimisations et, entre parenthèses, la meilleure optimisation parmi les 100 (en prenant la probabilité d'échec comme critère). Les petites différences entre résultats moyens et meilleurs résultats indiquent que les optima locaux n'ont pas été trop sévères.

En général, le tableau 15.1 montre que FPG est au moins comparable avec Prottle et MOP, et meilleur sur le problème le plus difficile : *Machine Shop*. Les mauvaises performances de Prottle dans le problème *Teleport* — 79,8% d'échec par rapport aux 34,4% de FPG — viennent de ce qu'il considère ici des longueurs d'exécution de 20 unités de temps au plus.

Le tableau 15.1 montre que Prottle obtient de bons résultats *plus vite* sur *Maze* et *Machine Shop*. L'optimisation apparemment plus lente chez FPG ou MOP est due à leur convergence asymptotique. Pour FPG, le critère est d'*optimiser jusqu'à ce que la récompense moyenne à long terme ne s'améliore plus pendant 5 estimations de suite (de 10 000 pas chacune)*. En pratique, de bonnes politiques sont trouvées bien avant la convergence de ce critère [ABE 07a]. Les résultats expérimentaux pour le problème à temps continu *PitStop* montre la capacité d'optimisation de FPG dans un cadre où les variables aléatoires sont aussi bien discrètes que continues.

15.6. Conclusion et perspectives

Ce chapitre a présenté une application particulière des processus de décision markoviens : la planification d'opérations. Dans ces problèmes, les espaces d'état et d'action (une action étant un ensemble d'opérations) sont très structurés. On cherche donc à exploiter cette structure pour pallier l'explosion combinatoire de la taille de ces espaces. Les algorithmes présentés passent en général par une évaluation de la fonction de valeur, l'un d'entre eux (FPG) préférant à cela une optimisation par méthode de gradient. Ces algorithmes sont rendus plus efficaces en exploitant des méthodes classiques telles que :

Prob.	*Opt.*	*% échec*	*LP*	*R*	*Temps*
MS	FPG	1,33 (0,02)	6,6 (5,5)	118 (166)	532 (600)
MS	FPG	0,02	5,5	166	600
MS	Prottle	2,9			272
MS	MOP		débordement mémoire		
MS	aléatoire	99,3	18	0,1	
MS	naïve	100	20	0,0	
MZ	FPG	19,1 (14,7)	5,5 (6,9)	134 (130)	371 (440)
MZ	FPG	14,7	6,9	130	440
MZ	Prottle	17,8			10
MZ	MOP	7,92 (7,15)	8,0 (8,2)		71 (72)
MZ	MOP	7,15	8,2		72
MZ	aléatoire	76,5	13	16,4	
MZ	naïve	90,8	16	8,6	
TP	FPG	34,4 (33,3)	18 (18)	298 (305)	340 (600)
TP	FPG	33,3	18	305	600
TP	Prottle	79,8			442
TP	MOP		débordement mémoire		
TP	aléatoire	99,6	15	1,0	
TP	naïve	100	19	0,0	
PitStop	FPG	0,0	20180	142	41
PitStop	aléatoire	29,0	12649	41,0	
PitStop	naïve	100	66776	0,0	

Tableau 15.1. *Résultats sur 3 domaines d'essai. Les expérimentations pour MOP et FPG ont été répétées 100 fois. La colonne* Opt. *donne le moteur d'optimisation utilisé. %* échec=*pourcentage d'exécutions échouées,* LP=*longueur du plan,* R *est la récompense moyenne à long terme, et* Temps *est le temps d'optimisation en secondes.*

Paramètre	*Valeur*	Opt.
$\vec{\theta}_{init}$	0	FPG
α	1×10^{-5}	FPG
β	0,95	FPG
ϵ	1	MOP
ϵ	0,0 à 0,6	Prottle

Tableau 15.2. *Réglages de paramètres non discutés dans le texte.*

– utiliser une heuristique réputée indiquer souvent la bonne direction à suivre ;

– résoudre d'abord un problème simplifié avant de résoudre le problème original (on parle de relaxation de contraintes) ; ou

– restreindre l'espace des solutions explorées (au risque de perdre en optimalité).

Planification probabiliste non-temporelle non-concurrente

Les travaux présentés ici sont tous récents parce qu'ils sont parmi les premiers à s'attaquer au problème difficile de la planification d'opérations. Mais ils ont été précédés par des recherches dans des cas plus simples de planification probabiliste, cas dans lesquels les opérations ne peuvent être concurrentes et le temps n'apparaît pas. Un bon endroit pour trouver des références sur ce sujet est la compétition internationale de planification (IPC), au cours de laquelle une catégorie « planification probabiliste » a déjà été organisée en 2004 et 2006.

Bibliographie

[ABE 02] ABERDEEN D., BAXTER J., « Scaling Internal-State Policy-Gradient Methods for POMDPs », *Proceedings of the Nineteenth International Conference on Machine Learning (ICML'02)*, July 2002.

[ABE 03] ABERDEEN D., Policy-Gradient Algorithms for Partially Observable Markov Decision Processes, PhD thesis, The Australian National University, Canberra, Australia, March 2003.

[ABE 04] ABERDEEN D., THIÉBAUX S., ZHANG L., « Decision-Theoretic Military Operations Planning », *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS'04)*, June 2004.

[ABE 05] ABERDEEN D., « Policy-Gradient Methods for Planning », *Advances in Neural Information Processing Systems 19 (NIPS'05)*, 2005.

[ABE 07a] ABERDEEN D., BUFFET O., « Temporal Probabilistic Planning with Policy-Gradients », *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS'07)*, September 2007.

[ABE 07b] ABERDEEN D., BUFFET O., THOMAS O., « Policy-Gradient for PSRs and POMDPs », *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS'07)*, 2007.

[ADD 05] ADDA C., LAURENT G. J., LE FORT-PIAT N., « Learning to control a real micropositioning system in the STM-Q framework », *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'05)*, Barcelone, Spain, p. 4580–4585, April 18–22 2005.

[AMA 98] ICHI AMARI S., « Natural Gradient Works Efficiently in Learning », *Neural Computation*, vol. 10, n°2, p. 251–276, February 1998.

[ANO 07] ANONYMOUS, « IUCN cat projects database », October 2007, Zoological Society of London, (ZSL).

[ANT 07] ANTOS A., SZEPESVÁRI C., MUNOS R., « Learning Near-Optimal Policies with Bellman-Residual Minimization Based Fitted Policy Iteration and a Single Sample Path », *To appear in Machine Learning Journal*, 2007.

[ARA 07] ARAS R., DUTECH A., CHARPILLET F., « Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs », *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'07)*, 2007.

[AST 65] ASTRÖM K., « Optimal control of markov decision processes with incomplete state estimation », *Journal of Mathematical Analysis and Applications*, vol. 10, p. 174–205, 1965.

[ATK 97] ATKESON C. G., MOORE A. W., SCHAAL S. A., « Locally Weighted Learning », *AI Review*, vol. 11, 1997.

[AUE 95] AUER P., CESA-BIANCHI N., FREUND Y., SCHAPIRE R., « Gambling in a Rigged Casino : The Adversarial Multi-Armed Bandit Problem », *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, p. 322–331, 1995.

[AUM 02] AUMANN R. J., HART S., Eds., *Handbook of Game Theory with Economic Applications*, Elsevier Science, volume 3, 2002.

[BAG 01] BAGNELL J., NG A. Y., SCHNEIDER J., Solving Uncertain Markov Decision Problems, Rapport n°CMU-RI-TR-01-25, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2001.

[BAG 03] BAGNELL J., SCHNEIDER J., « Covariant Policy Search », *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003.

[BAH 93] BAHAR R., FROHM E., GAONA C., HACHTEL G., MACII E., PARDO A., SOMENZI F., « Algebraic Decision Diagrams and their Applications », *IEEE/ACM International Conference on CAD*, Santa Clara, California, p. 188–191, 1993.

[BAI 93] BAIRD L., Advantage Updating, Rapport n°WL-TR-93-1146, Wright-Patterson Air Force Base Ohio : Wright Laboratory, 1993.

[BAI 95] BAIRD L. C., « Residual Algorithms : Reinforcement Learning with Function Approximation », *Proceedings of the Twelfth International Conference in Machine Learning (ICML'95)*, San Francisco, CA, Morgan Kaufman Publishers, 1995.

[BAI 99a] BAIRD L., Reinforcement Learning Through Gradient Descent, PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, 1999.

[BAI 99b] BAIRD L., MOORE A., « Gradient Descent for General Reinforcement Learning », *Advances in Neural Information Processing Systems 11 (NIPS'99)*, The MIT Press, 1999.

[BAR 83] BARTO A., SUTTON R., ANDERSON C. W., « Neuron-like Adaptive Elements That Can Solve Difficult Learning Control Problems », *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, n°5, p. 834–846, 1983.

[BAR 95] BARTO A., BRADTKE S., SINGH S., « Learning to Act Using Real-time Dynamic Programming », *Artificial Intelligence*, vol. 72, p. 81–138, 1995.

[BAR 06] BARDOS D. C., DAY R. W., LAWSON N. T., LINACRE N. A., « Dynamical Response to Fishing Varies with Compensatory Mechanism : An Abalone Population Model », *Ecological Modelling*, vol. 192, n°3-4, p. 523–542, 2006.

[BAU 98] BAUR C., BUGACOV A., KOEL B., MADHUKAR A., MONTOYA N., RAMACHANDRAN T., REQUICHA A., RESCH R., WILL P., « Nanoparticle Manipulation by Mechanical Pushing : Underlying Phenomena and Real-Time Monitoring », *Nanotechnology*, vol. 9, p. 360–364, 1998.

[BAX 00] BAXTER J., BARTLETT P., « Reinforcement Learning in POMDP's via Direct Gradient Ascent », *In Proceedings of the 17th International Conference on Machine Learning (ICML'00)*, 2000.

[BAX 01a] BAXTER J., BARTLETT P., « Infinite-Horizon Policy-Gradient Estimation », *Journal of Artificial Intelligence Research*, vol. 15, p. 319–350, 2001.

[BAX 01b] BAXTER J., BARTLETT P., WEAVER L., « Experiments with Infinite-Horizon, Policy-Gradient Estimation », *Journal of Artificial Intelligence Research*, vol. 15, p. 351–381, 2001.

[BEC 03] BECKER R., ZILBERSTEIN S., LESSER V., GOLDMAN C., « Transition-Independent Decentralized Markov Decision Processes », *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'03)*, Melbourne, Australia, ACM Press, p. 41-48, July 2003.

[BEC 04a] BECKER R., LESSER V., ZILBERSTEIN S., « Decentralized Markov Decision Processes with Event-Driven Interactions », *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'04)*, New-York, USA, p. 302-309, 2004.

[BEC 04b] BECKER R., ZILBERSTEIN S., LESSER V., GOLDMAN C., « Solving Transition Independent Decentralized Markov Decision Processes », *Journal of Artificial Intelligence Research*, vol. 22, p. 423-455, Morgan Kaufmann Publishers, December 2004.

