

Tabu-NG : hybridation de programmation par contraintes et recherche locale pour la résolution de CSP

Mohammad Dib

▶ **To cite this version:**

Mohammad Dib. Tabu-NG : hybridation de programmation par contraintes et recherche locale pour la résolution de CSP. Algorithme et structure de données [cs.DS]. Université de Technologie de Belfort-Montbeliard, 2010. Français. NNT : . tel-00644178

HAL Id: tel-00644178

https://theses.hal.science/tel-00644178

Submitted on 23 Nov 2011

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.

THÈSE

préparée à

L'UNIVERSITE DE TECHNOLOGIE DE BELFORT-MONTBELIARD

pour obtenir le grade de

DOCTEUR de l'Université de Technologie de Belfort-Montbéliard et
de l'Université de Franche-Comté

Discipline : **INFORMATIQUE**

par

Mohammad DIB

Tabu-NG : hybridation de programmation par contraintes et recherche locale pour la résolution de CSP

Soutenue publiquement le mercredi 8 décembre 2010

Composition du jury :

Président :	Patrick SIARRY, Professeur - Université Paris-Est Créteil
Rapporteurs :	Christine SOLNON, Maitre de Conférences HDR - Université de Lyon I
	Michel VASQUEZ, Professeur - Ecole des Mines d'Alès
Examinateurs :	Clarisse DHAENENS, Professeur - Polytech'Lille, Université de Lille I
	Thierry DEFAIX, Docteur - Délégation Générale de l'Armement
	Jean-François LEGENDRE, Docteur - Délégation Générale de l'Armement
Directeur de thèse :	Alexandre CAMINADA, Professeur, UTBM
Co-encadrant :	Hakim MABED, Maître de Conférences, Université de Franche-Comté

Table des matières

1. Introduction 15

1.1. Contexte de travail 15
1.2. Motivations et principales contributions 16
1.3. Organisation de la thèse 18

2. Introduction aux CSP et à *Tabu-NG* 20

2.1. Les problèmes de satisfaction de contraintes 21
2.1.1. Le modèle CSP et ses principales caractéristiques 21
2.1.2. Les problèmes d'optimisation sous contraintes 24
2.1.3. Exemple de problèmes et formalisations 24
2.2. Méthodes de résolution de CSP 27
2.2.1. Méthodes de propagation de contraintes 28
2.2.2. Méthodes exactes de résolution 33
2.2.3. Les métaheuristiques 42
2.2.4. Hybridation de propagation de contraintes et recherche locale 49
2.3. *Tabu-NG* : *Tabu Search based on NoGoods*, une nouvelle méthode hybride 54
2.3.1. Principes généraux de la méthode proposée 54
2.3.2. Evaluation du voisinage et extension d'une configuration 56
2.3.3. Filtrage des domaines 59
2.3.4. Réparation d'une configuration partielle 60
2.3.5. *Tabu-NG* : Algorithme général 62
2.4. Synthèse 64

3. *Tabu-NG* et affectation de fréquences 66

3.1. Affectation de Fréquences Discrètes pour les réseaux tactiques militaires 67
3.1.1. Contexte 67
3.1.2. Description physique du problème 67
3.1.3. Description mathématique du problème 69
3.1.4. Les objectifs 76
3.1.5. Définition des classes d'optimisation 79

3.1.6. Les instances 81
3.1.7. Complexité 84
3.2. Modèle de référence et méthodes de résolution 84
3.2.1. AFD et coloration de graphe 84
3.2.2. AFD et satisfaction de contraintes 86
3.2.3. Méthodes de résolution 87
3.2.4. Discussion et orientation 92
3.3. Méthode *Tabu-NG* pour le problème AFD classique 93
3.3.1. Prétraitement 93
3.3.2. Propagation de contraintes et identification des *nogoods* 97
3.3.3. Paramètres de la méthode 101
3.3.4. Stratégies d'optimisation 102
3.3.5. Résultats et analyse 104
3.3.6. Discussions 110
3.4. Méthode *Tabu-NG* pour AFDP avec contraintes n-aires 110
3.4.1. Algorithme de filtrage et *nogood* partiel 110
3.4.2. Traitement des contraintes n-aires 117
3.4.3. Paramètres de la méthode 119
3.4.4. Stratégies d'optimisation 120
3.4.5. Résultats et analyse 125
3.4.6. Discussions 139
3.5. Synthèse 139

4. *Tabu-NG* et coloration de graphe 142

4.1. Coloration de graphe 143
4.1.1. Contexte 143
4.1.2. Définition formelle du problème et quelques notions 143
4.1.3. CSP et coloration de graphe 146
4.1.4. Benchmarks utilisés 147
4.1.5. Méthodes de résolution 148
4.2. Implémentations pour la coloration de graphe 151
4.2.1. Coloration de graphe versus affectation de fréquences discrètes 151
4.2.2. Adaptation de l'algorithme de filtrage et de propagation de contraintes 152
4.2.3. Adaptation des structures de données 154
4.2.4. Stockage des *nogoods* 159

4.3. Méthode *Tabu-NG* pour la coloration de graphe 160
4.3.1. Schéma général 160
4.3.2. Heuristique d'extension d'une configuration partielle consistante 162
4.3.3. Heuristique de réparation d'une configuration partielle inconsistante 162
4.3.4. Durée Tabou 163
4.3.5. Tests et analyse 164
4.4. Réglages des actions d'intensification et de diversification 166
4.4.1. Diversification de l'extension d'une configuration partielle consistante 166
4.4.2. Intensification de la réparation d'une configuration partielle inconsistante 166
4.4.3. Tests et analyse 167
4.4.4. Diversification du choix des couleurs 171
4.5. Comparaison des résultats obtenus avec la littérature 172
4.6. Synthèse 174

5. Conclusion et perspectives 177

Publications 181

Table des figures

Figure 1 : Carte de l'Australie 25
Figure 2 : Solution possible pour le coloriage de la carte de l'Australie 26
Figure 3 : Une solution réalisable pour le placement de 8 reines sur un échiquier 8*8 26
Figure 4 : Exemple d'une grille SUDOKU 9*9 à compléter 27
Figure 5 : Quelques relations entre des algorithmes d'arc-consistance 33
Figure 6 : Schéma général du *backtracking* 35
Figure 7 : Différences entre FC et MAC 37
Figure 8 : Comparaison de différents algorithmes de *look-ahead* 37
Figure 9 : Redondance et *trashing* 39
Figure 10 : Exemple de structures de voisinage 45
Figure 11 : *CN-Tabu* - Hybridation de la propagation de contraintes avec la recherche locale 51
Figure 12 : *Decision-Repair* : Hybridation de recherche locale, propagation de contraintes et *Nogood-Recording* 52
Figure 13 : *Tabu-NG* : Une hybridation de la propagation de contraintes, de *Nogood-Recording* et de recherche locale 55
Figure 14 : Organigramme global de *Tabu-NG* 55
Figure 15 : Fonctionnement de l'extension d'une configuration partielle 59
Figure 16 : Liaison hertzienne entre 2 sites 68
Figure 17 : Identification des classes de problèmes correspondant aux 9 modes d'optimisation 80
Figure 18 : Courbe représentative de la fonction T 118
Figure 19 : SCEN02 - Evolution de la taille de la liste Tabou Γ des *nogoods* en fonction du temps 126
Figure 20 : SCEN05 - Evolution de la taille de la liste Tabou Γ des *nogoods* en fonction du temps 128
Figure 21 : SCEN12 - Evolution de la liste Tabou Γ des *nogoods* en fonction du temps 129
Figure 22 : SCEN30 - Evolution de la liste Tabou Γ des *nogoods* en fonction du temps 129
Figure 23 : Un graphe *G(V, E)* 146
Figure 24 : Coloration partielle consistante du graphe *G(V, E)* 146
Figure 25 : Coloration valide du graphe *G(V, E)* 147

Figure 26 : Le graphe du problème *Myciel3* avec son tableau de contraintes 154
Figure 27 : Structure de la solution pour le problème *Myciel3* 155
Figure 28 : Représentation des domaines pour le problème *Myciel3* 155
Figure 29 : Propagation de contraintes pour le problème *Myciel3* 156
Figure 30 : Cas d'un deadend pour le problème *Myciel3* 157
Figure 31 : *Nogoods* déduits sur *Myciel3* 158
Figure 32 : Schéma général de *Tabu-NG* pour la coloration de graphe 161
Figure 33 : Blocage des variables 163
Figure 34 : Nombre de mouvements par couleur et par variable pour des problèmes résolus à l'optimalité 169
Figure 35 : Nombre de mouvements par couleur et par variable pour des problèmes non résolus 170
Figure 36 : Exemple détaillé pour *latin_square_10* non résolu avec 105 couleurs 171

Liste des tableaux

Tableau 1 : Complexité des algorithmes d'arc-consistance 33
Tableau 2 : Caractéristiques et objectifs des instances CELAR et GRAPH 82
Tableau 3 : Caractéristiques des instances privées 83
Tableau 4 : Rapport entre CSP et AFD 87
Tableau 5 : *MinSpec* – Synthèse des résultats sur 4 instances CALMA 88
Tableau 6 : *MinFreq* - Synthèse des résultats sur 10 instances CALMA 88
Tableau 7 : Synthèse des résultats sur les 44 instances AFDP 89
Tableau 8 : Réduction de domaines sur 15 instances 93
Tableau 9 : Comparaison des heuristiques de voisinage 105
Tableau 10 : Taille de la liste Tabou des *nogoods* 106
Tableau 11 : Comparaison des résultats obtenus sur les instances *MinSpec* avec la littérature 106
Tableau 12 : Comparaison des résultats de *Tabu-NG* et *CN-Tabu* sur les instances *MinSpec* 107
Tableau 13 : Comparaison des différentes stratégies sur les instances *MinFreq* 108
Tableau 14 : Comparaison des résultats obtenus avec la littérature sur *MinFreq* 109
Tableau 15 : Comparaison des résultats de *Tabu-NG* et *CN-Tabu* sur les instances *MinFreq* 109
Tableau 16 : Qualité des *nogoods* stockés pour SCEN02 127
Tableau 17 : Temps d'exécution alloué à chaque instance 131
Tableau 18 : Comparaison des résultats obtenus pour *MinFreq* 132
Tableau 19 : Comparaison des résultats obtenus pour *MinSpec* 133
Tableau 20 : Comparaison des résultats obtenus pour *MinFreqD* 134
Tableau 21 : Comparaison des résultats obtenus pour *MaxFreqG* 135
Tableau 22 : Comparaison des résultats obtenus pour *MinEcart* 136
Tableau 23 : Comparaison des résultats obtenus pour *MinCard* 137
Tableau 24 : Comparaison des résultats obtenus pour *MinMob* 138
Tableau 25 : Comparaison des résultats obtenus pour *MinMax* 138
Tableau 26 : Synthèse des résultats obtenus sur les instances privées fournies par le CELAR 139
Tableau 27 : Gain de temps sur la propagation de contraintes 153

Tableau 28 : Adaptation de *Tabu-NG* et comparaison de performances 158
Tableau 29 : Nombre d'itérations de *Tabu-NG* en 5 secondes avec et sans *nogood* 159
Tableau 30 : Tests de performances avec et sans *nogood* 159
Tableau 31 : Résultats obtenus sur des instances faciles 164
Tableau 32 : Résultats obtenus sur des instances difficiles 165
Tableau 33 : Résultats obtenus après le calibrage de *Tabu-NG* 168
Tableau 34 : Résultats obtenus après la diversification implémentée dans *Tabu-NG* 172
Tableau 35 : Comparaison des résultats de *Tabu-NG* avec la littérature pour les instances DIMACS 173

Liste des algorithmes

Algorithme 1 : Consistance de nœud 30
Algorithme 2 : Etablir la consistance pour un arc 31
Algorithme 3 : AC-1 31
Algorithme 4 : AC-3 32
Algorithme 5 : Fonctionnement global de la recherche locale 43
Algorithme 6 : Etendre une configuration partielle 58
Algorithme 7 : Vérifier l'appartenance d'une configuration partielle à la liste des *nogoods* ... 58
Algorithme 8 : Réparer une configuration partielle 61
Algorithme 9 : Ajouter un *nogood* à la liste des *nogoods* 62
Algorithme 10 : *Tabu-NG* : Algorithme général 63
Algorithme 11 : Prétraitement et filtrage par contraintes unaires 94
Algorithme 12 : Prétraitement : Forward Checking 95
Algorithme 13 : Prétraitement : Filtrage par Arc-Consistance 95
Algorithme 14 : Prétraitement : Algorithme de filtrage 95
Algorithme 15 : Initialisation et remplissage des structures 98
Algorithme 16 : Filtrage après extension de la configuration partielle courante 98
Algorithme 17 : Propagation de l'effet d'une valeur supprimée d'un domaine d'une variable . 99
Algorithme 18 : Filtrage après la réparation d'une configuration partielle courante 100
Algorithme 19 : Mise à jour des domaines après la réparation de la configuration partielle courante 101
Algorithme 20 : Filtrage et propagation des contraintes binaires et n-aires après extension d'une configuration partielle courante 113
Algorithme 21 : Propagation de l'effet de l'ajout d'une nouvelle affectation à une configuration partielle sur les variables voisines 114
Algorithme 22 : Filtrage des domaines par une contrainte d'intermodulation 114
Algorithme 23 : Filtrage des domaines par une contrainte de sommation de perturbateurs .. 115
Algorithme 24 : Filtrage des domaines par une contrainte binaire ou n-aire 116
Algorithme 25 : Filtrage des domaines après réparation d'une configuration partielle inconsistante 116
Algorithme 26 : Propagation d'une instanciation dans un problème de coloriage de graphe 153

Algorithme 27 : Pseudo-code général de *Tabu-NG* pour la coloration de graphe 161
Algorithme 28 : Etendre une configuration partielle consistante..... 162
Algorithme 29 : Réparation d'une configuration partielle inconsistante..... 167

Remerciements

Le présent travail a été mené au sein du laboratoire Systèmes et Transports (SeT), de l'Université de Technologie de Belfort-Montbéliard (UTBM), sous la direction de Monsieur Alexandre CAMINADA et de Monsieur Hakim MABED. Je leur adresse ma profonde gratitude pour m'avoir permis d'intégrer leur structure dans le cadre de mon doctorat. Sans eux, ce travail n'aurait pas été ce qu'il est. Merci à Alexandre pour son encadrement, son soutien, sa confiance, sa générosité et sa disponibilité (Alexandre je ne peux pas te remercier par une simple phrase, tu es pour moi beaucoup plus qu'un directeur de thèse). Merci à Hakim, qui compte beaucoup pour moi, pour son écoute, ses encouragements et son amitié.

Je remercie chaleureusement l'ensemble des membres du jury pour le grand intérêt qu'ils ont porté à mon travail. Je remercie les rapporteurs pour avoir accepté la charge de travail qu'implique la lecture critique du présent mémoire. Merci à Michel VASQUEZ pour les nombreux commentaires qu'il m'a apportés et pour la précision de ses remarques qui m'ont permis de corriger certaines de mes erreurs. Merci à Christine SOLNON pour les remarques pertinentes et les conseils avisés qui nous ont amené à de nouvelles réflexions et perspectives de recherche. Je remercie également Clarisse DHAENENS et Patrick SIARRY pour l'attention qu'ils ont portée à mon travail et pour les discussions fructueuses que j'ai eues avec eux. Ce travail aurait eu peu d'intérêt sans Thierry DEFAIX et plus généralement la délégation générale de l'armement qui nous ont fourni un problème réel à traiter. Je remercie également Jean-François LEGENDRE (plus généralement SILICOM) pour sa disponibilité, son aide, ses analyses et ses validations.

Je remercie d'une façon plus générale, tous les membres de notre équipe avec qui j'ai pu échanger et travailler ; tout spécialement Jean-Noël MARTIN, Enseignant à l'UTBM, Sid LAMROUS et Oumaya BAALA maitres de conférences à l'UTBM, Chibli JOUMAA actuellement enseignant au kuweit et Alexandre GONDRAN qui est actuellement maitre de conférences à l'ENAC dont j'ai apprécié chez eux autant leurs compétences techniques que leurs qualités humaines.

Enfin, je remercie les enseignants et administratifs de l'UTBM, tous les membres du laboratoire SET ainsi que tous mes amis. Un très grand merci à Rouwaida ABDALLAH pour les longues discussions avec elle, pour son écoute, son intelligence et ses qualités humaines.

Enfin, mes parents, mes frères et mes sœurs qui ont toujours cru en moi, m'ont soutenu et encouragé tout au long de ce doctorat et qui sont fiers du chemin parcouru.

A mon adorable mère qui m'a beaucoup donné,

1. Introduction

1.1. Contexte de travail

Au cours de ces quinze dernières années, les systèmes de communications hertziens ont connu une croissance exceptionnelle. Etant donné que le spectre radioélectrique est une ressource naturelle limitée et qui fait l'objet de plus en plus de demandes en raison du déploiement rapide de nouveaux services de radiocommunication, les opérateurs doivent optimiser les ressources fréquentielles des systèmes qu'ils utilisent. La réutilisation des fréquences dans un réseau de radiocommunications est la meilleure stratégie pour faire de réelles économies de spectre. Cependant elle entraine immédiatement des pertes de qualité car l'utilisation de la même fréquence ou de fréquences très proches sur le spectre cause des interférences entre les liaisons allant même jusqu'à brouiller complètement le signal. La réutilisation des fréquences demande donc la mise au point d'outils performants d'allocation de ressources. Le modèle théorique le plus proche de ces problèmes est probablement la coloration de graphes pour laquelle on cherche à minimiser le nombre de couleurs utilisées ou la *k*-coloration de graphes pour laquelle on cherche à minimiser le nombre de contraintes insatisfaites pour *k* couleurs.

Depuis ses premières apparitions dans le projet européen CALMA (*Combinatorial ALgorithms for Military Applications*) pour le déploiement de réseaux militaires, le problème d'affectation de fréquences FAP (*Frequency Assignment Problem*) est présenté à travers diverses modélisations. Au début le but du FAP était de trouver une solution qui respecte un ensemble de contraintes d'écarts entre les liaisons. Une deuxième modélisation FAPI a permis de prendre en compte les contraintes d'intermodulation. Une autre version FAPG a été proposée pour prendre en compte des contraintes globales entre les liaisons. Puis FAPP a introduit la notion de polarisation des fréquences où le but n'était plus seulement d'affecter à une liaison hertzienne une fréquence mais un couple de ressources (fréquence, polarisation). Chacun de ces problèmes se définit pour différents systèmes allant des systèmes point-à-point (faisceaux hertziens), aux réseaux de radios (PR4G) en passant par les réseaux cellulaires (GSM). La plupart du temps, les systèmes introduisent des paramètres spécifiques sur les modes de calcul des interférences et sur les seuils de qualité attendus mais ils conservent à peu près tous les mêmes types de contraintes et d'objectifs.

Dans le cadre d'un projet de recherche sur 3 ans avec le CELAR (*Centre Electronique de l'ARmement* en France), nous avons considéré une nouvelle modélisation du FAP plus complète vis-à-vis des phénomènes de brouillage et plus complexe vis-à-vis des objectifs. Le but de ce projet était de fournir une méthode ou un ensemble de méthodes au sein d'un seul outil capable de répondre à différents problèmes décrits par un cahier des charges. L'outil fourni devait pouvoir remplacer un ensemble d'outils existants tout en respectant un ensemble d'exigences sur les performances en termes de non régression sur la qualité des solutions produites. La nouvelle modélisation du FAP proposée par le CELAR inclut des contraintes de tous types, unaires, binaires et n-aires, avec des notions de contraintes dures à respecter impérativement et de contraintes souples à différents niveaux de relâchement. Elle propose aussi plusieurs modes d'optimisation tel que trouver une solution réalisable, i.e. qui satisfait toutes les contraintes d'un problème, jusqu'à l'optimisation de critères sur le spectre ou sur le nombre de contraintes violées pour les instances non réalisables. Afin d'évaluer les méthodes proposées, le CELAR nous a fourni pour ce projet un ensemble de 30 instances issues de cas

réels ou générées automatiquement comprenant des instances de petites tailles (50 variables et quelques centaines de contraintes) et de très grandes tailles (3 000 variables et 300 000 contraintes).

1.2. Motivations et principales contributions

De l'élaboration d'emploi du temps au problème du voyageur de commerce, les chercheurs sont confrontés à des problèmes difficiles à résoudre même avec les meilleures méthodes d'optimisation connues. En particulier les problèmes de nature combinatoire ou dénombrable possèdent un nombre fini de configurations possibles, mais très grand, de sorte qu'une résolution par une simple énumération est non concevable en pratique.

Pour un problème d'optimisation combinatoire sous contraintes, l'objectif est de trouver une configuration qui satisfasse les contraintes et optimise une fonction donnée parmi l'ensemble des configurations possibles. Les problèmes ne possèdent pas tous la même difficulté. Des classes de difficultés ont été définies afin de cerner la difficulté d'un problème donné. Lorsque le problème à résoudre est NP-difficile, une méthode exacte qui garantit l'obtention de la meilleure solution peut nécessiter un temps de calcul très long qui croit de façon exponentielle avec la taille du problème traité. Dans ce cas, l'utilisation d'une méthode approchée peut s'avérer pratique pour obtenir une solution que l'on espère de bonne qualité, la meilleure possible en tout cas, en un temps raisonnable. La notion de temps raisonnable ne veut absolument rien dire en général, pour un problème particulier le temps de recherche *raisonnable* de la méthode est fixé par l'utilisateur. Pour un problème NP-difficile, il peut très bien y avoir des instances faciles et d'autres difficiles à résoudre. Intuitivement on peut penser que la difficulté d'une instance est proportionnelle du nombre de variables, du nombre de ressources par variable et du nombre de contraintes. La réalité est bien plus complexe, c'est une fonction de l'ensemble de ces facteurs qui rend une instance facile ou difficile, la taille ne suffit pas.

Un très grand nombre de problèmes combinatoires réels et théoriques appartient à la famille des problèmes de satisfaction de contraintes (*Constraint Satisfaction Problem* ou *CSP*) : configuration, planification, ordonnancement, affectation de ressources... [97]. Ces problèmes partagent une description commune, basée sur un formalisme très simple, qui autorise en général une modélisation claire et intuitive. Un CSP [31][32] est défini par un ensemble X de variables, un ensemble D de domaines de définition qui encadrent les valeurs que peuvent prendre ces variables, et un ensemble C de contraintes qui conditionnent les valeurs que pourront prendre les variables appartenant à X. Les enjeux de la résolution de ces problèmes sont très importants, sur le plan scientifique pour produire du savoir algorithmique, et sur le plan économique pour améliorer la performance de systèmes de toutes natures.

La conception et le développement de méthodes efficaces de résolution des CSP est une question ouverte. Il existe actuellement des solveurs sous licences commerciales pour résoudre de nombreux problèmes mais il y a de plus en plus de travaux de recherche pour proposer de nouveaux algorithmes, plus rapides ou plus performants en qualité de solutions. Les méthodes sont classées essentiellement en deux familles. La première, qui est la base de la Programmation Par Contraintes (PPC), englobe les méthodes dites complètes ou exactes, qui peuvent prouver la réalisabilité d'un problème, ou son contraire le cas échéant. En général, ces méthodes permettent également d'extraire l'ensemble des solutions d'un problème. La deuxième famille englobe les méthodes dites incomplètes ou approchées qui ont été proposées pour faire face à l'explosion combinatoire sans garantir l'optimalité. Ce genre de méthode aborde généralement la résolution d'un CSP comme un problème d'optimisation combinatoire

pour lequel il s'agit de chercher une configuration satisfaisant le plus grand nombre de contraintes, l'objectif visé étant de les satisfaire toutes. Dans ce cas, on parle aussi de problème de satisfiabilité maximale (MaxCSP).

D'un point de vue algorithmique, les méthodes exactes reposent souvent sur l'utilisation d'une recherche arborescente. Elles manipulent des configurations partielles, en partant d'une configuration vide (aucune variable n'est instanciée), et construisent la solution en instanciant les variables une à une sans violer de contraintes. Si une affectation conduit à un viol de contrainte, des mécanismes de retours sur les décisions d'affectations antérieures sont nécessaires (*backtrack*). Améliorer ce genre de méthode revient souvent à prévenir des mauvais choix et à détecter les incohérences en utilisant la propagation des contraintes pour réduire l'espace de recherche, à réduire le nombre de *backtracks* en faisant des *backtracks* intelligents et à apprendre pendant la recherche afin d'éviter de répéter les mêmes erreurs (mémorisation de coupes ne conduisant pas à une solution via les *nogoods*). Du fait de la démarche constructive, ces méthodes ont plutôt tendance à intensifier la recherche dans une direction en étant guidée de proche en proche par les réductions de domaines dues aux contraintes. Si cette direction est mauvaise, le prix à payer pour en changer peut être très lourd (nombreux retours nécessaires).

Les méthodes approchées manipulent généralement des configurations complètes en partant d'une configuration initiale où toutes les variables du problème sont instanciées. Puis, elles essayent de réparer cette solution de proche en proche jusqu'à atteindre un critère d'arrêt prédéfini. Ce sont des méthodes qui fonctionnent selon une approche de recherche dite locale ou par voisinage ou par réparation. Deux grands mécanismes ont été proposés pour améliorer l'efficacité de ce genre de méthode, l'intensification et la diversification. L'intensification consiste à fouiller dans une zone précise de l'espace de recherche afin de trouver un optimum local tandis que la diversification permet de se déplacer vers une autre zone de recherche non encore visitée. Ne s'appuyant pas sur la réduction des domaines pour guider sa recherche, l'efficacité d'une méthode incomplète dépend beaucoup de l'alternance de ces deux mécanismes d'une façon intelligente.