[BEL 57] BELLMAN R. E., *Dynamic Programming*, Princeton University Press, 1957.

[BEL 59] BELLMAN R., DREYFUS S., « Functional Approximation and Dynamic Programming », *Math. Tables and other Aids Comp.*, vol. 13, p. 247–251, 1959.

[BEL 63] BELLMAN R., KALABA R., KOTKIN B., « Polynomial Approximation - a New Computational Technique in Dynamic Programming », *Math. Comp.*, vol. 17, n°8, p. 155–161, 1963.

[BEL 04] BELLOSTA M., BRIGUI I., KORNMAN S., PINSON S., VANDERPOOTEN D., « Un mécanisme de négociation multicritère pour le commerce électronique », *Reconnaissance des Formes et Intelligence Artificielle (RFIA'04)*, Toulouse, p. 1009–1016, Janvier 2004.

[BEN 02] BENFERHAT S., DUBOIS D., GARCIA L., PRADE H., « On the Transformation between Possibilistic Logic Bases and Possibilistic Causal Networks », *International Journal of Approximate Reasoning*, vol. 29, p. 135–173, 2002.

[BER 87] BERTSEKAS D. P., *Dynamic Programming : Deterministic and Stochastic Models*, Prentice-Hall, 1987.

[BER 89] BERTSEKAS D. P., TSITSIKLIS J. N., *Parallel and Distributed Computation : Numerical Methods*, Prentice-Hall, 1989.

[BER 95] BERTSEKAS D., *Dynamic programming and optimal control*, Athena Scientific, Belmont, MA, 1995.

[BER 96] BERTSEKAS D., TSITSIKLIS J., *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.

[BER 02] BERNSTEIN D., GIVAN R., IMMERMAN N., ZILBERSTEIN S., « The Complexity of Decentralized Control of Markov Decision Processes », *Mathematics of Operations Research*, vol. 27, n°4, p. 819–840, JSTOR, 2002.

[BER 05] BERNSTEIN D., HANSEN E.A., ZILBERSTEIN S., « Bounded Policy Iteration for Decentralized POMDPs », *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland, 2005.

[BEY 04] BEYNIER A., MOUADDIB A.I., « A Decentralized MultiAgent Decision Approach for Handling Temporal and Resource Constraints : preliminary report », *Proceedings of the AAAI Symposium on Bridging the Multi-Agent and Multi-Robotic Research Gap*, 2004.

[BEY 05] BEYNIER A., MOUADDIB A.I., « A polynomial algorithm for Decentralized Markov Decision Processes with temporal constraints », *Proceedings of the Fourth International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS'05)*, p. 963-969, 2005.

[BEY 06] BEYNIER A., MOUADDIB A.I., « An iterative algorithm for solving Constrained Decentralized Markov Decision Processes », *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI'06)*, 2006.

[BLU 97] BLUM A., FURST M., « Fast Planning Through Planning Graph Analysis », *Artificial Intelligence*, vol. 90, p. 281–300, 1997.

[BON 00] BONET B., GEFFNER H., « Planning with Incomplete Information as Heuristic Search in Belief Space », *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, 2000.

[BON 02] BONET B., PEARL J., « Qualitative MDPs and POMDPs : An order-of-magnitude approximation », *Proceedings of the Annual Conference on Uncertainty in Artificial Intelligence (UAI'02)*, vol. 18, p. 61-68, 2002.

[BON 03] BONET B., GEFFNER H., « Labeled RTDP : Improving the Convergence of Real-Time Dynamic Programming », *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS'03)*, 2003.

[BOU 92] BOUCHERON S., *Théorie de l'Apprentissage : de l'approche formelle aux enjeux cognitifs*, Hermès, 1992.

[BOU 94] BOUTILIER C., « Toward a logic for qualitative decision theory », *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, Bonn, Allemagne, p. 75–86, 1994.

[BOU 95] BOUTILIER C., DEARDEN R., GOLDSZMIDT M., « Exploiting Structure in Policy Construction », *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, Montreal, p. 1104–1111, 1995.

[BOU 96a] BOUTILIER C., « Planning, Learning and Coordination in Multiagent Decision Processes », *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'96)*, p. 195-201, 1996.

[BOU 96b] BOUTILIER C., GOLDSZMIDT M., « The Frame Problem and Bayesian Network Action Representations », *Proceedings of the Eleventh Biennial Canadian Conference on Artificial Intelligence (AI '96)*, Toronto, CA, p. 69–83, 1996.

[BOU 99a] BOUTILIER C., DEAN T., HANKS S., « Decision-Theoretic Planning : Structural Assumptions and Computational Leverage », *Journal of Artificial Intelligence Research (JAIR)*, vol. 11, p. 1—94, 1999.

[BOU 99b] BOUTILIER C., T. D., HANKS S., « Decision-Theoretic Planning : Structural Assumptions and Computational Leverage », *Journal of Artificial Intelligence Research*, vol. 11, p. 1–94, 1999.

[BOU 99c] BOUTILIER G., « Sequential Optimality and Coordination in MultiAgent Systems », *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, p. 478-485, 1999.

[BOU 00a] BOUTILIER C., DEARDEN R., GOLDSZMIDT M., « Stochastic Dynamic Programming with Factored Representations », *Artificial Intelligence*, vol. 121, n°1, p. 49–107, 2000.

[BOU 00b] BOUYSSOU D., MARCHANT T., PERNY P., PIRLOT M., TSOUKIÀS A., VINCKE P., *Evaluation and decision models : a critical perspective*, Kluwer, 2000.

[BOU 07] BOUSSARD M., BOUZID M., MOUADDIB A.-I., « Multi-Criteria Decision Making for local Coordination in Multi-Agent Systems », *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI'07)*, Octobre 2007.

[BOW 02a] BOWLING M., VELOSO M., « Multiagent Learning using a Variable Learning Rate », *Artificial Intelligence*, vol. 136, n°2, p. 215–250, 2002.

[BOW 02b] BOWLING M., VELOSO M., « Scalable Learning in Stochastic Games », *AAAI Workshop on Game Theoretic and Decision Theoretic Agents*, 2002.

[BOW 03a] BOWLING M., Multiagent Learning in the Presence of Agents with Limitations, PhD thesis, University of Toronto, 2003.

[BOW 03b] BOWLING M., VELOSO M., « Simultaneous Adversarial Multirobot Learning », *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003.

[BOW 04] BOWLING M., « Convergence and No-Regret in Multiagent Learning », *Advances in Neural Information Processing Systems 17 (NIPS'04)*, p. 209–216, 2004.

[BOY 99] BOYAN J., « Least-Squares Temporal Difference Learning », *Proceedings of the 16th International Conference on Machine Learning (ICML'99)*, p. 49-56, 1999.

[BRA 96] BRADTKE S., BARTO A., « Linear Least-Squares Algorithms for Temporal Difference Learning », *Journal of Machine Learning*, vol. 22, p. 33-57, 1996.

[BRA 01] BRAFMAN R. I., TENNENHOLTZ M., « R-max : a General Polynomial Time Algorithm for Near-optimal Reinforcement Learning », *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, p. 953–958, 2001.

[BRA 03] BRAFMAN R. I., TENNENHOLTZ M., « Learning to Coordinate Efficiently : A Model Based Approach », *Journal of Artificial Intelligence Research*, vol. 19, p. 11–23, 2003.

[BRE 02] BRESINA J., DEARDEN R., MEULEAU N., RAMAKRISHNAN S., SMITH D., WASHINGTON R., « Planning Under Continuous Time and Resource Uncertainty : A Challenge for AI », *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI'02)*, 2002.

[BRO 51] BROWN G., « Iterative solution of games by fictitious play », *Activity Analysis of Production and Allocation*, vol. 13, p. 374–376, Wiley, 1951.

[BRY 86] BRYANT R. E., « Graph-Based Algorithms for Boolean Function Manipulation », *IEEE Transactions on Computers*, vol. C-35, n°8, p. 677–691, 1986.

[BRY 06] BRYCE D., KAMBHAMPATI S., « A Tutorial on Planning Graph Based Reachability Heuristics », *AI Magazine*, vol. 27, n°4, 2006.

[BUF 05] BUFFET O., ABERDEEN D., « A Two-Teams Approach for Robust Probabilistic Temporal Planning », *Proceedings of the ECML'05 workshop on Reinforcement Learning in Non-Stationary Environments*, 2005.

[BUF 06] BUFFET O., ABERDEEN D., « The Factored Policy Gradient planner (IPC'06 Version) », *Proceedings of the Fifth International Planning Competition (IPC-5)*, 2006.

[BUR 07] BURKOV A., CHAIB-DRAA B., « Multiagent Learning in Adaptive Dynamic Systems », *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'07)*, Honolulu, Hawai'i, 2007.

[CAR 01] CARDON S., MOUADDIB A. I., ZILBERSTEIN S., WASHINGTON R., « Adaptive Control of Acyclic Progressive Processing task Structures », *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, WA, USA, p. 701–706, 2001.

[CAS 94] CASSANDRA A., KAELBLING L., LITTMAN M., « Acting Optimally in Partially Observable Stochastic Domains », *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, Seattle, WA, 1994.

[CAS 98] CASSANDRA A., Exact and Approximate Algorithms for Partially Observable Markov Decision Processes, PhD thesis, Brown University, 1998.

[CAS 05] CASSANDRA A., « pomdp-solve : POMDP solver software », 2005.

[CHA 01] CHANG Y., KAELBLING L., « Playing is Believing : The Role of Beliefs in Multi-Agent Learning », *Advances in Neural Information Processing Systems (NIPS'01)*, Canada, 2001.

[CHA 02] CHADÈS I., SCHERRER B., CHARPILLET F., « A Heuristic Approach for Solving Decentralized-POMDP : Assessment on the Pursuit Problem », *Proceedings of the Sixteenth ACM Symposium on Applied Computing (SAC'02)*, 2002.

[CHA 05] CHANTHERY E., BARBIER M., FARGES J.-L., « Planning Algorithms for an Autonomous Aerial Vehicle », *16th IFAC World Congress*, Prague, République Tchèque, 2005.

[CHA 08] CHADES I., MCDONALD-MADDEN E., MCCARTHY M., LINKIE M., POSSINGHAM H., « Save, Survey or Surrender : Optimal Management of Threatened Species », *Submitted*, 2008.

[CHE 88] CHENG H.-T., Algorithms for Partially Observable Markov Decision Processes, PhD thesis, University of British Columbia, Canada, 1988.

[CHU 03] CHU F., HALPERN J., « Great Expectations. Part I : On the Customizability of Generalized Expected Utility », *Proceedings of the Eighteenth International Joint Conference in Artificial Intelligence (IJCAI'03)*, p. 291–296, 2003.

[CIC 95] CICHOSZ P., « Truncating Temporal Differences : On the Efficient Implementation of TD(λ) for Reinforcement Learning », *Journal of Artificial Intelligence Research*, vol. 2, p. 287–318, 1995.

[CIT 07] CITES, « Appendices I, II, and III. », August 2007.

[CLA 98] CLAUS C., BOUTILIER C., « The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems », *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, Menlo Park, CA, AAAI Press, p. 746–752, 1998.

[COU 02] COULOM R., Reinforcement Learning using Neural Networks, with Applications to Motor Control, PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[CRI 97] CRITES R., BARTO A., « Improving Elevator Performance using Reinforcement Learning », *Advances in Neural Information Processing Systems 9 (NIPS'97)*, 1997.

[CUS 07] CUSHING W., KAMBHAMPATI S., MAUSAM, WELD D. S., « When is Temporal Planning Really Temporal ? », *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007.

[DAM 05] DAMIANI S., VERFAILLIE G., CHARMEAU M.-C., « A Continuous Anytime Planning Module for an Autonomous Earth Watching Satellite », *ICAPS'05 Workshop on Planning and Scheduling for Autonomous Systems*, Monterey, CA, USA, p. 19–28, 2005.

[DAR 92] DARWICHE A., GINSBERG M., « A Symbolic Generalization of Probability Theory », *Proceedings of the National Conference on Artificial Intelligence (AAAI'92)*, p. 622–627, 1992.

[DAR 94] DARWICHE A., GOLDSZMIDT M., « On the Relation Between Kappa Calculus and Probabilistic Reasoning », *Proceedings of the Tenth Conference on Uncertainty in Artificial International (UAI'94)*, Seattle, WA, Morgan Kaufmann, p. 145–153, 1994.

[DAV 97] DAVIES G., MALLAT S., AVELLANEDA M., « Adaptive Greedy Approximations », *Journal of Constructive Approximation*, vol. 13, p. 57–98, 1997.

[DAY 94] DAYAN P., SEJNOWSKI T. J., « TD(λ) Converges with Probability 1 », *Machine Learning*, vol. 14, n°3, p. 295–301, 1994.

[D'E 63] D'EPENOUX F., « A probabilistic production and inventory problem », *Management Science*, vol. 10, p. 98–108, 1963.

[DEA 89] DEAN T., KANAZAWA K., « A Model for Reasoning about Persistence and Causation », *Computational Intelligence*, vol. 5, p. 142–150, 1989.

[DEA 98] DEARDEN R., FRIEDMAN N., RUSSELL S., « Bayesian Q-learning », *Proceedings of the National Conference on Artificial Intelligence (AAAI'98)*, 1998.

[DEA 03] DEARDEN R., MEULEAU N., RAMAKRISHNAN S., SMITH D., WASHINGTON R., « Incremental Contingency Planning », *Proceedings of the ICAPS-03 Workshop on Planning under Uncertainty*, 2003.

[DEC 93] DECKER K., LESSER V., « Quanititative Modeling of Complex Computational Task Environnement », *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, p. 217–224, January 1993.

[DEG 06] DEGRIS T., SIGAUD O., WUILLEMIN P.-H., « Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems », *Proceedings of the International Conference on Machine Learning (ICML'06)*, 2006.

[DEG 07] DEGRIS T., Apprentissage par renforcement dans les processus de décision markoviens factorisés, PhD thesis, Université Pierre et Marie Curie - Paris 6, 2007.

[DEM 67] DEMPSTER A., « Upper and Lower Probabilities Induced by a Multivalued Mapping », *Annals of Mathematical Statistics*, vol. 38, p. 325-339, 1967.

[DEV 97] DEVORE R., *Nonlinear Approximation*, Acta Numerica, 1997.

[DUB 88] DUBOIS D., PRADE H., *Possibility theory*, Plenum Press, 1988.

[DUB 93] DUBOIS D., PRADE H., SANDRI S., « *Fuzzy Logic : State of the Art* », Chapitre On Possibility/Probability Transformations, p. 103–112, Kluwer Academic Publishers, 1993.

[DUB 94] DUBOIS D., PRADE H., « A Survey of Belief Revision and Updating Rules in Various Uncertainty Models », *International Journal of Intelligent Systems*, vol. 9, p. 61–100, 1994.

[DUB 95] DUBOIS D., PRADE H., « Possibility Theory as a Basis for Qualitative Decision Theory », *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, Montreal, Canada, p. 1925–1930, 20-25 août 1995.

[DUB 96] DUBOIS D., FARGIER H., PRADE H., « Possibility Theory in Constraint Satisfaction Problems », *Applied Intelligence*, vol. 6, n°4, p. 287–309, 1996.

[DUB 98] DUBOIS D., PRADE H., SABBADIN R., « Qualitative Decision Theory with Sugeno Integrals », COOPER G. F., MORAL S., Eds., *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI'98)*, Madison, WI, Morgan Kaufmann, p. 121–128, 24-26 Juillet 1998.

[DUT 00] DUTECH A., « Solving POMDP using Selected Past-Events », *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI'00)*, 2000.

[DUT 01] DUTECH A., BUFFET O., CHARPILLET F., « Multi-Agent Systems by Incremental Gradient Reinforcement Learning », *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, 2001.

[DUT 06] DUTECH A., ARAS R., CHARPILLET F., « Apprentissage par renforcement et théorie des jeux pour la coordination des systèmes multi-agents », *Colloque africain pour la recherche en informatique (CARI'06)*, Cotonou, Bénin, 2006.

[EHR 03] EHRGOTT M., TENFELDE-PODEHL D., « Computation of Ideal and Nadir Values and Implications for their Use in MCDM Methods. », *European Journal of Operational Research*, vol. 151, n°1, p. 119–139, 2003.

[EME 04] EMERY-MONTEMERLO R., GORDON G., SCHNEIDER J., THRUN S., « Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs », *Proceedings of the Third Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS'04)*, 2004.

[ERK 98] ERKUT E., VERTER V., « Modeling of Transport Risk for Hazardous Materials », *Operations Research*, vol. 48, p. 624–642, 1998.

[ERN 05] ERNST D., GEURTS P., WEHENKEL L., « Tree-Based Batch Mode Reinforcement Learning », *Journal of Machine Learning Research*, vol. 6, p. 503–556, 2005.

[FAB 07] FABIANI P., FUERTES V., BESNERAIS G. L., MAMPEY R., PIQUEREAU A., TEICHTEIL F., « The ReSSAC Autonomous Helicopter : Flying in a Non-Cooperative Uncertain World with embedded Vision and Decision Making », *A.H.S. Forum*, 2007.

[FAR 98] FARGIER H., LANG J., SABBADIN R., « Towards Qualitative Approaches to Multi-Stage Decision Making », *International Journal of Approximate Reasoning*, vol. 19, p. 441–471, 1998.

[FAR 01] DE FARIAS D., VAN ROY B., « The Linear Programming Approach to Approximate Dynamic Programming », *Operations Research*, vol. 51, n°6, p. 850–856, 2001.

[FAR 03] FARGIER H., SABBADIN R., « Qualitative Decision under Uncertainty : Back to Expected Utility », *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, Acapulco, Mexique, p. 303–308, 9-15 Août 2003.

[FAR 05] FARGIER H., SABBADIN R., « Qualitative decision under uncertainty : back to expected utility », *Artificial Intelligence*, vol. 164, p. 245-280, 2005.

[FEN 02] FENG Z., HANSEN E., « Symbolic Heuristic Search for Factored Markov Decision Processes », *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, Edmonton, Alberta, Canada, p. 455–460, 2002.

[FEN 03] FENG Z., HANSEN E., ZILBERSTEIN S., « Symbolic Generalization for On-line Planning », *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI'03)*, Morgan Kaufmann, 2003.

[FER 95] FERBER J., *Les Systèmes multi-agents : Vers une intelligence collective*, Inter Editions, 1995.

[FIN 64] FINK A., « Equilibrium in a Stochastic n-Person Game », *Journal of Science in Hiroshima University Series*, vol. 28, p. 89–93, 1964.

[FOX 03] FOX M., LONG D., « PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains », *Journal of Artificial Intelligence Research*, vol. 20, p. 61–124, 2003.

[FRE 99] FREUND Y., SCHAPIRE R., « Adaptive Game Playing using Multiplicative Weights », *Games and Economic Behavior*, vol. 29, n°79–103, page336, 1999.

[FRI 95] FRIEDMAN N., HALPERN J., « Plausibility Measures : A User's Guide », *Proceedings of the Eleventh International Conference on Uncertainty in Artificial Intelligence (UAI'95)*, p. 175–184, 1995.

[FUD 91] FUDENBERG D., TIROLE J., *Game Theory*, Mit Press, 1991.

[FUD 99] FUDENBERG D., LEVINE D. K., *The Theory of Learning in Games*, MIT Press,, 1999.

[GAL 06] GALAND L., PERNY P., « Search for Compromise Solutions in Multiobjective State Space Graphs », *Proceedings of the Seventeenth European Conference on Artificial Intelligence*, p. 93–97, 2006.

[GAR 98] GARCIA F., NDIAYE S., « A Learning Rate Analysis of Reinforcement-Learning Algorithms in Finite-Horizon », *Proceedings of the Fifteenth International Conference on Machine Learning (ICML'98)*, Madison, USA, Morgan Kaufmann, p. 215–223, 1998.

[GAR 06] GARCIA L., SABBADIN R., « Possibilistic Influence Diagrams », *Proceedings of the European Conference on Artificial Intelligence (ECAI'06)*, p. 372–376, 2006.

[GAR 07] GARCIA L., SABBADIN R., « Diagrammes d'influence possibilistes », *Revue d'Intelligence Artificielle*, vol. 21, n°4, p. 521–554, 2007.

[GAR 08] GARCIA L., SABBADIN R., « Complexity results and algorithms for possibilistic influence diagrams », *Artificial Intelligence*, page27 pages, 2008, To appear.

[GEF 98] GEFFNER H., BONET B., « Solving Large POMDPs by Real Time Dynamic Programming », *Working Notes Fall AAAI Symposium on POMDPs*, 1998.

[GEI 01] GEIBEL P., « Reinforcement Learning with Bounded Risk », *Proceedings of the Eighteenth International Conference on Machine Learning (ICML'01)*, 2001.

[GEN 00] GENTIS H., Ed., *Game Theory Evolving : A Problem-Centered Introduction to Modeling Strategic Interaction*, Princeton University Press, 2000.

[GER 04] GERBER L. R., KATE E. B., GLENN V., « Density Dependence and Risk of Extinction in a Small Population of Sea Otters », *Biodiversity and Conservation*, vol. 13, p. 2741–2757, 2004.

[GHA 04] GHALLAB M., NAU D., TRAVERSO P., *Automated Planning : Theory and Practice*, Morgan Kauffmann Publishers, 2004.

[GIA 99] GIANG P. H., SHENOY P. P., « On Transformations between Probability and Spohnian Disbelief Functions », PRADE H., Ed., *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99)*, Stockholm, Sweden, Morgan Kaufmann, p. 236–244, 1999.