On voit donc que les procédés ne sont pas les mêmes et qu'il peut y avoir des enseignements à tirer à la fois de la réduction des domaines et de la recherche locale. Dans la littérature, une activité intense existe depuis une dizaine d'années sur l'hybridation entre des approches complètes et incomplètes, et particulièrement entre la recherche locale et la programmation par contraintes. Des progrès importants ont été réalisés sur la résolution de certains problèmes et ont même parfois donné lieu à la conception de méthodes assez génériques telles *CN-Tabu* et *Decision-Repair* toutes deux basées sur des configurations partielles consistantes. Notre travail se situe dans ce cadre ; cette thèse est consacrée à l'élaboration d'une nouvelle méthode hybridant la recherche locale et la programmation par contraintes. L'analyse de diverses méthodes que nous présentons dans le premier chapitre nous a permis d'identifier des mécanismes complémentaires, nous les avons ensuite assemblés pour proposer la méthode *Tabu-NG* soit *Tabu Search based on NoGood*.

Pour en situer brièvement le contexte vis-à-vis de l'existant on pourrait dire que cette méthode apporte le bénéfice de l'apprentissage des erreurs par rapport à *CN-Tabu* et le bénéfice de la diversification par rapport à *Decision-Repair*. Bien entendu cette méthode dans sa version issue de notre travail ne prétend pas être un solveur universel, cependant nous avons tenté de conserver une certaine généricité dans ses principes pour en rendre l'application possible pour un grand nombre de problèmes. *Tabu-NG* manipule des configurations partielles consistantes ; elle applique le filtrage et la propagation de contraintes sur chaque nœud de l'arbre afin de réduire l'espace de recherche et guider efficacement le

processus de recherche d'une solution ; elle mémorise et utilise des *nogoods* pour expliquer les échecs et réaliser des coupes dans l'espace ; elle utilise une recherche locale afin d'atteindre une nouvelle configuration à partir d'une configuration partielle consistante ; et elle utilise aussi une recherche locale pour réparer une configuration partielle inconsistante. Sur ces différents éléments il existe des options ou des paramètres à positionner en fonction du problème. Par exemple les recherches locales peuvent inclure des heuristiques spécifiques du problème pour en améliorer l'efficacité.

Le contexte du projet dans lequel ces travaux ont été réalisés a bien évidemment été de grande importance pour le développement de la méthode. La réalisabilité étant un élément clef de performance pour quasiment tous les problèmes, il était important de concevoir une méthode centrée sur la construction de solution consistante. Le filtrage et la propagation de contraintes avaient donc un rôle fondamental pour cet objectif. Par ailleurs le cas des problèmes d'optimisation de fonction en plus de la réalisabilité pour certains niveaux de relâchement des contraintes, fait que nous devions aussi proposer une solution permettant de diversifier la recherche en s'appuyant sur une recherche locale. L'hybridation avec la Recherche Tabou nous a paru être le meilleur choix au vu des résultats présentés sur d'autres versions du FAP.

Nous avons appliqué *Tabu-NG* sur deux problèmes très différents, un problème réel, le problème d'affectation de fréquences, et un problème académique théorique, la coloration de graphes. Le contexte du projet nous a permis de comparer la méthode avec les meilleurs scores générés par le CELAR à l'aide de différentes méthodes dont il disposait, mais ce n'était pas suffisant pour situer l'apport de la méthode. Nous avons donc aussi utilisé les instances très référencées du projet CALMA sur d'autres problèmes FAP pour comparer les performances de *Tabu-NG*, notamment sans contraintes n-aires. Enfin nous avons choisi de réaliser une version de la méthode pour le coloriage de graphe en utilisant des instances DIMACS très référencées elles-aussi.

1.3. Organisation de la thèse

Le manuscrit se décompose en trois chapitres. Le premier présente tout d'abord un état de l'art sur les méthodes de résolution de CSP ainsi que les techniques de propagation utilisées pour réduire l'espace de recherche, puis nous présentons la méthode *Tabu-NG* dans son contexte général avec ses grandes composantes algorithmiques. Ainsi nous présentons le formalisme CSP et ses caractéristiques avec des exemples, ensuite les notions de consistance et les méthodes de propagation de contraintes existantes, puis la résolution des CSP par des méthodes exactes avec les techniques d'améliorations ajoutées. Nous présentons aussi les métaheuristiques de résolution et en particulier quelques méthodes de recherche locale. Ensuite nous argumentons sur l'intérêt des méthodes proposant des hybridations entre propagation de contraintes et recherche locale. La partie sur *Tabu-NG* donne les composantes de la méthode, avec le filtrage et la propagation, les différentes listes Tabou employées, les mécanismes d'extension et de réparation des configurations.

Le deuxième chapitre est consacré à l'utilisation de *Tabu-NG* pour des problèmes d'affectation de fréquences. Nous donnons tout d'abord la description physique et mathématique de ces problèmes, avec un état de l'art sur les modèles de référence et les méthodes de résolution existantes. Ensuite, nous présentons l'application de *Tabu-NG* avec le détail de ses composantes et de ses paramètres sur les problèmes d'affectation de fréquences avec contraintes binaires connus sous le nom des instances CELAR et GRAPH. Puis, nous décrivons l'application de *Tabu-NG* sur les FAP plus complexes fournis par le CELAR dans

le cadre du projet de recherche mentionné dans cette introduction. Pour les deux cas, nous donnons le détail des algorithmes implémentés, nous présentons les instances traitées, les stratégies d'optimisation et un ensemble de résultats et d'analyse justifiant les choix adoptés.

Le troisième chapitre de la thèse décrit les travaux réalisés sur l'adaptation de *Tabu-NG* pour traiter les problèmes de coloration de graphe. Nous commençons par un bref rappel sur les problèmes de coloration de graphe et sur quelques notions liées à la coloration. Puis nous présentons les instances utilisées ainsi qu'un bilan sur les meilleures méthodes de résolution actuelles. Ensuite, nous présentons les adaptations faites sur *Tabu-NG* afin d'améliorer son efficacité et de bénéficier de la nature du nouveau problème à traiter. Enfin, l'application de *Tabu-NG* sur les instances de coloration ainsi que les paramètres utilisés sont donnés avec un ensemble de résultats, d'analyse et de justifications.

La thèse se termine par une conclusion générale qui reprend les travaux de l'ensemble des chapitres et quelques perspectives majeures pour la poursuite de ce travail.

2. Introduction aux CSP et à *Tabu-NG*

L'objectif de ce premier chapitre est de situer et de comprendre le formalisme CSP, les mécanismes de propagation liés ainsi que les méthodes de résolution existantes. Nous introduisons aussi dans ce chapitre une nouvelle méthode appelée *Tabu-NG* pour la résolution de CSP. Ce chapitre est divisé en trois parties distinctes. Dans un premier temps, le modèle CSP et ses caractéristiques sont introduits. Puis nous présentons les techniques de propagation de contraintes et les méthodes de résolution de CSP ; nous mettons l'accent sur l'hybridation de la recherche locale avec la programmation par contraintes. Dans la troisième partie de ce chapitre, nous proposons la méthode hybride *Tabu-NG* en décrivant ses principes généraux et ses différentes composantes en restant dans un contexte hors application.

Une partie de ce travail a été publiée dans la conférence CIMSim (*Computanional Intelligence, Modelling and Simulation*) en 2010.

Sommaire

2. Introduction aux CSP et à *Tabu-NG* 20

2.1. Les problèmes de satisfaction de contraintes 21
2.2. Méthodes de résolution de CSP.....27
2.3. *Tabu-NG* : *Tabu Search based on NoGoods*, une nouvelle méthode hybride.....54
2.4. Synthèse 64

2.1. Les problèmes de satisfaction de contraintes

Un très grand nombre de problèmes réels et théoriques appartiennent à la famille des problèmes de satisfaction de contraintes (*Constraint Satisfaction Problem CSP*) : configuration, planification, ordonnancement, affectation de ressources... [97]. Les CSP ont la particularité commune d'être fortement combinatoire ; bon nombre d'entre eux font partie de la classe des problèmes NP-complets [80][81] et donc induisent des complexités algorithmiques élevées. De ce fait, un très grand nombre de travaux ont été menés pour concevoir et développer des méthodes de résolution les plus efficaces possible.

Dans ce chapitre, nous proposons une introduction aux CSP, puis nous voyons les méthodes de résolution basées sur la programmation par contraintes (PPC), les métaheuristiques de résolution ainsi qu'une introduction sur les résolutions hybrides. A la fin de ce chapitre, nous proposons une nouvelle méthode de résolution hybride qui est construite à partir de combinaison d'autres méthodes. A noter que nous limitons notre présentation aux CSP à domaines finis tels qu'introduits dans [79].

2.1.1. Le modèle CSP et ses principales caractéristiques

Un problème de satisfaction de contraintes (CSP) [31][32] est défini par un ensemble de variables, des domaines de définition qui encadrent les valeurs que peuvent prendre les variables, et un ensemble de contraintes qui conditionnent les valeurs que pourront prendre les variables.

Définition 1 : CSP

Un problème de satisfaction de contraintes (*CSP*) [31][32] est défini par un triplé $P=(X, D, C)$ où :

- $X = \{x_1, x_2, \ldots, x_n\}$ est l'ensemble fini des n variables du problème.
- $D = \{D_{x1}, \ldots, D_{xn}\}$ est l'ensemble des n domaines finis pour les variables. D_{xi} est l'ensemble des valeurs possibles pour la variable x_i. Notons d la taille du plus grand domaine.
- $C = \{c_1, \ldots, c_m\}$ est l'ensemble des m contraintes qui lient les variables entre elles.

Les contraintes peuvent être exprimées sous différents formes : formules mathématiques, tables de valeurs compatibles, etc. Elles sont des relations entre les variables qui définissent la structure du problème à résoudre.

Définition 2 : Contrainte

Une contrainte $c \in C$ sur les variables $x_{i1},\ldots, x_{ik}$ pour $i_1, \ldots, i_k \in \{1, \ldots, n\}$ est une relation mathématique entre ces variables. On note *var(c)* l'ensemble des variables intervenant dans la contrainte c.

Définition 3 : Arité

L'arité d'une contrainte $c \in C$ est donnée par la cardinalité de *var(c)* ; autrement dit le nombre de variables sur lequel porte *c*. Une contrainte *c* est dite :

- unaire si son arité est égale à 1
- binaire si son arité est égale à 2
- n-aire si son arité est égale à n

Un CSP est dit binaire, si toutes ses contraintes sont d'arité inférieure ou égale à deux.

La résolution d'un CSP consiste à affecter des valeurs aux variables du problème, de telle sorte que toutes les contraintes soient satisfaites. On introduit les notations et les définitions suivantes.

Définition 4 : Affectation ou Configuration

On appelle affectation ou configuration le fait d'instancier des variables par des ressources prises dans leur domaine respectif. Une configuration est une fonction :

$$S : X \rightarrow \prod_{i=1}^{n} D_{x_i}$$

Telle que $S(x_i) \in D_{x_i}$ pour i $\in$ [1..n].

Une configuration est dite partielle si elle porte sur un sous ensemble *Y* de *X*. Elle est dite complète si elle porte sur tout l'ensemble *X*.

Définition 5 : Configuration partielle consistante ou cohérente

Une configuration partielle qui porte sur un ensemble $Y \subset X$ est dite consistante localement ou cohérente si elle satisfait toutes les contraintes sur *Y*.

Définition 6 : Variable libre

Une variable est dite libre si elle ne fait pas l'objet d'une affectation ; elle n'est pas instanciée.

Définition 7 : Espace de recherche d'un CSP

L'espace de recherche d'un CSP est l'ensemble des configurations possibles que l'on notera *E*.

$$E = D_{x1} \times D_{x2} \times \ldots \times D_{x_n}$$

L'espace de recherche est égal au produit cartésien de l'ensemble des domaines des variables ; une configuration *S* sera assimilée à un élément de cet ensemble *E*.

Définition 8 : Solution réalisable

Une solution réalisable, appelée parfois simplement une solution, pour un CSP P est une configuration complète $s \in E$ qui satisfait toutes les contraintes du problème. On note Sol(P) l'ensemble des solutions réalisables :

$$Sol(P) = \{s \in E \mid \forall c \in C, s \text{ satisfait } c\}$$

Définition 9 : Graphe et hypergraphe de contraintes

Un hypergraphe *(X, C)* de contraintes pourra être associé à chaque CSP. Ce hypergraphe pourra être obtenu en remplaçant les variables par des sommets et les contraintes par des hyper-arêtes. Chaque hyper-arête relie l'ensemble des variables de la contrainte à laquelle elle correspond.

Dans le cas d'un CSP binaire, *(X, C)* est alors dit graphe de contraintes ou réseau de contraintes. Une contrainte portant sur les variables x_i et x_j, notée c_{ij}, se représente sur le graphe *(X, C)* par une arête qui lie les sommets x_i et x_j.