[GIA 01] GIANG P., SHENOY P., « A Comparison of Axiomatic Approaches to Qualitative Decision Making Using Possibility Theory », *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI'01)*, vol. 17, p. 162–170, 2001.

[GIE 06] GIES O., CHAIB-DRAA B., « Apprentissage de la coordination multiagent : une méthode basée sur le Q-learning par jeu adaptatif », *Revue d'Intelligence Artificielle*, vol. 20, n°2-3, p. 385–412, 2006.

[GLO 07] GLOANNEC S. L., Contrôle adaptatif d'un agent rationnel à ressources limitées dans un environnement dynamique et incertain, PhD thesis, Univ. Caen Basse Normandie, 2007.

[GLY 89] GLYNN P., IGLEHART D., « Importance Sampling for Stochastic Simulations », *Management Science*, vol. 35, n°11, p. 1367–1392, 1989.

[GOL 03] GOLDMAN C., ZILBERSTEIN S., « Optimizing Information Exchange in Cooperative MultiAgent Systems », *International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'03)*, p. 137–144, 2003.

[GOL 04] GOLDMAN C., ZILBERSTEIN S., « Decentralized Control of Cooperative Systems : Categorization and Complexity Analysis », *Journal of Artificial Intelligence Research*, vol. 22, p. 143–174, 2004.

[GON 01] GONDRAN M., MINOUX M., *Graphes, dioides et semi-anneaux*, Editions Technique et Documentation, 2001.

[GOR 95] GORDON G., « Stable Function Approximation in Dynamic Programming », PRIEDITIS A., RUSSELL S., Eds., *Proceedings of the Twelfth International Conference on Machine Learning (ICML'95)*, San Francisco, CA, Morgan Kaufmann, p. 261–268, 1995.

[GOS 04] GOSAVI A., « A Reinforcement Learning Algorithm Based on Policy Iteration for Average Reward : Empirical Results with Yield Management and Convergence Analysis », *Machine Learning*, vol. 55, p. 5–29, 2004.

[GRA 02a] GRABISCH M., PERNY P., « Agrégation multicritère », *Logique floue, principes, aide à la décision*, p. 81–120, Hermes, 2002.

[GRÄ 02b] GRÄDEL E., THOMAS W., WILKE T., Eds., *Automata, Logics and Infinite Games*, Springer-Verlag, vol. 2500 of LNCS, 2002.

[GRE 01] GREENSMITH E., BARTLETT P., BAXTER J., « Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning », *Advances in Neural Information Processing Systems 14 (NIPS'01)*, 2001.

[GRO 91] GROSOF B., « Generalizing prioritization », *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, p. 289–300, 1991.

[GUE 01a] GUESTRIN C., KOLLER D., PARR R., « Max-norm Projections for Factored MDPs », *Proceedings of the SevenTeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, p. 673–680, 2001.

[GUE 01b] GUESTRIN C., KOLLER D., PARR R., « Solving factored POMDPs with linear value functions », *Proceedings of the IJCAI-01 Workshop on Planning under Uncertainty and Incomplete Information*, Seattle, WA, 2001.

[GUE 03a] GUESTRIN C., Planning Under Uncertainty in Complex Structured Environments, PhD thesis, Computer Science Department, Stanford University, USA, 2003.

[GUE 03b] GUESTRIN C., KOLLER D., PARR R., VENKATARAMAN S., « Efficient Solution Algorithms for Factored MDPs », *Journal of Artificial Intelligence Research*, vol. 19, p. 399–468, 2003.

[GUE 04] GUESTRIN C., HAUSKRECHT M., KVETON B., « Solving Factored MDPs with Continuous and Discrete Variables », *Proceedings Proceedings of the Twentieth Annual Conference on Uncertainty in Artificial Intelligence (UAI'04)*, Banff, Canada, 2004.

[GYÖ 02] GYÖRFI L., KOHLER M., KRZYŻAK A., WALK H., *A Distribution-Free Theory of Nonparametric Regression*, Springer-Verlag, 2002.

[HAL 01] HALPERN J., « Conditional Plausibility Measures and Bayesian Networks », *Journal of Artificial Intelligence Research*, vol. 14, p. 359–389, 2001.

[HAN 98a] HANSEN E., Finite-Memory Control of Partially Observable Systems, PhD thesis, Dept. of Computer Science, University of Massachussets at Amherst, 1998.

[HAN 98b] HANSEN E., « An Improved Policy Iteration Algorithm for Partially Observable MDPs », *Advances in Neural Information Processing Systems 10 (NIPS)*, 1998.

[HAN 98c] HANSEN T., KÜHLE A., SORENSEN A., BOHR J., LINDELOF P., « A Technique for Positioning Nanoparticles using an Atomic Force Microscope », *Nanotechnology*, vol. 9, p. 337–342, 1998.

[HAN 01] HANSEN E., ZILBERSTEIN S., « LAO* : A Heuristic Search Algorithm that Finds Solutions with Loops », *Artificial Intelligence*, vol. 129, p. 35–62, 2001.

[HAN 04] HANSEN E., BERNSTEIN D., ZILBERSTEIN S., « Dynamic Programming for Partially Observable Stochastic Games », *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI'04)*, p. 709–715, 2004.

[HAR 00] HART S., MAS-COLELL A., « A Simple Adaptive Procedure Leading to Correlated Equilibrium », *Econometrica*, vol. 68, n°5, p. 1127–1150, Blackwell Synergy, 2000.

[HAS 01] HASTIE T., TIBSHIRANI R., FRIEDMAN J., *The Elements of Statistical Learning*, Springer Series in Statistics, 2001.

[HAU 95] HAUSSLER D., « Sphere Packing Numbers for Subsets of the Boolean n-Cube with Bounded Vapnik-Chervonenkis Dimension », *Journal of Combinatorial Theory Series A*, vol. 69, p. 217–232, 1995.

[HAU 98] HAURIE A., KRAWCZYK J. B., *An Introduction to Dynamic Games*, Faculty of Economics and Social Sciences, University of Geneva, Geneva, Switzerland, 1998, Handouts.

[HAU 00] HAUSKRECHT M., « Value-Function Approximations for Partially Observable Markov Decision Processes », *Journal of Artificial Intelligence Research*, vol. 13, p. 33–94, 2000.

[HEN 99] HENRION M., PROVAN G., DEL FAVEROL B., SANDERS G., « An Experimental Comparison of Numerical and Qualitative Probabilistic Reasoning », *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI'94)*, Seattle, WA, Morgan Kaufmann, p. 319–326, 1999.

[HIS 78] HISDAL E., « Conditional Possibilities–Independence and Non-Interactivity », *Fuzzy Sets and Systems*, vol. 1, p. 283–297, 1978.

[HOE 99] HOEY J., ST-AUBIN R., HU A., BOUTILIER C., « SPUDD : Stochastic Planning using Decision Diagrams », *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99)*, Morgan Kaufmann, p. 279–288, 1999.

[HOE 00] HOEY J., ST-AUBIN R., HU A., BOUTILIER C., Optimal and Approximate Stochastic Planning using Decision Diagrams, Rapport n°TR-00-05, University of British Columbia, 2000.

[HOF 66] HOFFMAN A., KARP R., « On Nonterminating Stochastic Games », *Management Science*, vol. 12, n°5, p. 359–370, JSTOR, 1966.

[HU 03] HU J., WELLMAN M., « Nash Q-learning for General-Sum Stochastic Games », *Journal of Machine Learning Research*, vol. 4, p. 1039–1069, MIT Press, 2003.

[JAA 94a] JAAKKOLA T., JORDAN M. I., SINGH S. P., « On the Convergence of Stochastic Iterative Dynamic Programming Algorithms », *Neural Computation*, vol. 6, p. 1185–1201, 1994.

[JAA 94b] JAAKKOLA T., SINGH S., JORDAN M., « Reinforcement learning algorithm for partially observable Markov decision problems. », *Advances in Neural Information Processing Systems 7 (NIPS'94)*, Cambridge, MA, MIT Press, 1994.

[JAL 89] JALALI A., FERGUSON M., « Computationally Efficient Adaptative Control Algorithms for Markov Chains », *Proceedings of the IEEE Conference on Decision and Control (CDC'89)*, vol. 28, p. 1283–1288, 1989.

[JON 06] JONSSON A., BARTO A., « Causal Graph Based Decomposition of Factored MDPs », *Journal of Machine Learning Research*, vol. 7, p. 2259–2301, 2006.

[JOZ 01] JOZEFOWIEZ J., *Conditionnement opérant et Problèmes décisionnels de Markov*, Thèse de doctorat de l'Université Lille III, Lille, 2001.

[JUD 98] JUDD K., *Numerical Methods in Economics*, MIT Press, 1998.

[JUN 02] JUNKER U., « Preference-Based Search and Multi-Criteria Optimization », *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, p. 34–40, 2002.

[KAE 93] KAELBLING L. P., *Learning in Embedded Systems*, MIT Press, Cambridge, MA, USA, 1993.

[KAE 96] KAELBLING L. P., LITTMAN M. L., MOORE A. W., « Reinforcement Learning : A Survey », *Journal of Artificial Intelligence Research*, vol. 4, p. 237–285, 1996.

[KAE 98] KAELBLING L., LITTMAN M., CASSANDRA A., « Planning and Acting in Partially Observable Stochastic Domains », *Artificial Intelligence*, vol. 101, p. 99–134, 1998.

[KAK 02] KAKADE S., « A Natural Policy Gradient », *Advances in Neural Information Processing Systems 14 (NIPS'02)*, 2002.

[KAK 03] KAKADE S., On the Sample Complexity of Reinforcement Learning, PhD thesis, Gatsby Computationel Neuroscience Unit, University College London, 2003.

[KEA 98] KEARNS M., SINGH S., « Near-Optimal Reinforcement Learning in Polynomial Time », *Machine Learning*, vol. 49, 1998.

[KEA 99] KEARNS M., KOLLER D., « Efficient Reinforcement Learning in Factored MDPs », *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, 1999.

[KEA 00] KEARNS M., MANSOUR Y., SINGH S., « Fast Planning in Stochastic Games », *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'00)*, 2000.

[KEA 02] KEARNS M., MANSOUR Y., NG A. Y., « A sparse Sampling Algorithm for Near-Optimal Planning in large Markov Decision Processes », *Machine Learning*, vol. 49, p. 193–208, 2002.

[KEE 76] KEENEY R., RAIFFA H., *Decisions with Multiple Objectives : Preferences and Value Tradeoffs*, J. Wiley, New York, 1976.

[KIM 98] KIMURA H., KOBAYASHI S., « Reinforcement Learning for Continuous Action using Stochastic Gradient Ascent », *Proceedings of the Fifth International Conference on Intelligent Autonomous Systems (IAS'98)*, p. 288–295, 1998.

[KIT 97] KITANO H., ASADA M., KUNIYOSHI Y., NODA I., OSAWA E., « RoboCup : The Robot World Cup Initiative », *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, New York, NY, USA, ACM Press, p. 340–347, 1997.

[KLO 72] KLOPF H. A., Brain Function and Adaptive Systems, A Heterostatic Theory, Rapport n°Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, 1972.

[KLO 75] KLOPF H. A., « A Comparison of Natural and Artificial Intelligence », *SIGART newsletter*, vol. 53, p. 11–13, 1975.

[KOE 96] KOENIG S., SIMMONS R. G., « The Effect of Representation and Knowledge on Goal-Directed Exploration with Reinforcement-Learning Algorithms », *Machine Learning*, vol. 22, p. 227–250, 1996.

[KOL 94] KOLLER D., MEGIDDO N., VON STENGEL B., « Fast Algorithms for Finding Randomized Strategies in Game Trees », *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC '94)*, p. 750–759, 1994.

[KOL 99] KOLLER D., PARR R., « Computing Factored Value Functions for Policies in Structured MDPs », *Proceedings Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, p. 1332–1339, 1999.

[KOL 00] KOLLER D., PARR R., « Policy Iteration for Factored MDPs », *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'00)*, p. 326–334, 2000.

[KUS 97] KUSHNER H., YIN G., *Stochastic Approximation Algorithms and Applications*, Springer-Verlag, New York, 1997.

[KUV 96] KUVAYEV L., SUTTON R. S., « Model-Based Reinforcement Learning with an Approximate, Learned Model », *Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*, New Haven, CT, Yale University Press, p. 101–105, 1996.

[LAG 03] LAGOUDAKIS M., PARR R., « Least-Squares Policy Iteration », *Journal of Machine Learning Research*, vol. 4, p. 1107–1149, 2003.

[LAU 02] LAURENT G., Synthèse de comportements par apprentissages par renforcement parallèles : application à la commande d'un micromanipulateur plan, Thèse de doctorat, Université de Franche-Comté, Besançon, France, 2002.

[LIB 02] LIBERATORE P., « The size of MDP factored policies », *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, p. 267–272, 2002.

[LIN 93] LIN L.-J., « Scaling Up Reinforcement Learning for Robot Control », *Proceedings of the Tenth International Conference on Machine Learning (ICML'93)*, Amherst, MA, Morgan Kaufmann, p. 182-189, 1993.

[LIN 06] LINKIE M., CHAPRON G., MARTYR D. J., HOLDEN J., LEADER-WILLIAMS N., « Assessing the Viability of Tiger Subpopulations in a Fragmented Landscape », *Journal of Applied Ecology*, vol. 43, n°3, p. 576–586, 2006.

[LIT 94] LITTMAN M., « Markov Games as a Framework for Multi-Agent Reinforcement Learning », *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, 1994.

[LIT 95a] LITTMAN M., CASSANDRA A., KAELBLING L., Efficient Dynamic Programming Updates in Partially Observable Markov Decision Processes, Rapport n°CS-95-19, Brown University, 1995.

[LIT 95b] LITTMAN M. L., CASSANDRA A. R., KAELBLING L. P., « Learning Policies for Partially Observable Environments : Scaling Up », *Proceedings of the Twelfth International Conference on Machine Learning (ICML'95)*, 1995.

[LIT 95c] LITTMAN M., DEAN T., KAELBLING L., « On the Complexity of Solving Markov Decision Problems », *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI'95)*, Montreal, Québec, Canada, 1995.

[LIT 96] LITTMAN M. L., Algorithms for Sequential Decision Making, PhD thesis, Computer Science Department, Brown University, 1996.

[LIT 01] LITTMAN M., STONE P., « Leading Best-Response Strategies in Repeated Games », *Seventeenth Annual International Joint Conference on Artificial Intelligence – Workshop on Economic Agents, Models, and Mechanisms*, 2001.

[LIT 02] LITTMAN M., SUTTON R., SINGH S., « Predictive Representation of State », *Advances in Neural Information Processing Systems 16 (NIPS'02)*, 2002.

[LIT 05] LITTLE I., ABERDEEN D., THIÉBAUX S., « Prottle : A Probabilistic Temporal Planner », *Proceedings of the Twentieth American National Conference on Artificial Intelligence (AAAI'05)*, 2005.

[LIT 06] LITTLE I., THIÉBAUX S., « Concurrent Probabilistic Planning in the Graphplan Framework », *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS'06)*, 2006.

[LON 03] LONG D., FOX M., « Exploiting a Graphplan Framework in Temporal Planning », *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS'03)*, p. 51–62, 2003.

[LUC 03] LUCE R., « Increasing Increment Generalizations Of Rank-Dependent Theories », *Theory and Decision*, vol. 55, n°2, p. 87–146, 2003.

[MAC 03] MACKAY D., *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.

[MAH 96a] MAHADEVAN S., « An Average-Reward Reinforcement Learning Algorithm for Computing Bias-Optimal Policies », *Proceedings of the National Conference on Artificial Intelligence (AAAI'96)*, vol. 13, 1996.

[MAH 96b] MAHADEVAN S., « Average Reward Reinforcement Learning : Foundations, Algorithms and Empirical Results », *Machine Learning*, vol. 22, p. 159–196, 1996.

[MAH 97] MAHADEVAN S., MARCHALLECK N., DAS T., GOSAVI A., « Self-Improving Factory Simulation using Continuous-Time Average-Reward Reinforcement Learning », *Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97)*, 1997.

[MAL 88] MALONE T.W., « What is coordination theory », *National Science Foundation Coordination Theory Workshop*, 1988.

[MAL 97] MALLAT S., *A Wavelet Tour of Signal Processing*, Academic Press, 1997.

[MAN 60] MANNE A. S., *Linear Programming and Sequential Decisions*, Cowles Foundation for Research in Economics at Yale University, 1960.

[MAR 75] MARTIN D. A., « Borel determinacy », *Annals of Mathematics*, vol. 102, p. 363–371, 1975.

[MAR 07] DE MARGERIE E., MOURET J.-B., DONCIEUX S., MEYER J.-A., « Artificial Evolution of the Morphology and Kinematics in a Flapping-Wing Mini UAV », *Bioinspir. Biomim.*, vol. 2, p. 65–82, 2007.

[MAT 62] MATTHES K., « Zut Theorie der Bedienungsprozesse », *Transactions of the Third Prague Conference on Information Theory, Satatistical Decision Functions, Random Processes*, Publishing House of the Czechoslovak Academy of Sciences, 1962.

[MAU 04] MAUSAM, WELD D., « Solving Concurrent Markov Decision Processes », *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI'04)*, 2004.

[MAU 05] MAUSAM, WELD D., « Concurrent Probabilistic Temporal Planning », *Proceedings of the Fifteenth International Conference on Planning and Scheduling (ICAPS'05)*, 2005.

[MAU 06] MAUSAM, WELD D., « Probabilistic Temporal Planning with Uncertain Durations », *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI'06)*, 2006.

[MAY 73] MAY R. M., *Stability and complexity in model ecosystems*, Monographs in population biology ; 6., Princeton University Press, Princeton, N.J., 1973, Robert M. May. Model ecosystems ill. ; 23 cm.

[MCC 95] MCCALLUM A., Reinforcement Learning with Selective Perception and Hidden State, PhD thesis, Dept. of Computer Science, University of Rochester, Rochester, NY, 1995.

[MCD 08] MCDONALD-MADDEN E., CHADES I., MCCARTHY M., LINKIE M., POSSINGHAM H., « ? ? ? », *Proceedings of the International Congress on Modelling and Simulation (MODSIM'08)*, 2008.

[MEU 96] MEULEAU N., *Le dilemme entre exploration et exploitation dans l'apprentissage par renforcement*, Cemagref, Thèse de doctorat de l'Université de Caen, 1996.

[MEU 99a] MEULEAU N., BOURGINE P., « Exploration of Multi-State Environments : Local Measures and Back-Propagation of Uncertainty », *Machine Learning*, vol. 35, n°2, p. 117–154, 1999.

[MEU 99b] MEULEAU N., PESHKIN L., KIM K.-E., KAELBLING L., « Learning Finite-State Controllers for Partially Observable Environments », *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99)*, p. 427–436, 1999.

[MEU 01] MEULEAU N., PESHKIN L., KIM K., Exploration in Gradient-Based Reinforcement Learning, Rapport n°AI Memo 2001-003, MIT - AI lab, 2001.

[MIC 61] MICHIE D., « Trial and Error », *Science Survey*, vol. 2, p. 129–145, 1961.

[MIC 68] MICHIE D., CHAMBERS R., « BOXES : An Experiment in Adaptive Control », *Machine Intelligence*, vol. 2, p. 137–152, 1968.

[MIN 77] MINOUX M., « Generalized Path Algebra », *Surveys of Mathematical Programming*, Publishing House of the Hungarian Academy of Sciences, p. 359–364, 1977.

[MON 82] MONAHAN G. E., « A Survey of Partially Observable Markov Decison Processes : Theory, Models and Algorithms », *Management Science*, vol. 28, n°1, p. 1–16, 1982.

[MOO 93] MOORE A., ATKESON C., « Prioritized Sweeping : Reinforcement Learning with Less Data and Less Real Time », *Machine Learning*, vol. 13, p. 103–130, 1993.

[MOU 04] MOUADDIB A. I., « Multi-Objective Decision-Theoretic Path Planning », *IEEE International Conference on Robotics and Automaton (ICRA'04)*, 2004.

[MUN 00] MUNDHENK M., GOLDSMITH J., LUSENA C., ALLENDER E., « Complexity of Finite-Horizon Markov Decision Process Problems », *Journal of the ACM (JACM)*, vol. 47, n°4, p. 681–720, ACM Press New York, NY, USA, 2000.

[MUN 03] MUNOS R., « Error Bounds for Approximate Policy Iteration », *Proceedings of the Nineteenth International Conference on Machine Learning (ICML'03)*, 2003.

[MUN 06] MUNOS R., « Geometric Variance Reduction in Markov chains. Application to Value Function and Gradient Estimation », *Journal of Machine Learning Research*, vol. 7, p. 413–427, 2006.

[MUN 07a] MUNOS R., SZEPESVÁRI C., « Finite Time Bounds for Sampling Based Fitted Value Iteration », *To appear in Journal of Machine Learning Research*, 2007.

[MUN 07b] MUNOS R., « Performance Bounds in Lp norms for Approximate Value Iteration », *SIAM Journal on Control and Optimization*, vol. 46, 2007.

[MYE 97] MYERSON R. B., Ed., *Game Theory : Analysis of Conflict*, Harvard University Press, 1997.

[NAI 03] NAIR R., TAMBE M., YOKOO M., MARSELLA S., PYNADATH D.V., « Taming Decentralized POMDPs : Towards Efficient Policy Computation for Multiagent Settings », *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'03)*, p. 705-711, 2003.

[NAI 05] NAIR R., PRADEEP V., MILIND T., MAKOTO Y., « Networked Distributed POMDPs : A Synthesis of Distributed Constraint Optimization and POMDPs », *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI'05)*, 2005.

[NAS 51] NASH J. F., « Non-cooperative games », *Annals of Mathematics*, vol. 54, p. 286–295, 1951.