Définition 10 : Deadend

Soit *I=S(Y)* une configuration partielle consistante pour un ensemble de variables $Y \subset X$. *I* est une impasse, ou un *deadend*, s'il existe une variable x_j non instanciée ($x_j \in X \backslash Y$) telle que toutes les extensions de *I*, notées $\{(I, v_j), \forall v_j \in D_j\}$, sont inconsistantes.

Dans le cas où un algorithme de filtrage est utilisé, un *deadend* est une configuration partielle telle que le domaine d'une variable libre devient vide.

Définition 11 : Nogood

Une configuration partielle consistante *I* est un *nogood* si elle ne peut pas être étendue en une solution réalisable.

Toute configuration partielle n'apparaissant dans aucune solution réalisable du problème est dite *nogood.*

Toute configuration partielle qui est un *deadend* est aussi un *nogood*. Par contre, un *nogood* n'est pas forcément un *deadend*.

Définition 12 : Nogood minimal

Un *nogood* minimal est un *nogood* n'incluant pas lui-même un autre *nogood*.

Exemple 1 (CSP Simple)

Soient :

- X = {*x*, *y*, *z*, *w*}

- $D_x = D_y = D_z = D_w = \{0,1\}$
- $C = \{x \neq z, x+z=y, x+y=2, x+w=2\}$

Ce CSP comporte 4 variables *x, y, z et w,* chacune pouvant prendre 2 valeurs (0 ou 1). Une solution possible pour ce problème est {*x=1, y=1, z=0, w=1*}.

Pour cet exemple, la configuration partielle I = {x=0, y=1} est un *deadend* et donc I est aussi un *nogood,* mais I n'est pas un *nogood* minimal car on a le sous ensemble {x=0} de I qui est aussi un *nogood*. Aucune solution réalisable pour ce problème ne peut contenir le sous ensemble {x=0} qui est un *nogood* minimal.

2.1.2. Les problèmes d'optimisation sous contraintes

Résoudre un CSP consiste à trouver une solution réalisable, ou éventuellement un nombre donné ou toutes les solutions réalisables. Un problème d'optimisation sous contraintes est un problème pour lequel on cherche parmi l'ensemble de toutes les solutions réalisables la meilleure solution selon une fonction qui définit un objectif donné. La fonction a pour rôle d'évaluer la qualité d'une solution et les contraintes d'éliminer les configurations qui ne sont pas des solutions. Le problème est alors double : le premier consiste à trouver l'ensemble de toutes les solutions réalisables et le deuxième à trouver dans cet ensemble la meilleure solution qui maximise ou minimise la fonction.

Définition 13 : Problèmes d'optimisation sous contraintes (CSOP) [31]

Etant donné un CSP $P=(X, D, C)$ et E l'espace de recherche défini par les domaines D. Soit f une fonction objectif de E vers un ensemble K.

Un problème d'optimisation sous contraintes est défini comme la maximisation (respectivement la minimisation) de la fonction f dans l'espace de recherche restreint aux solutions réalisables. Résoudre à l'optimalité ce problème consiste à trouver une solution réalisable s de plus grande valeur (respectivement plus petite) pour $f(s)$. Formellement :

$$s \in Sol(P) / \forall s' \in Sol(P),\ f(s') \leq f(s)$$

$$(Respectivement\ f(s') \geq f(s))$$

2.1.3. Exemple de problèmes et formalisations

De nombreux problèmes théoriques et réels peuvent se modéliser sous forme de CSP. Dans cette section, nous donnons quelques problèmes typiques et leurs CSP équivalents.

Le formalisme CSP est assez générique et abstrait, il n'impose aucune limitation sur la nature et le nombre de contraintes, ni sur les types des domaines. La modélisation d'un problème sous forme de CSP est très simple, il suffit d'identifier :

1. Les variables
2. Les domaines des valeurs des variables
3. Les contraintes entre les variables

Un problème possède souvent plusieurs modélisations possibles. Choisir un bon modèle parmi les possibles est très important. Les critères de choix sont la simplicité et l'efficacité par

exemple vis-à-vis de la taille de l'espace de recherche engendré par le modèle, sa simplicité d'expression, la rapidité d'évaluation des configurations...

Après la phase de modélisation d'un problème en CSP, il faut le résoudre. Un algorithme de base qui énumère toutes les combinaisons possibles peut évidemment résoudre tous les problèmes formulés en CSP, cependant ce type d'algorithme ne peut résoudre que les problèmes de petite taille en temps raisonnable. Il faut donc des méthodes plus efficaces que de simple méthode énumérative.

2.1.3.1. Coloration de carte

Le coloriage de carte est un cas spécial du coloriage de graphe. Ce problème peut se modéliser facilement sous forme d'un CSP. La résolution de ce problème vise à colorier avec un ensemble donné de couleurs chaque région d'une carte de telle manière que deux régions voisines ne soient pas de la même couleur.

Exemple : Coloriage de la carte d'Australie

La question qui se pose est la suivante : peut-on colorier la carte des états de l'Australie avec trois couleurs ?

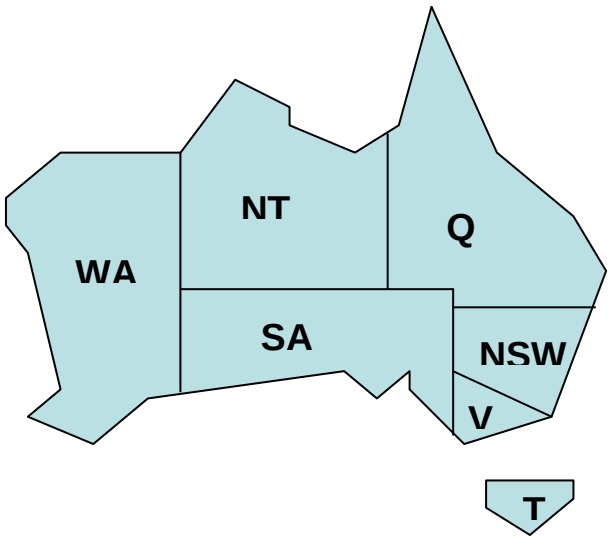


Figure 1 : Carte de l'Australie

La carte en Figure 1 comporte 7 régions devant être coloriées en rouge, bleu et vert. Pour obtenir le CSP équivalent, chaque région sera représentée par une variable. A chaque variable on attribue un domaine qui correspond à l'ensemble des couleurs. Pour chaque paire de régions adjacentes, une contrainte binaire est créée entre les variables correspondantes. Cette contrainte interdit d'affecter une couleur identique à ces deux variables.

Formellement :

- X = {WA, NT, SA, Q, NSW, V, T}
- D = {D_{WA}, D_{NT}, D_{SA}, D_Q, D_{NSW}, D_V, D_T} avec $D_{WA} = D_{NT} = D_{SA} = D_Q = D_{NSW} = D_V = D_T$ = {rouge, bleu, vert}

- C = {WA ≠ NT, WA ≠ SA, NT ≠ SA, NT ≠ Q, SA ≠ Q, SA ≠ NSW, SA ≠ V, Q ≠ NSW, NSW ≠ V}

Une solution réalisable pour ce problème est la suivante (voir Figure 2) :

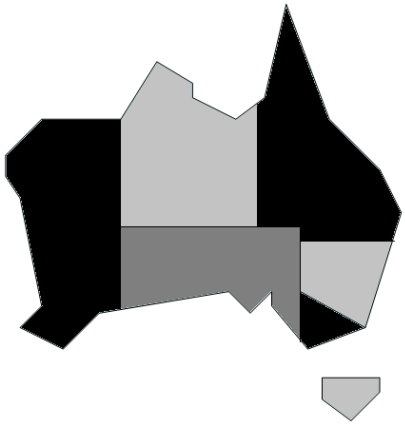


Figure 2 : Solution possible pour le coloriage de la carte de l'Australie

2.1.3.2. Placement des n-reines

Le problème des n-reines consiste à placer *n* reines sur un échiquier de *n*n* cases de telle manière qu'aucune reine ne soit pas prise par une autre. Le but est donc de placer une seule reine sur la même ligne, colonne et diagonale.

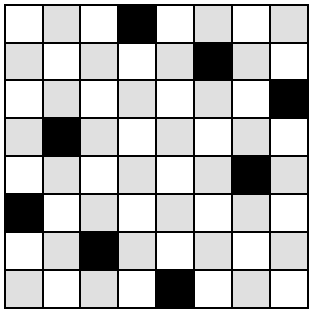


Figure 3 : Une solution réalisable pour le placement de 8 reines sur un échiquier 8*8

Il y a plusieurs façons de modéliser ce problème sous forme d'un CSP. Une possibilité est de considérer chaque colonne i comme une variable x_i. Une variable possède alors un domaine de (1 à n) qui désigne le numéro de ligne où se place la reine de la colonne.

Formellement, et pour le problème avec 8 reines, nous avons :

- $X = \{x_1, x_2, \ldots, x_8\}$
- $D = \{D_{x1}, D_{x2}, \ldots, D_{x8}\}$ avec $D(x_i) = \{1, \ldots, 8\}$
- C est défini par, $\forall i$, $\forall j \neq i$: les reines doivent être sur des lignes différentes, soit $x_i \neq x_j$, et les reines doivent être sur des diagonales différentes, soit $x_i + i \neq x_j + j$ et $x_i - i \neq x_j - j$

Une solution réalisable de ce problème est illustrée en Figure 3.

2.1.3.3. Jeu Sudoku

Le jeu Sudoku est un jeu de logique qui consiste à compléter une grille $n*n$, avec $n=p^2$ et $p>=2$, avec les chiffres de 1 à n. Les contraintes sont qu'un chiffre ne doit pas être utilisé deux fois par ligne, par colonne et par carré de n cases.

Prenons l'exemple illustré en Figure 4. Le CSP équivalent de cette grille pourra être obtenu en représentant chaque case par une variable ; si la case est vide, le domaine de la variable est [1,…,9] ; si la case est pré-affectée, le domaine de la variable est restreint à la valeur pré-affectée. On définit des contraintes de différences pour toutes les cases qui se trouvent dans les mêmes carrés de 3*3, sur les mêmes lignes et sur les mêmes colonnes.

Formellement et pour l'exemple, le CSP équivalent est :

- $X = \{x_{11},..., x_{99}\}$
- $D = \{D_{x11},..., D_{x99}\}$
- $C =$
  $x_{11}\neq x_{12},..., x_{11}\neq x_{19}, x_{21}\neq x_{22},..., x_{91}\neq x_{99}$ (contraintes de ligne)
  $x_{11}\neq x_{21},..., x_{11}\neq x_{91}, x_{12}\neq x_{22},..., x_{19}\neq x_{99}$ (contraintes de colonne)
  $AllDiff(x_{11},...,x_{33}),\ldots, AllDiff(x_{77},...,x_{99})$ (contraintes de carré)

	1	2	3	4	5	6	7	8	9
1	**8**	**6**			**2**				
2				**7**				**5**	**9**
3									
4					**6**		**8**		
5		**4**							
6			**5**	**3**					**7**
7									
8		**2**					**6**		
9			**7**	**5**		**9**			

Figure 4 : Exemple d'une grille SUDOKU 9*9 à compléter

2.2. Méthodes de résolution de CSP

La formalisation CSP est une modélisation abstraite permettant la représentation d'un très grand nombre de problème. Il n'existe pas actuellement une méthode universelle capable de résoudre tous les CSP d'une manière efficace. De nombreuses techniques et méthodes ont été conçues, développées et testées pour résoudre des CSP avec plus ou moins de succès selon la

nature des problèmes. Dans la section suivante nous présenterons différentes techniques et méthodes de résolution :

- Les techniques basées sur les notions de consistance avec les algorithmes de filtrage et de propagation de contraintes.
- Les méthodes de résolution complètes, de la méthode la plus basique qui consiste à l'énumération de toutes les combinaisons possibles vers des méthodes plus élaborées qui utilisent les notions de consistance et de filtrage.
- Les métaheuristiques qui sont des méthodes de résolution incomplètes souvent capables de donner une bonne solution en un temps acceptable.
- Les méthodes hybrides qui combinent des techniques issues de la programmation par contraintes (PPC) avec des méthodes incomplètes comme la recherche locale et les algorithmes génétiques.

2.2.1. Méthodes de propagation de contraintes

2.2.1.1. Notions de consistance

Résoudre un CSP consiste à trouver une affectation pour chaque variable satisfaisant l'ensemble de contraintes. L'idée principale de la consistance est la réduction de l'espace de recherche afin de rendre moins difficile la résolution du problème.

Les méthodes de résolution complètes essayent de supprimer des valeurs, dites inconsistantes, dans les domaines des variables correspondants. La notion de consistance est donc liée à la notion de contrainte. En effet, le rôle d'une contrainte *c* est de forcer les variables à pouvoir prendre certaines valeurs, la consistance intervient dans le cas où une valeur ne pourra dans un aucun cas satisfaire *c*. La propriété de consistance est atteinte dans le cas où aucune valeur ne pourra pas être supprimée.

Une notion de consistance est associée à chaque type de contrainte. La consistance de nœud ou de sommet pour les contraintes unaires, l'arc consistance pour les contraintes binaires et la consistance de chemin ainsi que la k-consistance pour les contraintes n-aires.

2.2.1.2. Consistance de nœud

La consistance de nœud ne concerne que les contraintes unaires. Etant donné un CSP *(X, D, C)*, pour le rendre consistant de nœud il faut et il suffit pour chaque contrainte unaire de *C* portant sur les variables de supprimer toutes les valeurs inconsistantes dans le domaine de ces variables.

Définition 14 : Consistance de Nœud

Un CSP *(X, D, C)* est consistant de nœud si pour toute variable x_i de *X* et pour toute valeur *v* de D_{x_i}, l'affectation (x_i, *v*) satisfait toutes les contraintes unaires de *C*.

2.2.1.3. Consistance d'arc

La consistance d'arc concerne les contraintes binaires, elle est dite aussi 2-consistance. Une contrainte binaire qui porte sur deux variables x_i et x_j est dite arc consistante, si chaque valeur dans D_{x_i} possède au moins une valeur dans D_{x_j} tels que ces deux valeurs vérifient les contraintes sur le couple (x_i, x_j).

Définition 15 : Consistance d'arc

Soit un CSP *(X, D, C)* et une contrainte binaire $c \in C$ qui porte sur deux variables x_i et x_j. La contrainte *c* est arc consistante si :

1. $\forall a \in D_{x_i}, \exists b \in D_{x_j}, (a,b)$ vérifie la contrainte *c*
2. et $\forall a \in D_{x_j}, \exists b \in D_{x_i}, (a,b)$ vérifie la contrainte *c*

Un CSP est arc consistant si toutes ses contraintes binaires sont arc consistantes.

Exemple 2 : Consistance d'arc

Soit le CSP suivant :

- X = {*x*, *y*}
- $D_x = D_y$ = {0, 1, 2, 3, 4, 5}
- *C* = {*x*+2 < *y*}

Ce CSP comporte 2 variables *x* et *y*, chacune pouvant prendre 6 valeurs de 0 à 5. Ce CSP n'est pas arc consistant car en considérant la valeur 5 dans D_x, aucune valeur *b* dans D_y ne vérifie la contrainte 5+2 < *b*

2.2.1.4. Consistance d'hyper-arc

L'arc consistance ne concerne que les contraintes binaires, tandis que la consistance hyper-arc généralise cette notion pour les contraintes n-aires.

Définition 16 : Consistance d'hyper-arc

Soit un CSP *(X, D, C)* et une contrainte n-aire $c \in C$ qui porte sur *n* variables x_1, $x_{2,\ldots,}$ x_n. La contrainte *c* est hyper-arc consistante si :

1. $\forall i \in [1..m], \forall a \in D_{x_i}, \exists$ un n-uplet *d* avec $a \in d$ et *d* vérifie la contrainte *c*.

Un CSP est hyper-arc consistant si toutes ses contraintes sont hyper-arc consistantes.

2.2.1.5. La k-consistance

La k-consistance est une généralisation de toutes les notions de consistances précédentes. Cooper a présenté dans [25] un algorithme capable d'établir la *k-consistance*.

Définition 17 : k-consistance

Soit un CSP *(X, D, C)*. Etant donné un ensemble de variables $Y \subseteq X$ avec $|Y| = k-1$. Une configuration partielle *I* sur *Y* est dite *k-consistante* si et seulement si pour toute $k^{ème}$ variable $x_k \in X \setminus Y$, il existe une valeur $v_k \in D_{x_k}$ telle que $I \cup \{(x_k, v_k)\}$ est consistante localement.

Un CSP est *k*-consistant si et seulement si pour tout ensemble *Y* de *k-1* variables, toute configuration partielle sur *Y* est *k-consistante*.

A partir de la définition précédente, nous pouvons remarquer que la K-consistance est un cas général avec :

- *k* = 1 : Consistance de nœud
- *k* = 2 : Consistance d'arc

2.2.1.6. Algorithmes de filtrage et propagation de contraintes

Les algorithmes de filtrage reposent sur le type des contraintes exprimées. A chaque type de contrainte est associée une famille d'algorithme de filtrage ; par exemple pour les contraintes unaires, nous avons l'algorithme de filtrage par consistance de nœuds, pour les contraintes binaires nous avons les algorithmes de filtrage par arc-consistance, etc.

L'algorithme de filtrage associé aux contraintes unaires (voir Algorithme 1) est un algorithme très simple à comprendre et à mettre en œuvre. L'idée est de supprimer pour une contrainte donnée *c* qui porte sur une variable x_i l'ensemble des valeurs appartenant à D_{xi} ne satisfaisant pas la contrainte *c*.

```
Function ConsistanceDeNoeud(X, D, C)
1.    foreach contrainte unaire c_i in C do
2.        foreach valeur v dans D_i then
3.            if not c.satisfy(v) then
4.                delete v from D_i
5.            endif
6.        endforeach
7.    endforeach
```

Algorithme 1 : Consistance de nœud

Les algorithmes d'arc-consistance ont été très bien étudiés dans la littérature, c'est un mécanisme utilisé par tous les solveurs de CSP. Plusieurs raisons importantes ont poussé les chercheurs à étudier l'arc-consistance, notamment :

- Toutes les améliorations apportées à l'arc-consistance peuvent être appliquées pour atteindre les consistances locales.

- Atteindre la propriété de consistance locale en utilisant la propagation de contraintes et le filtrage simplifie la résolution d'un problème donné en réduisant l'espace de recherche.

Pour atteindre l'arc-consistance, il est nécessaire de vérifier les couples de valeurs possibles. La procédure *Revise-arc* (voir Algorithme 2), extraite de [12], permet d'atteindre pour un arc donné c_{ij}, la propriété de consistance.

```
Function Revise-arc(c_ij)
1.    delete = false
2.    foreach valeur a dans D_i do
3.        if ∀b ∈ D_j, (a,b) ne vérifie pas c_ij then
4.            Remove a from D_i
5.            delete = true;
6.        endif
7.    endforeach
8.    return delete
```

Algorithme 2 : Etablir la consistance pour un arc

Dans un graphe de contraintes binaires, pour faire en sorte que chaque arc soit consistant, une seule application de la fonction *Revise-arc* n'est pas suffisante. En effet la suppression d'une valeur d'un domaine pour une contrainte donnée peut entrainer la suppression d'autres valeurs dans d'autres domaines. D'où la nécessité de réviser à nouveau des arcs déjà révisés.

Mackworth a proposé le premier algorithme AC-1 [12] capable d'établir l'arc-consistance dans un graphe de contraintes. Il a proposé aussi son amélioration AC-3 [12][13]. AC-3 est l'algorithme le plus connu dans la littérature.

AC-1, l'algorithme le plus simple (voir Algorithme 3), tente de réviser systématiquement toutes les contraintes en dépit du fait qu'elles ne soient pas affectées par une révision antérieure. La complexité d'AC-1 qui est donnée par *Nombre d'appels à Revise-arc * coût de Revise-arc* est de $O(nmd^3)$.

```
Function AC-1(X, D, C)
1.    Q = All c_ij in C
2.    do
3.        Change = false
4.        foreach c_ij in Q do
5.            Change = Revise-arc(c_ij)
6.        endforeach
7.    while (Change == true)
```

Algorithme 3 : AC-1

Une cause évidente de l'inefficacité d'AC-1 est qu'il suffit qu'un seul appel de *Revise-arc* effectue une unique suppression pour que tous les arcs soient réexaminés lors de l'itération suivante, alors qu'une faible part seulement a pu être affectée par la suppression. L'amélioration apportée par AC-3 (voir Algorithme 4) consiste à ne pas réappliquer Revise-arc à *tous* les arcs, mais uniquement à ceux susceptibles d'être affectés par la suppression de la valeur d'un domaine. Les arcs devant être traités sont mis en attente dans la liste Q, et bien évidemment, un arc n'y est pas réinséré s'il y figure déjà.

Function AC-3(*X*, *D*, *C*)
1. Q = All c_{ij} in C
2. while ***Q* is not EMPTY** do
3. $c_{ij} = Q$.pop()
4. if Revise-arc(c_{ij}) == true then
5. $Q = Q \bigcup \{c_{mi}\}$ / $\{c_{mi}\}$ l'ensemble des contraintes de la variable i
6. endif ;
7. endwhile

Algorithme 4 : AC-3

Depuis la proposition d'AC-3, plusieurs algorithmes établissant l'arc-consistance ont été proposés, ces algorithmes visent à améliorer la complexité théorique et le temps de calcul effectif.

- AC-4 [11] : Une amélioration d'AC-3 qui permet de réduire la complexité théorique par l'utilisation de structure de données sophistiquées. AC-4 évite des tests redondants liés à AC-3. AC-4 possède 2 étapes : la première vise à sauvegarder dans une structure de données les informations concernant les valeurs et ses supporteurs dans les autres domaines ; la deuxième étape consiste à supprimer les valeurs ne possédant pas de supporteurs. Quand une valeur est supprimée, elle est ajoutée dans une file afin de propager l'effet de sa suppression. Cet algorithme n'est pas facile à mettre en œuvre, il peut même se révéler moins efficace que AC-3, de plus il possède une grande complexité spatiale.
- AC-6 [14] qui mixe des principes issus d'AC-3 et AC-4. Le but principal d'AC-6 est de réduire la complexité spatiale engendrée par AC-4. AC-6 sauvegarde seulement un support pour chaque valeur. Les valeurs sont supposées ordonnées.
- AC-Inference et AC-7 : AC-Inference a été introduit dans [15][16]. La connaissance des propriétés des contraintes peut permettre de réduire le coût de la vérification de la consistance. AC-7 [15][16] est un cas particulier d'AC-Inference est une amélioration d'AC-6. AC-7 exploite le fait que si *(x, a)* est supporté par *(y, b)* alors *(y, b)* est supporté par *(x, a)*. Ainsi, AC-7 est capable d'économiser quelques vérifications par rapport à AC-6 tout en gardant les mêmes complexités spatiales et temporelles.
- AC-2000 [17] et AC-2001/3.1 [18][19] : AC-2000 est une modification d'AC-3. AC-2001 (ou AC-3.1) est aussi basé sur AC-3 mais il utilise des concepts utilisé en AC-6 afin de réduire la complexité temporelle.
- Autres AC : beaucoup d'autres algorithmes d'arc-consistance ont été proposés dans la littérature comme AC-3.3 [20] qui est une amélioration d'AC-3, AC-8 [21], AC-3d [22] et AC-3.2 qui améliore AC-3, etc.

L'AC-3 est relativement simple à implémenter, il représente la complexité spatiale optimale (voir Tableau 1). AC-4 possède la complexité temporelle optimale O(md2) mais une grande complexité spatiale. AC-6 élimine quelques vérifications redondantes par rapport à AC-4 et réduit sa complexité temporelle. AC-2000 a la même complexité temporelle qu'AC-3 mais il fait moins de tests de vérification de consistance. AC-2001 est une amélioration d'AC-3 et possède la complexité temporelle optimale.

Algorithme	Complexité temporelle	Complexité spatiale
AC-3 [12][13]	$O(md^3)$	O(m+nd)
AC-4 [11]	$O(md^2)$	$O(md^2)$
AC-5 [100]	$O(md^2)$	O(md)
AC-6 [14]	$O(md^2)$	O(md)
AC-7 [15][16]	$O(md^2)$	O(md)
AC-2000 [17]	$O(md^3)$	O(md)
AC-2001 [18][19]	$O(md^2)$	O(md)

Tableau 1 : Complexité des algorithmes d'arc-consistance

La figure (voir Figure 5) montre quelques relations entre les différents algorithmes d'arc-consistance cités dans cette partie, par exemple AC-6 utilise des principes provenant d'AC-3 et AC-4. Pour plus de détails sur les algorithmes de filtrage, nous recommandons la lecture de [23][24][25].

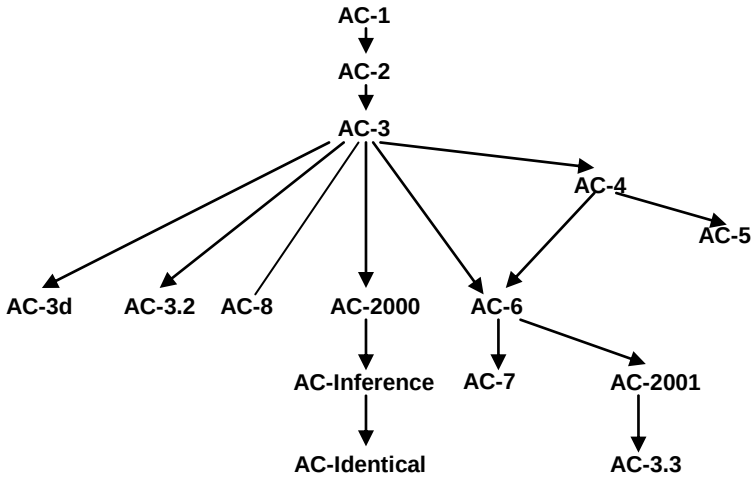


Figure 5 : Quelques relations entre des algorithmes d'arc-consistance

La notion d'arc-consistance s'étend aux CSP n-aires pour la consistance d'hyper-arc. Cette technique est généralement très lourde. Très peu d'algorithmes de résolution utilisent la consistance par hyper-arc.