[NDI 99] NDIAYE S., *Apprentissage par renforcement en horizon fini : application à la génération de règles pour la Conduite de Culture*, Thèse de doctorat de l'Universite Paul Sabatier, Toulouse, février 1999.

[NEU 28] VON NEUMANN J., « Zur Theorie der Gesellschaftsspiele », *Mathematische Annalen*, vol. 100, n°1928, p. 295–320, 1928.

[NIL 04] NILIM A., GHAOUI L. E., « Robustness in Markov Decision Problems with Uncertain Transition Matrices », *Advances in Neural Information Processing Systems 16 (NIPS'03)*, 2004.

[NIL 05] NILIM A., GHAOUI L. E., « Robust Solutions to Markov Decision Problems with Uncertain Transition Matrices », *Operation Research*, vol. 53, n°5, 2005.

[ORM 02] ORMONEIT D., SEN S., « Kernel-Based Reinforcement Learning », *Machine Learning*, vol. 49, p. 161–178, 2002.

[OSB 04] OSBORNE M. J., Ed., *An Introduction to Game Theory*, Oxford University Press, 2004.

[PAP 87] PAPADIMITRIOU C. H., TSITSIKLIS J. N., « The Complexity of Markov Decision Processes », *Journal of Mathematics of Operations Research*, vol. 12, n°3, p. 441–450, 1987.

[PAP 95] PAPADIMITRIOU C. H., « Algorithms, Games, and the Internet », *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC'91)*, ACM Press, p. 749–753, 1995.

[PAR 02] PARSONS S., GMYTRASIEWICZ P., WOOLWRIDGE M., Eds., *Game Theory and Decision Theory in Agent-Based Systems*, Springer Verlag, 2002.

[PEA 88] PEARL J., *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, 1988.

[PEN 92] PENG J., WILLIAMS R., « Efficient Learning and Planning within the DYNA framework », MEYER J.-A., ROITBLAT H. L., WILSON S. W., Eds., *Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB'92)*, Cambridge, MA, MIT Press, p. 281–290, 1992.

[PEN 93] PENG J., WILLIAMS R. J., « Efficient Learning and Planning within the Dyna Framework », *Adaptive Behavior*, vol. 1, n°4, p. 437–454, 1993.

[PEN 94] PENG J., WILLIAMS R. J., « Incremental Multi-Step Q-Learning », *Proceedings of the International Conference on Machine Learning (ICML'94)*, vol. 11, p. 226–232, 1994.

[PEN 96] PENG J., WILLIAMS R. J., « Incremental Multi-Step Q-learning », *Machine Learning*, vol. 22, p. 283–290, Elsevier, 1996.

[PER 04] PERET L., GARCIA F., « On-line Search for Solving MDPs via Heuristic Sampling », *Proceedings of the European Conference on Artificial Intelligence (ECAI'04)*, 2004.

[PER 05] PERNY P., SPANJAARD O., WENG P., « Algebraic Markov Decision Processes », *Proceedings of the International Joint Conference in Artificial Intelligence*, vol. 19, p. 1372–1377, 2005.

[PES 88] PESHKIN M., SANDERSON A., « The Motion of a Pushed, Sliding Workpiece », *IEEE Journal on Robotics and Automation*, vol. 4, n°6, p. 569–598, 1988.

[PES 00] PESHKIN L., KIM K., MEULEAU N., KAELBLING L., « Learning to Cooperate via Policy Search », *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'00)*, p. 489–496, 2000.

[PET 03] PETERS J., VIJAYAKUMAR S., SCHAAL S., Policy Gradient Methods for Robot Control, Rapport n°CS-03-787, University of Southern California, 2003.

[PET 05] PETERS J., VIJAYAKUMAR S., SCHAAL S., « Natural Actor-Critic », GAMA J., CAMACHO R., BRAZDIL P., JORGE A., TORGO L., Eds., *Proceedings of the Sixteenth European Conference on Machine Learning (ECML'05)*, vol. 3720 de *Lecture Notes in Computer Science*, Springer-Verlag, October 2005.

[PIN 03] PINEAU J., GORDON G., THRUN S., « Point-based value iteration : An anytime algorithm for POMDPs », *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'03)*, p. 1025–1032, 2003.

[POL 69] POLLATSCHEK M., AVI-ITZHAK B., « Algorithms for Stochastic Games with Geometrical Interpretation », *Management Science*, vol. 15, n°7, p. 399–415, JSTOR, 1969.

[POL 84] POLLARD D., *Convergence of Stochastic Processes*, Springer Verlag, New York, 1984.

[PON 62] PONTRYAGIN L., BOLTYANSKII V., GAMKRILEDZE R., MISCHENKO E., *The Mathematical Theory of Optimal Processes*, Interscience, New York, 1962.

[POO 97] POOLE D., « The Independent Choice Logic for Modelling Multiple Agents under Uncertainty », *Artificial Intelligence*, vol. 94, n°1-2, p. 7–56, 1997.

[POS 01] POSSINGHAM H. P., ANDELMAN S. J., NOON B. R., S. T., PULLIAM H. R., « Making Smart Conservation Decisions », SOULE M. E., ORIANS G. H., Eds., *Conservation Biology : Research Priorities for the Next Decade*, Island Press, Washington, 2001.

[POW 05a] POWERS R., SHOHAM Y., « Learning Against Opponents with Bounded Memory », *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, 2005.

[POW 05b] POWERS R., SHOHAM Y., « New Criteria and a New Algorithm for Learning in Multi-Agent Systems », SAUL L. K., WEISS Y., BOTTOU L., Eds., *Advances in Neural Information Processing Systems 17 (NIPS'05)*, MIT Press, 2005.

[PRA 06] PRALET C., VERFAILLIE G., SCHIEX T., « Decision with Uncertainties, Feasibilities and Utilities : Towards a Unified Algebraic Framework », *Proceedings of the European Conference on Artificial Intelligence (ECAI'06)*, p. 427–431, 2006.

[PUT 94] PUTERMAN M., *Markov Decision Processes : Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Inc., New York, USA, 1994.

[PYN 02] PYNADATH D. V., TAMBE M., « The Communicative Multiagent Team Decision Problem : Analyzing Teamwork Theories and Models », *Journal of Artificial Intelligence Research*, vol. 16, p. 389–423, 2002.

[QUI 93] QUINLAN J. R., *C4.5 : Programs for Machine Learning*, Morgan Kaufmann, San Mateo, 1993.

[RAS 04] RASMUSSEN C., KUSS M., « Gaussian Processes in Reinforcement Learning », THRUN S., SAUL L., SCHÖLKOPF B., Eds., *Advances in Neural Information Processing Systems 16 (NIPS'04)*, MIT Press, p. 751–759, 2004.

[REE 77] REETZ D., « Approximate Solutions of a Discounted Markovian Decision Problem », *Bonner Mathematischer Schriften*, vol. 98 : Dynamische Optimierungen, p. 77–92, 1977.

[RES 72] RESCORLA R. A., WAGNER A. R., « A Theory of Pavlovian Conditioning : Variations in the Effectiveness of Reinforcement and Nonreinforcement », BLACK A. H., PROKAZY W. F., Eds., *Classical Conditioning II*, p. 64–99, Appleton Century Croft, New York, NY, 1972.

[RES 00] RESCH R., LEWIS D., MELTZER S., MONTOYA N., KOEL B., MADHUKAR A., REQUICHA A., WILL P., « Manipulation of Gold Nanoparticles in Liquid Environnements using Scanning Force Microscopy », *Ultramicroscopy*, vol. 82, p. 135–139, 2000.

[RIV 87] RIVEST R. L., « Learning Decision Lists », *Machine Learning*, vol. 2, p. 229–246, 1987.

[RUM 94] RUMMERY G. A., NIRANJAN M., On-Line Q-learning using Connectionist Systems, Rapport, Cambridge University Engineering Department, Cambridge, England, 1994.

[RUS 96] RUST J., « Numerical Dynamic Programming in Economics », AMMAN H., KENDRICK D., RUST J., Eds., *Handbook of Computational Economics*, Elsevier, North Holland, 1996.

[RUS 03] RUSSEL S., NORVIG P., Eds., *Artificial Intelligence : A Modern Approach*, Prentice Hall Series, 2003.

[SAB 98] SABBADIN R., Une approche ordinale de la décision dans l'incertain : axiomatisation, représentation logique et application à la décision séquentielle, PhD thesis, Université Paul Sabatier de Toulouse, 1998.

[SAB 99] SABBADIN R., « A Possibilistic Model for Qualitative Sequential Decision Problems under Uncertainty in Partially Observable Environments », LASKEY K., PRADE H., Eds., *Proceedings of the Fifteenth Conference Uncertainty in Artificial Intelligence (UAI'99)*, Stockholm, Sweden, Morgan Kaufmann, p. 567–574, Jul. 30-Aug. 1 1999.

[SAB 01a] SABBADIN R., « Possibilistic Markov Decision Processes », *Engineering Applications of Artificial Intelligence*, vol. 14, p. 287–300, Elsevier, 2001.

[SAB 01b] SABBADIN R., « Towards Possibilistic Reinforcement Learning Algorithms », *Proceedings of the Tenth IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'01)*, vol. 1, Melbourne, p. 404–407, 2-5 Décembre 2001.

[SAM 59] SAMUEL A., « Some Studies in Machine Learning using the Game of Checkers », *IBM Journal of Research Development*, vol. 3, n°3, p. 210–229, 1959.

[SAM 67] SAMUEL A., « Some Studies in Machine Learning using the Game of Checkers, II – Recent Progress », *IBM Journal on Research and Development*, vol. 11, n°6, p. 601–617, 1967.

[SAV 54] SAVAGE L. J., *The Foundations of Statistics*, J. Wiley and Sons, New York, 1954.

[SCH 85] SCHWEITZER P., SEIDMANN A., « Generalized Polynomial Approximations in Markovian Decision Processes », *Journal of Mathematical Analysis and Applications*, vol. 110, p. 568–582, 1985.

[SCH 93] SCHWARTZ A., « A Reinforcement Learning Method for Maximizing Undiscounted Rewards », *Proceedings of the Tenth International Conference on Machine Learning (ICML'93)*, 1993.

[SCH 94] SCHOENAUER M., RONALD E., « Neuro-Genetic Truck Backer-Upper Controller », *Proceedings of the First International Conference on Evolutionary Computation (ICEC'94)*, June 1994.

[SCH 01] SCHOLKOPF B., SMOLA A. J., *Learning with Kernels : Support Vector Machines, Regularization, Optimization and Beyond*, MIT Press, 2001.

[SCH 02a] SCHERRER B., CHARPILLET F., « Cooperative Co-Learning : A Model-Based Approach for Solving Multi Agent Reinforcement Problems », *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI'02)*, 2002.

[SCH 02b] SCHOKNECHT R., « Optimality of Reinforcement Learning Algorithms with Linear Function Approximation », *Advances in Neural Information Processing Systems (NIPS'02)*, 2002.