Cependant des tests de consistance ont été développés spécifiquement pour certains types de contraintes n-aires. Ils sont plutôt rapides et aussi plus efficaces que les contraintes n-aires binarisées. La plus célèbre est la contrainte globale **all-different** (conjonction de contraintes de différences)**.** La consistance de cette contrainte [10] est assurée par la recherche d'un couplage maximal dans un graphe. Pour la contrainte cumulative, les techniques de consistance proposées sont incomplètes mais elles permettent d'inférer de nouvelles contraintes unaires et binaires, redondantes mais plus facilement traitées.

Dans la section suivante, nous proposons une synthèse très compacte sur les algorithmes de résolution des CSP.

2.2.2. Méthodes exactes de résolution

Nous nous intéressons dans cette partie à la résolution des CSP par les méthodes exactes qui sont capables de donner à la fin de la recherche une solution réalisable, voire toutes les

solutions réalisables, s'il y en a. Dans le cas ou le CSP traité n'a pas de solution, les méthodes exactes sont capables de conclure la non-satisfiabilité.

Nous présentons plus particulièrement des méthodes de recherche systématiques par opposition aux stratégies heuristiques (méthodes incomplètes comme la recherche locale). Nous commençons par le *backtracking* chronologique qui est l'algorithme de base de toutes ces méthodes ainsi que les heuristiques de prise de décision (ordonnancement des variables/valeurs) ajoutées à cet algorithme et qui améliore son efficacité. Ensuite, nous présentons d'autres méthodes qui améliorent le *backtracking* :

- En essayant d'éviter les conflits à l'avance : dans ce cas les techniques de filtrage sont les plus adaptées pour éviter l'exploration des branches infructueuses.
- En essayant de réparer les conflits détectés et de rendre le *backtracking* intelligent en identifiant et exploitant la cause de l'échec.

Nous présentons aussi des méthodes hybridant le *backtracking* intelligent avec les techniques de filtrage qui ont montré leurs efficacités sur de nombreux problèmes. Nous terminerons par les façons de prendre en compte un critère d'optimisation en plus de la recherche d'une solution qui satisfait l'ensemble de contraintes du problème.

2.2.2.1. Recherche systématique

« *Generate and test* » notée *GT* est une méthode basique pour résoudre un CSP. Elle consiste à générer toutes les configurations complètes possibles et à tester si elles vérifient l'ensemble de contraintes. Le nombre de possibilités testées est potentiellement immense ; c'est le cardinal du produit cartésien des domaines des variables du problème, ce qui pour des instances de très grande taille devient ingérable.

Le *backtracking* notée BT est l'algorithme de recherche systématique le plus répandue. Il est, presque, la base de toutes les méthodes de résolution exactes des CSP. BT améliore *GT* en ne développant que les affectations partielles consistantes. BT (voir Figure 6) a deux phases :

- Phase de marche avant : dans cette phase la prochaine variable à instancier est sélectionnée parmi l'ensemble de variables libres (non encore instanciées) et la configuration partielle courante est étendue en affectant à la variable sélectionnée une valeur consistante, si cette valeur existe.
- Phase de marche arrière : dans cette phase, quand une configuration partielle n'existe pas pour la variable courante (aucune valeur consistante dans le domaine de la variable courante), BT retourne à la dernière variable instanciée en essayant une autre valeur consistante pour cette variable. Le CSP est infaisable si BT est sur la 1ère variable de l'arbre de recherche, et qu'il tente une marche arrière.

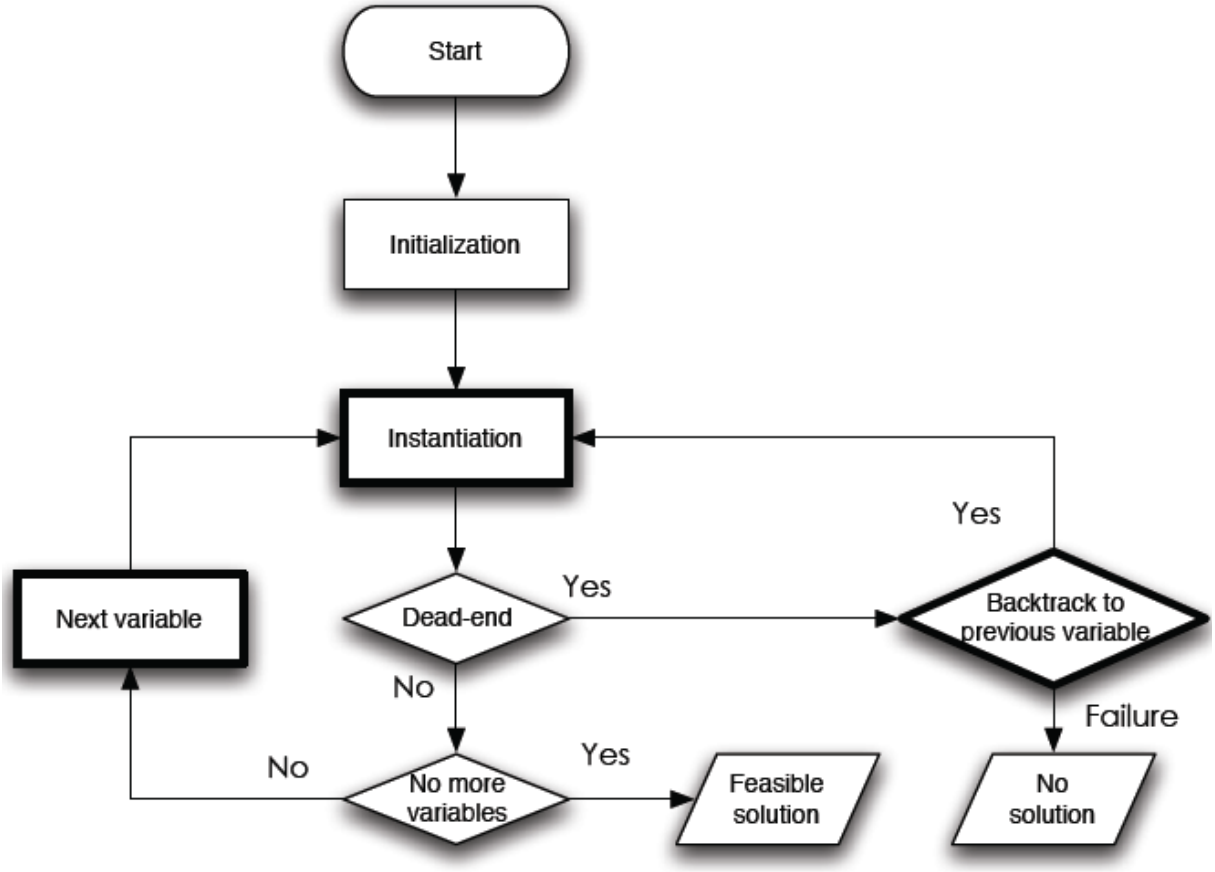


Figure 6 : Schéma général du *backtracking*

BT a l'avantage d'être générique, il est capable de résoudre tous les CSP en tenant en compte de tous les types de contraintes. Les inconvénients majeurs du BT sont : sa complexité temporelle exponentielle, la découverte redondante d'inconsistances locales et la détection tardive des conflits. Afin d'améliorer BT, plusieurs propositions ont été appliquées :

- Définition d'heuristiques sur l'ordre des variables à instancier et les valeurs dans les domaines des variables.
- Réduction de l'espace de recherche avant la recherche d'une solution en appliquant des algorithmes de filtrage et de propagation de contraintes en prétraitement.
- Modification de l'algorithme lui-même pour faire des retours intelligents.
- Modification de l'algorithme lui-même en tentant de détecter, à moindre coût et le plus rapidement possible, l'inconsistance de la configuration partielle courante.

Dans la sous-section suivante, nous allons expliquer comment améliorer l'efficacité du *backtracking* en changeant l'ordre des variables/valeurs à instancier.

2.2.2.2. Ordonnancement des variables/valeurs

Le *backtracking* classique n'utilise pas un ordonnancement spécial sur les variables ou les valeurs. Cependant, de nombreuses recherches [26] ont montré que les heuristiques d'ordonnancement des variables/valeurs peuvent jouer un rôle très important dans l'amélioration de l'efficacité du *backtracking* (temps de résolution) en influant sur la taille de l'espace de recherche. Ces heuristiques sont statiques ou dynamiques. Les heuristiques

statiques fixent au début de la recherche un ordre sur les variables/valeurs, cet ordre ne change pas pendant la recherche. Tandis que, l'ordre pour les heuristiques dynamiques peut changer pendant la recherche en fonction de la situation actuelle de la recherche et plus précisément en fonction de la configuration partielle courante.

Il est préférable de définir l'ordre de l'instanciation des variables/valeurs selon la nature de l'application à traiter. Cependant, il existe des stratégies génériques qui peuvent être appliquées sur tous les problèmes, parmi ces heuristiques nous pouvons citer :

- La cardinalité maximale : la prochaine variable à instancier doit avoir le plus grand nombre de contraintes avec les variables déjà instanciées.
- Le degré maximum : la prochaine variable à instancier doit avoir le plus grand nombre de contraintes (nombre de voisin).
- La largeur minimale : la prochaine variable à instancier est la variable de degré minimal en tenant seulement compte du graphe de contraintes induit par les variables libres.
- *Dom* ou *MRV : Dom* ou *MRV* pour *Minimum Remaining Value* consiste à choisir la variable possédant le plus petit domaine.
- *dom/deg :* La prochaine variable à instancier est la variable qui minimise le rapport *taille du domaine/nombre de contrainte.*

2.2.2.3. Evitements des conflits (look-ahead)

Une méthode d'amélioration du *backtracking* est de tenter de prévenir à l'avance les inconsistances futures (principe du look-ahead), en combinant *backtracking* et propagation de contraintes. Un nœud de l'arbre s'identifie à un sous problème CSP, où les domaines des variables instanciées sont réduits à des singletons. Le *backtracking* vérifie à chaque nœud les contraintes des variables instanciées pour détecter la violation de contraintes.

Haralick et Elliot [27] ont proposé l'algorithme FC (*Forward Checking*) notée. FC est un *backtracking* avec une maintenance partielle de l'arc consistance pendant la recherche ; on vérifie uniquement les contraintes directes avec les variables instanciées. FC calcule l'impact d'une instanciation avant même de prendre la décision. Avant d'affecter une valeur v à une variable x_i, FC vérifie l'impact de cette instanciation sur les variables voisines et libres de x_i. Si un domaine d'une variable devient vide (*deadend*), on changera de valeur pour x_i.

Le MFC (*Minimal Forward Checking*) [28] et le *backmarking* [29] sont des améliorations légères du FC, ces algorithmes essayent de minimiser le nombre de vérifications inutiles faites par le FC.

Sabin et Freuder ont proposé le MAC [82] (*Maintaining Arc Consistency during search*) qui est considéré comme l'un de meilleurs algorithmes pour la résolution des CSP [26]. Contrairement au FC (voir Figure 7) qui établit l'arc consistance avec les variables voisines de celles instanciées, MAC établit l'arc consistance en lançant la propagation dans tout le CSP à chaque nœud de l'arbre. Après la propagation deux cas de figures peuvent se reproduire :

1. Aucun domaine vide n'est généré et la configuration partielle courante pourra être étendue en instanciant une variable libre x_i à une valeur non encore éliminée v dans D_{xi}.

2. Un *deadend* est atteint (un domaine d'une variable est devenu vide). Dans ce cas, un retour est nécessaire. MAC retourne, comme le *backtracking* chronologique, à la dernière variable instanciée. Toutes les valeurs éliminées suite à la dernière affectation sont restaurées à leurs domaines respectifs.

L'algorithme MAC s'arrête lorsqu'il atteint une solution ou lorsqu'il démontre que le problème est infaisable.

MAC peut utiliser n'importe quel algorithme de filtrage AC-3, AC-4, AC-6, AC-2001, etc.