[SCH 05a] SCHAEFFER S., CLEMENT B., CHIEN S., « Probabilistic Reasoning for Plan Robustness », *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, 2005.

[SCH 05b] SCHRAUDOLPH N., YU J., ABERDEEN D., « Fast Online Policy-Gradient Learning With SMD Gain Vector Adaptation », *Advances in Neural Information Processing Systems 19 (NIPS'05)*, 2005.

[SER 06] SERAFINI P., « Dynamic Programming and Minimum Risk Paths », *European Journal of Operational Research*, vol. 175, p. 224–237, 2006.

[SEU 05] SEUKEN S., ZILBERSTEIN S., Formal Models and Algorithms for Decentralized Control of Multiple Agents, Rapport, Computer Science Department, University of Massachusetts, Amherst, 2005.

[SEU 07] SEUKEN S., ZILBERSTEIN S., « Memory-Bounded Dynamic Programming for DEC-POMDPs », *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, 2007.

[SHA 53] SHAPLEY L.S.., « Stochastic Games », *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, vol. 39, p. 1095–1100, 1953.

[SHA 76] SHAFER G., *A Mathematical Theory of Evidence*, Princeton University Press, 1976.

[SHE 01] SHELTON C., Importance Sampling for Reinforcement Learning with Multiple Objectives, Rapport n°AI Memo 2001-003, MIT AI Lab, 2001.

[SHO 04] SHOHAM Y., POWERS R., GRENAGER T., « Multi-Agent Reinforcement Learning : a Critical Survey », *Proceedings of the AAAI Fall Symposium on Artificial Multi-Agent Learning*, 2004.

[SIG 04] SIGAUD O., *Comportements adaptatifs pour les agents dans des environnements informatiques complexes*, Mémoire d'Habilitation à Diriger des Recherches de l'Université PARIS VI, 2004.

[SIN 94a] SINGH S., JAAKKOLA T., JORDAN M., « Learning without State Estimation in Partially Observable Markovian Decision Processes », *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, 1994.

[SIN 94b] SINGH S., KEARNS M., MANSOUR Y., « Nash Convergence of Gradient Dynamics in General-Sum Games », *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'94)*, San Francisco, CA, Morgan Kaufman, p. 541–548, 1994.

[SIN 96] SINGH S. P., SUTTON R. S., « Reinforcement Learning with Replacing Eligibility Traces », *Machine Learning*, vol. 22, n°1, p. 123–158, 1996.

[SIN 97] SINGH S., BERTSEKAS D., « Reinforcement Learning for Dynamic Channel Allocation in Cellular Telephone Systems », *Advances in Neural Information Processing Systems 9 (NIPS'97)*, 1997.

[SIN 00] SINGH S. P., JAAKKOLA T., LITTMAN M. L., SZEPESVARI C., « Convergence Results for Single-Step On-Policy Reinforcement Learning Algorithms », *Machine Learning*, vol. 38, n°3, p. 287–308, 2000.

[SIN 03] SINGH S., LITTMAN M., JONG N., PARDOE D., STONE P., « Learning Predictive State Representations », *Proceedings of the Twentieth International Conference of Machine Learning (ICML'03)*, 2003.

[SLA 01] SLANEY J., THIÉBAUX S., « Blocks World Revisited », *Artificial Intelligence*, vol. 125, p. 119–153, 2001.

[SMA 73] SMALLWOOD R. D., SONDIK E. J., « The Optimal Control of Partially Observable Markov Processes over a Finite Horizon », *Operations Research*, vol. 21, p. 1071–1088, 1973.

[SMI 02] SMITH J. M., Ed., *Evolution and the Theory of Games*, Cambridge University Press, 2002.

[SON 71] SONDIK E., The Optimal Control of Partially Observable Markov Decision Processes, PhD thesis, Stanford University, California, 1971.

[SON 78] SONDIK E., « The Optimal Control of Partially Observable Markov Processes over the Infinite Horizon : Discounted Costs », *Operations Research*, vol. 26, n°2, p. 282–304, 1978.

[SPA 05] SPAAN M., VLASSIS N., « Perseus : Randomized Point-based Value Iteration for POMDPs », *Journal of Artificial Intelligence Research*, vol. 24, p. 195–220, 2005.

[STA 00] ST-AUBIN R., HOEY J., BOUTILIER C., « APRICODD : Approximate Policy Construction Using Decision Diagrams », *Advances in Neural Information Processing Systems 13 (NIPS'00)*, p. 1089–1095, 2000.

[STO 00] STONE P., VELOSO M., « Multiagent Systems : A Survey from a Machine Learning Perspective », *Autonomous Robots*, vol. 8, n°3, p. 345–383, Springer, 2000.

[STR 07] STREHL A., DIUK C., LITTMAN M. L., « Efficient Structure Learning in Factored-state MDPs », *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI'07)*, 2007.

[SUT 81] SUTTON R., BARTO A., « Toward a Modern Theory of Adaptive Network : Expectation and Prediction », *Psychological Review*, vol. 88, n°2, p. 135–170, 1981.

[SUT 88] SUTTON R., « Learning to Predict by the Method of Temporal Differences », *Machine Learning*, vol. 3, n°1, p. 9–44, 1988.

[SUT 90a] SUTTON R. S., « Integrated Architectures for Learning, Planning and Reacting Based on Approximating Dynamic Programming », *Proceedings of the Seventh International Conference on Machine Learning (ICML'90)*, p. 216–224, 1990.

[SUT 90b] SUTTON R. S., « Planning by Incremental Dynamic Programming », *Proceedings of the Eighth International Conference on Machine Learning (ICML'91)*, San Mateo, CA, Morgan Kaufmann, p. 353-357, 1990.

[SUT 90c] SUTTON R., « Integrating Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming », *Proceedings of the Seventh International Conference on Machine Learning (ICML'90)*, San Mateo, CA, Morgan Kaufmann, p. 216–224, 1990.

[SUT 98] SUTTON R. S., BARTO A. G., *Reinforcement Learning : An Introduction*, Bradford Book, MIT Press, Cambridge, MA, 1998.

[SUT 00] SUTTON R., MCALLESTER D., SINGH S., MANSOUR Y., « Policy Gradient Methods for Reinforcement Learning with Function Approximation », *Advances in Neural Information Processing Systems 12 (NIPS'99)*, MIT Press, p. 1057–1063, 2000.

[SZE 05] SZER D., CHARPILLET F., ZILBERSTEIN S., « MAA* : A Heuristic Search Algorithm for Solving Decentralized POMDPs », *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI'05)*, 2005.

[SZI 06] SZITA I., LÖRINCZ A., « Learning Tetris Using the Noisy Cross-Entropy Method », *Neural Computation*, vol. 18, p. 2936–2941, 2006.

[TEI 05a] TEICHTEIL-KÖNIGSBUCH F., Approche Symbolique et Heuristique de la Planification en Environnement Incertain, PhD thesis, École Nationale Supérieure de l'Aéronautique et de l'Espace, 2005.

[TEI 05b] TEICHTEIL-KÖNIGSBUCH F., FABIANI P., « Symbolic Heuristic Policy Iteration Algorithms for Structured Decision-theoretic Exploration Problems », *ICAPS International Workshop on Planning under Uncertainty for Autonomous Systems*, 2005.

[TEI 05c] TEICHTEIL-KONIGSBUSCH F., Stratégie d'exploration pour un aéronef autonome, PhD thesis, Ecole Nationale Supérieure d'Aéronautique et de l'Espace, 2005.

[TES 95] TESAURO G., « Temporal Difference Learning and TD-Gammon », *Communication of the ACM*, vol. 38, p. 58–68, 1995.

[TES 04] TESAURO G., « Extending Q-Learning to General Adaptive Multi-Agent Systems », THRUN S., SAUL L., SCHOLKOPF B., Eds., *Advances in Neural Information Processing Systems 16 (NIPS'04)*, Cambridge, MA, MIT Press, 2004.

[THI 04] THISSE J.-F., *Théorie des jeux : une introduction*, Université catholique de Louvain, Département des sciences économiques, 2004, Notes de cours.

[THR 92] THRUN S., « The Role of Exploration in Learning Control », WHITE D., SOFGE D., Eds., *Handbook for Intelligent Control : Neural, Fuzzy and Adaptive Approaches*, Van Nostrand Reinhold, Florence, Kentucky 41022, 1992.

[TSI 96a] TSITSIKLIS J. N., VAN ROY B., « Feature-Based Methods for Large Scale Dynamic Programming », *Machine Learning*, vol. 22, p. 59–94, 1996.

[TSI 96b] TSITSIKLIS J., ROY B. V., An Analysis of Temporal Difference Learning with Function Approximation, Rapport n°LIDS-P-2322, MIT, 1996.

[UCH 04] UCHIBE E., DOYA K., « Competitive-Cooperative-Concurrent Reinforcement Learning with Importance Sampling », *From Animals to Animats 8 : Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior (SAB'04)*, p. 287–296, 2004.

[UTH 03] UTHER W., VELOSO M., Adversarial reinforcement learning, Rapport n°CMU-CS-03-107, School of Computer Science, Carnegie Mellon University, 2003.

[VAL 84] VALIANT L. G., « A Theory of the Learnable », *Communications of the ACM*, vol. 27, n°11, p. 1134–1142, November 1984.

[VAP 97] VAPNIK V., GOLOWICH S. E., SMOLA A., « Support Vector Method for Function Approximation, Regression Estimation and Signal Processing », *Advances in Neural Information Processing Systems (NIPS'97)*, p. 281–287, 1997.

[VAP 98] VAPNIK V., *Statistical Learning Theory*, John Wiley & Sons, New York, 1998.

[VER 07] VERFAILLIE G., LEMAÎTRE M., « A Generic Modular Architectural Framework for the Closed-Loop Control of a System », *Proceedings of the 2nd National Workshop on Control Architectures of Robots*, p. 19–31, 2007.

[VIN 89] VINCKE P., *L'aide multicritère à la décision*, Statistique et mathématiques appliquées, Edition de l'université de bruxelles, edition ellipses édition, 1989.

[VRI 87] VRIEZE O., *Stochastic Games with Finite State and Action Spaces*, Centrum voor wiskunde en informatica, 1987.

[WAN 99] WANG X., DIETTERICH T., « Efficient Value Function Approximation Using Regression Trees », *Proceedings of the IJCAI Workshop on Statistical Machine Learning for Large-Scale Optimization*, Stockholm, Sweden, 1999.

[WAS 96] WASHINGTON R., « Incremental Markov-Model Planning », *Proceedings of the Eighth International Conference on Tools with Artificial Intelligence (ICTAI'96)*, 1996.

[WAT 89] WATKINS C., Learning from Delayed Rewards, PhD thesis, Cambridge University, Cambridge, UK, 1989.

[WAT 92] WATKINS C., DAYAN P., « Q-learning », *Machine Learning*, vol. 8, n°3, p. 279–292, Elsevier, 1992.

[WAT 00] WATSON J. C., « The Effects of Sea Otters (Enhydra Lutris) on Abalone (Haliotis spp.) Populations », CAMPBELL A., Ed., *Workshop on Rebuilding Abalone Stocks in British Columbia*, vol. 130, Can. Spec. Publ. Fish. Aquat. Sci., p. 123–132, 2000.

[WEN 06a] WENG P., « Axiomatic Foundations for a Class of Generalized Expected Utility : Algebraic Expected Utility », *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI'06)*, p. 520–527, 2006.

[WEN 06b] WENG P., Modèles qualitatifs et approches algébriques pour la décision dans l'incertain : fondements axiomatiques et application à la décision séquentielle, PhD thesis, Université Paris VI, Décembre 2006.

[WHI 91] WHITE C. C., « Partially Observed Markov Decision Processes : A Survey », *Annals of Operational Research*, vol. 32, 1991.

[WIL 87] WILLIAMS R., « A Class of Gradient-Estimating Algorithms for Reinforcement Learning in Neural Networks », *Proceedings of the First International Conference on Neural Networks (ICNN'87)*, 1987.

[WIL 92] WILLIAMS R., « Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning », *Machine Learning*, vol. 8, n°3, p. 229–256, Kluwer Academic Publishers, 1992.

[WIL 93] WILLIAMS R. J., BAIRD III L. C., Tight Performance Bounds on Greedy Policies Based on Imperfect Value Functions, Rapport n°NU-CCS-93-14, Northeastern University, College of Computer Science, Boston, MA, November 1993.

[WIL 06] WILSON K. A., MCBRIDE M. F., BODE M., POSSINGHAM H. P., « Prioritizing Global Conservation Efforts », *Nature*, vol. 440, n°7082, p. 337–340, 2006.

[WIT 77] WITTEN I. H., « An Adaptive Optimal Controller for Discrete-Time Markov Environments », *Information and Control*, vol. 34, p. 286–295, 1977.

[XUA 01] XUAN P., LESSER V., ZILBERSTEIN S., « Communication Decisions in MultiAgent Cooperation : Model and Experiments », *Proceedings of the Fifth International Conference on Autonomous Agents (Agents'01)*, Montreal, p. 616–623, 2001.

[YIL 03] YILDIZOGLU M., Ed., *Introduction à la théorie des jeux*, Dunod, Paris, 2003.

[YOU 93] YOUNG H., « The Evolution of Conventions », *Econometrica*, vol. 61, n°1, p. 57–84, 1993.

[YOU 98] YOUNG H., *Individual Strategy and Social Structure : An Evolutionary Theory of Institutions*, Princeton University Press, Princeton, New Jersey, 1998.

[YOU 03a] YOUNES H., « Extending PDDL to Model Stochastic Decision Processes », *Proceedings of the ICAPS-03 Workshop on PDDL*, 2003.

[YOU 03b] YOUNES H., SIMMONS R., « A Framework for Planning in Continuous-time Stochastic Domains », *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS'03)*, 2003.

[YOU 04a] YOUNES H. L. S., SIMMONS R. G., « Solving Generalized Semi-Markov Decision Processes Using Continuous Phase-Type Distributions. », *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI'04)*, 2004.

[YOU 04b] YOUNES H., LITTMAN M. L., PPDDL1.0 : An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects, Rapport n°CMU-CS-04-167, Carnegie Mellon University, October 2004.

[ZAN 96] ZANG N., LIO W., Planning in Stochastic Domains : Problem Characteristics and Approximation, Rapport n°HKUST-CS96-31, Honk-Kong University of Science and Technology, 1996.

[ZES 98] ZESCH W., S.FEARING R., « Alignment of Microparts Using Force Controlled Pushing », *Proceedings of the SPIE Conference on Microrobotics and Micromanipulation*, vol. 3519, Boston, Massachusetts, p. 148–156, november 1998.

[ZHA 95] ZHANG W., DIETTERICH T., « A Reinforcement Learning Approach to Job-Shop Scheduling », *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, 1995.

[ZHA 99] ZHANG T., POOLE D., « On the Role of Context-specific Independence in Probabilistic Reasoning », *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, Stockholm, p. 1288–1293, 1999.

[ZIL 02] ZILBERSTEIN S., WASHINGTON R., BERSTEIN D., MOUADDIB A., « Decision-Theoretic Control of Planetary Rovers », *LNAI*, vol. 2466, n°1, p. 270–289, 2002.

[ZIN 03] ZINKEVICH M., « Online Convex Programming and Generalized Infinitesimal Gradient Ascent », *Proceedings of the Twentieth International Conference on Machine Learning (ICML'03)*, 2003.

[Zyv 06] ZYVEX COMPANY, « S100 Nanomanipulator System Datasheet », 2006, http ://www.zyvex.com.

Index

A

abalone 222
acteur-critique 53, 370
action 86
action 17
adapté(e) 95
  fonction de valeur 97
  politique 96
  Q-learning 100
agent -joueur 128
agent 17
algorithme ADL 158
  de Shapley 146
  du jeu adaptatif 151
  du jeu fictif 149
  élagage itératif 120
  GIGA 160
  Hyper-Q 158
  IGA 151
  iterative pruning 120
  JALs 151
  Minimax-Q 148
  Nash-Q 148
  PHC 153
  PHC-Exploiter 158
  Q-learning adapté 100
  WITNESS 117, 118
  WoLF 154
apprentissage supervisé 333 51
apprentissage par renforcement
  (application) 221
  possibiliste 184
approximation de fonctions 329
arbre de décision 284
arcs synchrones 279

B

balayage prioritaire 80
belief state 90
bias optimality 40
biodiversité 215
bonus d'exploration 57

C

capacité d'approximation 330
chaîne de Markov 27
  de Markov valuée 27
  ergodique 366
coalition 129
co-évolution 264
COMDP 415
compromis biais-variance 317, 355, 368
critère de performance 22
  fini 22
  γ-pondéré 22
  moyen 22
  total 22
critère de performance moyen adapté 100

D

décomposition additive 282
Dec-POMDP-COM 267

diagramme de décision algébrique 290
diagramme d'influence 19
  possibiliste 187
dilemme du prisonnier 133
dimension de VC 352
distribution de probabilité conditionnelle 280
DP-JESP 265

E

échantillonnage heuristique 82
échantillonnage selon l'importance 378
ED-Dec-MDP 250
environnement stochastique 17
équation de Bellman POMDP 94
équations de Bellman algébrique 192, 193
  possibilistes 180
équilibre 133
  de Nash 134, 144
  en stratégies dominantes 134
  en stratégies mixtes 137
erreur empirique 334
  en apprentissage 352
  en généralisation 352
état 87
  de croyance 90
  d'information 88
  d'information complet 89
  prédictif 126
état 17
exclusion mutuelle 413
extensibilité 155

F

facteur d'actualisation 24
Factored Policy Gradient (FPG) 426
feature 333
fonction de récompense localisée 282
fonction de valeur 23, 93
  adaptée 97
  ϵ-optimale 102
  linéaire par morceau 103
  POMDP 93
  relative 37
  vecteur représentatif 106

G H

gain escompté 141
gain espéré 136
gloutonne (politique) 331
gradient ascent infinitesimal 151
  naturel 376
gradient 358
graphe de planification 421
heuristique 267
historique du jeu 138
horizon fini 19, 102
  infini 19, 102

I J K

Induction arrière algébrique 193
itérations sur les politiques 123
  POMDP 123
itérations sur les politiques avec approximation 338
itérations sur les valeurs 332
  avec approximation 332
itération sur les politiques possibiliste 183
itération sur les valeurs possibiliste 180
Iterative Pruning 120
JESP 265
jeu 128
  coopératif 129
  de pile ou face 135
  du dilemme du prisonnier 140
  dynamique 133, 137
  efficace 158
  en forme extensive 129
  en forme stratégique 129, 130
  en information parfaite 138
  fini 130
  non-coopératif 129
  répété 138, 140
  statique 133
  stochastique 143

L

loutre 221

M

machines à vecteurs de support 335

matrice de déviation 38
limite 37
stochastique 19
mesure de plausibilité 189
décomposable 189
utilité espérée généralisée 189
mesure invariante 37
méthodes à noyaux 335
méthodes de résolution approchée 330
exacte 329
minimax 131
MMDP 265
modèle génératif 347
moindres carrés (méthode) 344
mutex 413

N O

Natural Actor-Critic 376
niveaux de sécurité 131
nombre de couverture 352
norme L^p 351
max 29
quadratique 334
semi-norme span 46
observabilité 236
observation 87
OLpomdp 369
opérateur d'approximation 333
de Bellman 331
opération 411

P

Pareto dominance au sens de 137
optimum de 137
Pareto-dominance algébrique 192
plan conditionnel 92
plan 414
planification 17
planning graph 421
plus court chemin 17
point selle 133
politique 18, 92, 130
adaptée 96
arbre de 92
automate à états fini 93
cyclique 93
d'action 20
optimale 23
paramétrée 360
POMDP 85, 86
transitoire 92
pomdp application 216
Incremental Pruning 217
multi-agent 220
possibilité distribution 175 126
prioritized sweeping possibiliste 184
probablement approximativement correct 315, 354
problème décisionnel de Markov 17
problème de planification d'opérations 411
processus markovien 27
markovien valué 27
régénératif 367
stationnaire 20
stochastique contrôlé 18 17, 190
processus décisionnel de Markov
algébrique 187
multicritère 191, 194
multicritère possibiliste 194
partiellement observable 86
partiellement observable possibiliste 185
possibiliste 175, 191, 195
possibiliste (itération sur les politiques) 183
possibiliste (itération sur les valeurs) 180
qualitatif 191
robuste 172
subjectif 266
processus décisionnels de Markov
factorisés 275
programmation linéaire 41
neurodynamique 53
programmation dynamique 31
avec approximation 329
POMDP 102
possibiliste 178
proie-predateur 221
Prottle 423
PSR 126

Q

QH-learning 226
Q-learning adapté 100
Q-learning 66

R

récompense 18
régression 333
regret 159
    algorithme GIGA 160
    no-regret 159
relaxation 420
réseaux bayésiens 279
    dynamiques 279
réseaux de neurones 335
revenu 18
RH-learning 226
richesse d'approximation 330

S

scalability 155
scope 282
semi-anneau idempotent 188
    préordre canonique 188
semi-anneau 188
statistiques exhaustives 89
statistiques suffisantes 89
stratégie 129
    de la meilleure réponse 135
    grim trigger 140
    markovienne 145
    mixte 130, 141
    pure 130
    stationnaire 145

T U

WITNESS 117, 118
théorie de l'approximation 335
théorie des jeux 127
trace d'éligibilité 68
utilité espérée généralisée 189
    qualitative binaire possibiliste 195
    qualitative optimiste 177, 191, 194
    qualitative pessimiste 177, 191, 194

V

variance 312
vecteur dominé 106, 113
    fonction de valeur 106
    utile 106
    voisinage 117