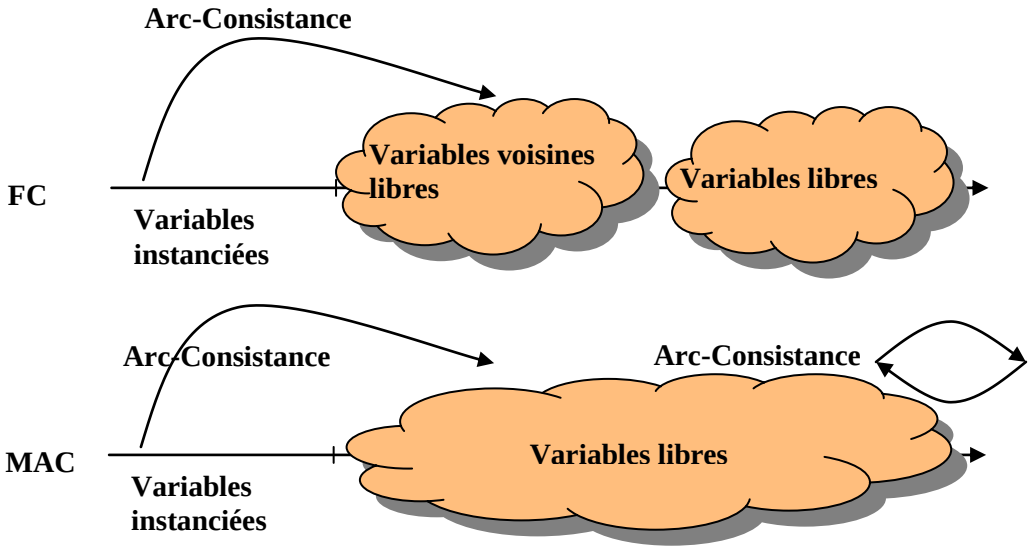


Figure 7 : Différences entre FC et MAC

Il est intéressant de noter l'existence de certains algorithmes qui effectuent plus de propagation que le FC et moins de propagation que le MAC, par exemple le *partial looking ahead* [27] ou le *directional arc consistency lookahead* [31].

Le schéma (voir Figure 8) extrait de [33] montre la différence entre les différents algorithmes de type *look-ahead*. Le *forward checking* agit sur les variables libres voisines des variables instanciées tandis que le MAC (full *look-ahead*) agit sur toutes les variables libres. Le *partial look-ahead* , un compromis entre les deux, agit sur un sous ensemble de variables libres.

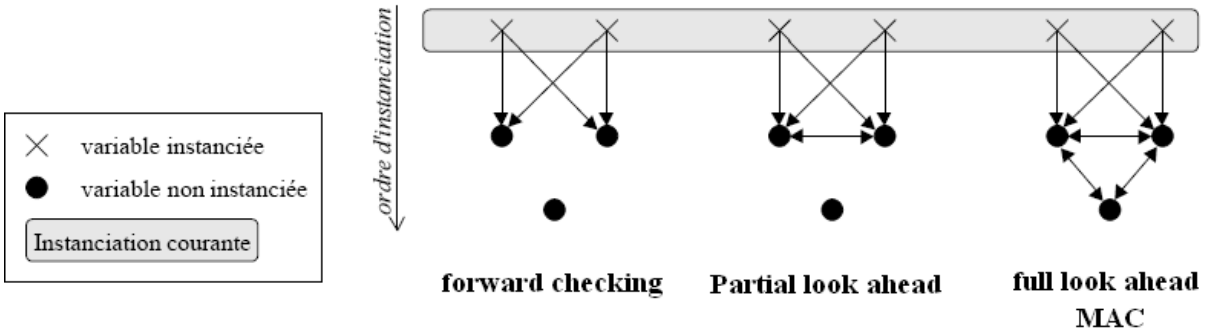


Figure 8 : Comparaison de différents algorithmes de *look-ahead*

2.2.2.4. Réparations des conflits (look-back)

Les algorithmes de retours intelligents sont des méthodes rétrospectives « *look-back* ». Cette famille de méthodes regarde de plus près les causes de l'inconsistance d'une configuration partielle courante *S'* et essaye d'identifier l'origine de cette inconsistance et de l'analyser afin de réparer *S'* en fonction du conflit identifié. Le but principal est de remédier à deux principaux défauts du *backtracking* classique : le *trashing*, qui est la reproduction d'un même échec dû à une affectation antérieure issue d'un *nogood* minimal, et la découverte redondante des mêmes configurations partielles inconsistantes.

L'exemple suivant, extrait de [95], illustre ce comportement, soit *P = (X, D, C)* avec :

- X = {x1, x2, x3, x4}
- $D_{x_1} = D_{x_2} = D_{x_3} = D_{x_3} = \{0,1,2\}$
- *C* = {*c*, *c'*} avec $c = x_3 + 2 < 2x_2$ et $c' = x_4 < x_1$

La Figure 9 représente une partie de l'arbre de recherche où les nœuds sont les configurations partielles. Par exemple, (0,0,*,*) est la configuration partielle $x_1 = x_2 = 0$ avec x_3 et x_4 libres. Dans un *backtracking* simple, avec un ordre numérique sur les variables et les valeurs, le phénomène de *trashing* apparait au moment d'instancier x_4 car, quelles que soient les valeurs de x_2 et x_3, aucune valeur de x_4 n'est compatible avec $x_1 = 0$.

BackJumping :

Afin d'éviter ce cas de figure, le *backjumping* effectue un retour sur une instanciation causant le conflit ; dans notre cas, on passe directement du nœud (0,2,0,*) au nœud (1,*,*,*) sans visiter les nœud (0,2,1,*) et (0,2,2,*). L'idée est de détecter la cause du conflit sur le nœud (0, 2, 0, *) qui est $x_1 = 0$ et par la suite faire un *backjumping* pour revenir sur ce choix en le changeant et donc passer directement au nœud (1.*.*.*)

Le *backjumping* notée BJ est une technique qui vise à améliorer le *backtracking* en faisant des retours sur des instanciations en conflit avec la dernière instanciation faite. Les algorithmes de *backjumping* (Backjumping de Gashnig [37][39], Graph-based *backjumping* [40], CBJ *Conflict-directed backjumping* [41], etc.) se distinguent entre eux par la manière d'identifier l'instanciation la plus haute dans l'arbre de recherche à laquelle il faut remonter sans oublier de solutions. Autrement dit, il s'agit d'identifier un sous ensemble appelé *conflict-set*, le plus petit possible, de la configuration partielle courante qui interdit la prise de l'instanciation présente (dernière instanciation faite). Quand un échec intervient au moment de l'instanciation d'une nouvelle variable x_i, le *conflict-set* peut être l'ensemble des variables voisines de x_i déjà instanciées et qui interviennent dans au moins une contrainte violée avec x_i. La convergence de l'algorithme est assurée en effectuant le retour sur l'instanciation la plus récente dans le *conflict-set.*

Le *backjumping* effectue un travail de reconstruction depuis le point de retour : toutes les instanciations entre le point de retour et l'instanciation présente sont annulées. Le *dynamic backtracking* notée DBT [42] améliore le *backjumping* en conservant ces instanciations. Il extrait une explication *I'* de chaque extension d'une configuration partielle *I* (*I*, x_i=*v*) inconsistante. (*I'*, x_i=*v*) est mémorisé temporairement comme un *nogood* servant à empêcher d'explorer toute la branche le contenant. Dans le cas ou *I* mène à un *deadend*, le DBT effectue un retour sur la dernière instanciation x_j=*w* du *nogood* stocké tout en conservant les

instanciations faites après x_j. Il mémorise le *nogood I'* comme explication du *deadend* rencontré et désactive tous les *nogood*s contenant l'affectation $x_j=w$.

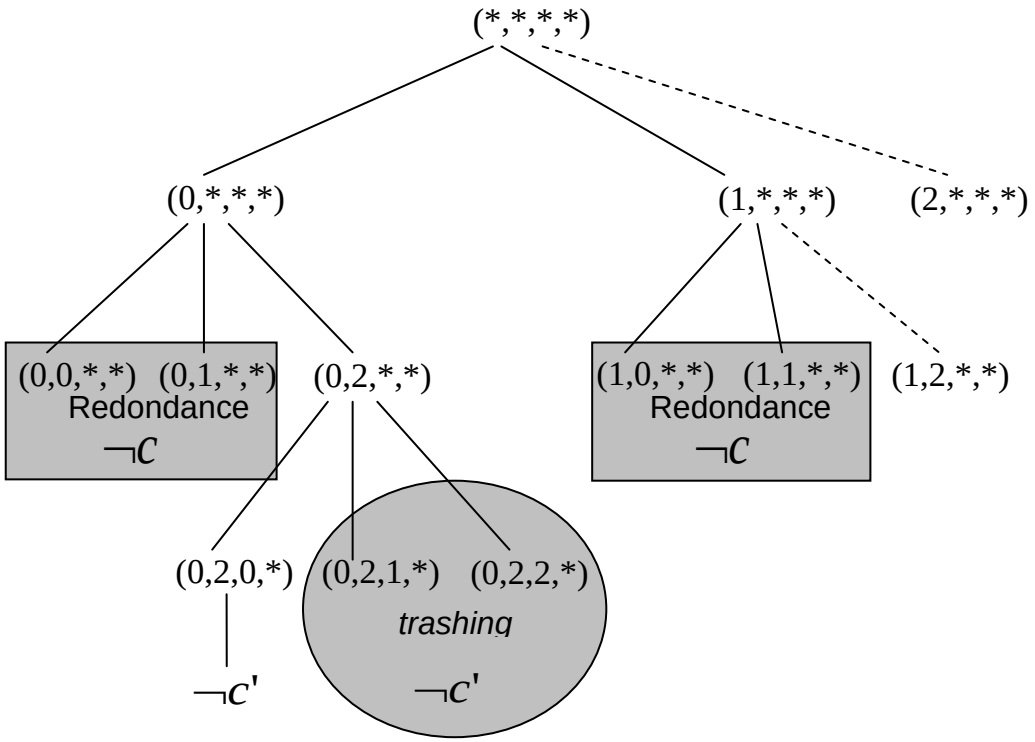


Figure 9 : Redondance et *trashing*

Les algorithmes de *backjumping* améliorent considérablement le *backtracking*, cependant ils n'empêchent pas la redondance de la recherche. Dans l'exemple de la Figure 9, nous remarquons qu'une partie du travail effectué sur les variables x_2 et x_3 pour l'instanciation $x_1 = 0$ est répétée pour l'instanciation $x_1 = 1$. La solution consiste à apprendre de la recherche passée, c'est-à-dire à enregistrer les informations déduites des branchements effectuées et qui ont conduits à des échecs. Nous présenterons dans la sous-section suivante, les approches basées sur le principe de l'apprentissage ou « *learning algorithms* ».

Apprentissage (learning) :

Une des techniques les plus efficaces pour améliorer les performances du *backtracking* est l'apprentissage qui se base sur les *nogood.* Le *Nogood-Recording* est une forme de cette technique. Il s'agit d'ajouter des nouvelles contraintes au CSP original. Les contraintes ajoutées n'affectent pas l'espace des solutions réalisables. Ajouter une contrainte « correcte » peut signifier que plusieurs *deadend* sont supprimés de l'arbre de recherche et d'autres *deadend* seront découverts avec beaucoup moins d'effort de recherche.

Il y a plusieurs méthodes d'ajout de contraintes, avant ou pendant la recherche. Une des techniques répandues est d'ajouter automatiquement des contraintes pendant la recherche et après la découverte d'une inconsistance locale ou un *deadend*. Le concept des *nogood* est la base de cette technique. Si on utilise le BT pour chercher une solution, chaque *deadend* correspond à un *nogood.* Ces *nogoods* sont la cause d'un effort de recherche inutile. Une fois qu'un *nogood* pour un *deadend* est identifié, il pourra être transformé en une contrainte à ajouter au CSP original. Bien sûr, c'est trop tard pour ce *deadend* (le BT a déjà réfuté ce nœud,

peut être avec un coût très élevé), mais l'espoir est que la contrainte ajoutée va réduire l'espace de recherche dans le futur. Cette technique a été introduite par Stallman et Sussmann [34], et elle est généralement connue sous le nom de *nogood* ou *Constraint Recording.*

Exemple : Imaginons que pour un CSP *P*, nous avons la configuration partielle $S' = \{x_1 = 2, x_2 = 5, x_3 = 3\}$ qui conduit à un *deadend*. Donc *S'* ne pourra jamais appartenir à une solution réalisable. Alors la contrainte $\neg(x_1 = 2 \wedge x_2 = 5 \wedge x_3 = 3)$ pourra être enregistrée ; elle signifie qu'une configuration partielle ne pourra jamais contenir les 3 affectations en même temps. La contrainte ajoutée pourra être testée comme n'importe quelle autre contrainte pendant la recherche.

Les discussions menées dans le monde de la recherche sur la découverte et sur l'identification des *nogoods* se sont concentrées principalement sur des méthodes qui s'intègrent dans le processus de recherche d'une solution et notamment dans le mécanisme de la propagation de contraintes. La manière la plus répandue et la plus simple afin de détecter un *nogood* est d'associer à chaque valeur dans le domaine d'une variable une explication d'élimination qui contient la cause de l'élimination de cette valeur. Lorsqu'un domaine d'une variable devient vide, un *nogood* sera l'union de toutes les explications d'élimination de toutes les valeurs de la variable concernée [96] [98].

La sauvegarde des *nogoods* est actuellement très peu utilisée dans la résolution des CSP [35]. La raison principale est la présence de contraintes n-aires dans la plupart des modèles CSP et que certaines formes de consistances locales sont coûteuses pour être établies pour ce type de contraintes.

Un problème important qui se pose pour le *Nogood-Recording* est le coût de la mise à jour et l'interrogation de la base des *nogoods* stockés. Stallman et Sussman [34] propose de sauvegarder un *nogood* à chaque *deadend* rencontré. Cependant, la base des *nogoods* devient très grande et son interrogation devient très coûteuse ; par la suite la réduction induite de l'espace de recherche devient globalement non bénéfique. Une des méthodes pour réduire ce coût est de fixer la taille de la base des *nogoods* stockés en stockant uniquement les *nogoods* utiles. Dechter [40] propose d'enregistrer seulement les *nogoods* de taille *i* (les *nogoods* qui contiennent *i* affectations), où *i* est un paramètre à choisir selon le type du problème à résoudre. Ginsberg [42] propose d'enregistrer tous les *nogoods* identifiés pendant la recherche et de supprimer les *nogoods* qui ne sont plus pertinents. Bayardo et Miranker [101] généralisent la proposition de Ginsberg, en supprimant seulement les *nogoods* qui ne sont plus pertinents et qui ont une taille plus grande qu'une certaine valeur *i* à déterminer. Toutes les expérimentations menées par les chercheurs sur l'enregistrement des *nogoods* pendant la recherche on conclut que l'enregistrement est très coûteux ; cependant il dépend de la taille de la base des *nogoods* stockés et de la taille fixée des *nogoods* à prendre en compte. Des expérimentations faites sur des CSP binaires aléatoires ont conclu que le CBJ avec *i = 2* est le meilleur compromis [35]. Bayardo et Schrag [103], avec des expérimentations sur plusieurs CSP réels et aléatoires, ont conclu qu'un algorithme de type CBJ avec *i=4* est le meilleur compromis. Lynce et Marques-Silva [102], avec des expérimentations sur des problèmes réels, ont conclu que le même algorithme est meilleur avec *i=20*. On voit que le problème n'est pas simple.

Au-delà de la limitation de la taille de la base des *nogoods* stockés, des techniques ont été proposées afin d'améliorer le coût de la mise à jour et de l'interrogation de la base des

nogoods stockés. L'une de ces techniques est la « *watch literals* » [36]. Les « *watch literals* » sont des structures de données qui réduisent énormément le nombre de *nogoods* à examiner pour répondre aux requêtes demandées ; elles réduisent aussi le coût des tests des grands *nogoods*. Cette technique a montré son efficacité sur de nombreux problèmes [36] [37].

2.2.2.5. Look-back + Look-ahead

Les algorithmes *look-back* et *look-ahead* visent à diminuer le nombre de nœuds explorés et donc réduire l'espace de recherche. Une idée très intéressante combine ces deux mécanismes. Ainsi, des nouvelles méthodes combinent la propagation de contraintes avec des techniques de retours intelligents :

- MAC et CBJ [43]
- FC et DBT [44]
- MAC et DBT [45]

MAC-DBT a montré [45] de très bonnes performances sur différents CSP (grande et petite taille). Bessière a montré [26] que MAC est souvent plus rapide tout seul qu'avec sa combinaison avec une technique de *backjumping*.

Toutes les méthodes et techniques présentées dans ce chapitre ne sont pas forcément complémentaires et leurs hybridations ne garantissent pas un apport d'efficacité et de rapidité sur la résolution d'un problème donné. Le choix de la méthode à adopter dépend fortement de la nature du problème qu'on traite, de sa structure et de ses propriétés.

Afin de concevoir une méthode de résolution rapide, il est primordial non seulement de trouver le meilleur compromis entre le nombre de nœuds du graphe de recherche et le temps de résolution passé à chaque nœud, mais aussi d'étudier la structure du problème traité et les combinaisons possibles entre heuristiques d'ordonnancement, techniques de filtrages et méthodes d'apprentissage.

2.2.2.6. Optimisation

Plusieurs applications importantes et dans des domaines différents tels que la planification et l'affectation de ressources ont pour but de trouver une solution qui satisfait un ensemble de contraintes donné mais aussi d'optimiser une fonction définissant un objectif. Nous supposons que le but est de trouver une solution pour $P=(X, D, C)$ qui minimise f et f est une fonction sur toutes les variables de P. Nous supposons qu'une variable t est ajoutée à la liste des variables X de P avec $t = f(X)$.

Toutes les méthodes exposées dans ce chapitre pour la recherche d'une solution peuvent être appliquées à la résolution des CSOP (*Constraint Satisfaction Optimisation Problem*). L'approche la plus intuitive est de trouver une solution optimale par la résolution d'une séquence de CSP. Plusieurs variations ont été proposées et évaluées dans la littérature. Van Hentenryck [48] propose une version de la méthode *Branch-and-bound*. Initialement, un algorithme de *backtracking* (BT basique et la plupart de ses améliorations) est utilisé pour trouver une solution qui satisfait toutes les contraintes. Chaque fois qu'une solution est trouvée, une contrainte s'ajoute au CSP avec la forme $t > f(S)$ qui exclut toutes les solutions qui ne sont pas meilleures que S. Ce processus est répété jusqu'à ce que le CSP résultant soit infaisable ; dans ce cas la dernière solution trouvée est optimale. Baptiste, Le Pape, et Nuijten [49] suggère d'itérer sur les valeurs possibles de t de la plus petite vers la plus grande jusqu'à

l'obtention d'une solution ou de la plus grande vers la plus petite jusqu'à ce que l'on ne trouve plus de solution.

2.2.3. Les métaheuristiques

Les méthodes exactes ne sont pas toujours applicables à cause du temps de calcul. En effet, pour des problèmes de très grande taille le temps nécessaire à la résolution peut être très important et pour certains problèmes de références ou de l'industrie la recherche d'une solution doit être réalisée avec une contrainte de temps de calcul. Une méthode exacte [6] [7], qui consiste à faire une énumération exhaustive en décomposant le problème, a été appliqué à différents problèmes d'optimisation comme le problème de coloriage de graphe et testée sur les instances DIMACS [7]. Plusieurs instances ont été résolues en quelques heures d'une façon optimale. Une de ces instances a été résolue par la méthode exacte en deux heures alors que certaines méthodes approchées trouvent une solution de même qualité en moins d'une minute.

Ainsi, lorsque l'on est limité par un temps de calcul sur des problèmes de grande taille, il faut mieux chercher une solution de bonne qualité avec des méthodes approchées que d'utiliser des méthodes exactes sans garantie de trouver une solution dans le temps imposé. Cette classe de méthodes permet de trouver une solution de bonne qualité en un temps raisonnable ; par contre elle ne garantit pas l'obtention d'une solution optimale.

Parmi les méthodes approchées, on peut distinguer deux catégories : les heuristiques et les métaheuristiques [8]. Les heuristiques sont des méthodes qui traitent un problème particulier alors que les métaheuristiques sont des méthodes générales qui peuvent être appliquées à différents problèmes d'optimisation. Widmer et al [9] ont regroupé les différentes définitions connues sur les métaheuristiques. Ils ont aussi proposé un ensemble de propriétés intéressantes qui caractérisent les métaheuristiques :

- Les métaheuristiques sont des stratégies qui permettent de guider la recherche d'une solution optimale.
- Le but visé par les métaheuristiques est d'explorer l'espace de recherche efficacement afin de déterminer des solutions (presque) optimales.
- Les techniques qui constituent des algorithmes de type métaheuristique vont de la simple procédure de recherche locale à des processus d'apprentissage complexes.
- Les métaheuristiques sont en général non-déterministes et ne donnent aucune garantie d'optimalité.
- Les métaheuristiques peuvent contenir des mécanismes qui permettent d'éviter d'être bloqué dans des régions de l'espace de recherche.
- Les concepts de base des métaheuristiques peuvent être décrits de manière abstraite, sans faire appel à un problème spécifique.
- Les métaheuristiques peuvent faire appel à des heuristiques qui tiennent compte de la spécificité du problème traité mais ces heuristiques sont contrôlées par une stratégie de niveau supérieur.
- Les métaheuristiques peuvent faire usage de l'expérience accumulée durant la recherche de l'optimum pour mieux guider la suite du processus de recherche.

Ces propriétés définissent le comportement de toutes les métaheuristiques pendant la recherche d'une solution, en allant de la recherche locale jusqu'aux algorithmes génétiques qui sont parmi les plus complexes. Les métaheuristiques visent principalement à minimiser ou maximiser une fonction de coût donnée appelée fonction *fitness* dans le sens où elle vise à évaluer la qualité d'une solution. Ce qui pour les CSP correspond par exemple à minimiser la somme des contraintes violées pour des configurations complètes.

Le nombre de méthodes et leurs variantes étant extrêmement important, dans les sections suivantes nous ne présentons que quelques méthodes en rapport avec nos travaux. Pour plus de détails sur les méthodes incomplètes de résolution, nous recommandons la lecture de [35] [46] [47].

2.2.3.1. Méthodes de recherche locale

Contrairement aux méthodes constructives, la plupart des méthodes de recherche locale manipulent des configurations complètes qu'elles modifient afin de rechercher une meilleure solution. La recherche locale se base donc sur l'idée d'amélioration d'une configuration complète donnée S en faisant des changements sur les affectations. Les configurations complètes obtenues en modifiant S s'appellent les voisines de S et constituent l'ensemble $N(S)$. Ainsi, la recherche locale commence par une configuration complète initiale et se déplace d'une configuration vers une voisine pour visiter l'espace de recherche. La notion de voisinage est alors primordiale.

L'algorithme (voir Algorithme 5) résume le fonctionnement global des méthodes de recherche locale.

Function RechercheLocale
1. Construire une configuration S de départ
2. while **un critère d'arrêt n'est pas vérifié** do
3. S =Select($N(S)$)
4. endwhile
5. return Meilleure configuration rencontrée

Algorithme 5 : Fonctionnement global de la recherche locale

La première étape d'une méthode de recherche locale est la construction d'une configuration de départ. Cette étape est simple, on utilise généralement une configuration aléatoire ou des algorithmes gloutons pour construire une bonne configuration de départ. Le reste est un processus itératif. Il consiste à remplacer la configuration courante S par une autre, voisine de S. Il s'arrête lorsque le critère d'arrêt prédéfini est vérifié : un nombre d'itérations, temps d'exécution, etc. A la fin du processus, la meilleure configuration trouvée durant la recherche est retournée.

2.2.3.2. Notions de voisinage

Duhamel [64] propose un modèle conceptuel des méthodes par amélioration itérative. Les points communs de ces méthodes peuvent être résumés en trois règles définissant :

1. Le voisinage : Elle permet de construire l'ensemble des configurations voisines de la configuration courante, cela correspond à ce que nous appellerons structure de voisinage.
2. La sélection de voisins parmi les configurations du voisinage : Elle organise la recherche dans l'ensemble des configurations voisines, nous la noterons opérateur de sélection.
3. La modification effectuée : Elle correspond aux mouvements ou aux règles de passage vers la nouvelle configuration à partir du sous-ensemble de voisines sélectionnées.

La définition de la structure de voisinage est une étape clé dans la mise au point d'un algorithme de recherche locale puisqu'elle permet de définir l'ensemble de configurations voisines qu'on peut atteindre à partir d'une solution donnée en faisant une série de transformations.

Définition 18 : Voisinage

S étant une configuration complète, le voisinage de *S* est un sous-ensemble de configurations complètes directement atteignable à partir d'une transformation donnée de *S*. Il est noté $V(S)$ et une configuration $S' \in V(S)$ est dite voisine de *S*.

La structure du voisinage dépend de la transformation autorisée. Ainsi, à partir d'une configuration donnée, on peut établir plusieurs structures de voisinage selon la transformation que l'on autorise. Chaque structure de voisinage fournit un voisinage, ou un ensemble de configurations, précis via la transformation définie.

La Figure 10 prise de [78] présente deux voisinages distincts. Une configuration est composée ici de 3 variables binaires. Selon le voisinage V_1, deux solutions sont voisines si et seulement si une seule composante diffère entre elles. Le voisinage V_2 définit deux configurations voisines si et seulement si une permutation entre deux composantes permet de passer de l'une à l'autre. Depuis une configuration *S* quelconque l'ensemble $V_1(S)$ est différent de $V_2(S)$. Cependant les voisinages se recoupent souvent très largement ; par exemple certaines solutions appartenant à l'ensemble $V_2(S)$ sont accessibles à partir de l'ensemble $V_1(S_1)$. La structure du voisinage permet de se déplacer d'une solution à une autre dans l'espace de recherche en établissant des chemins entre les configurations.

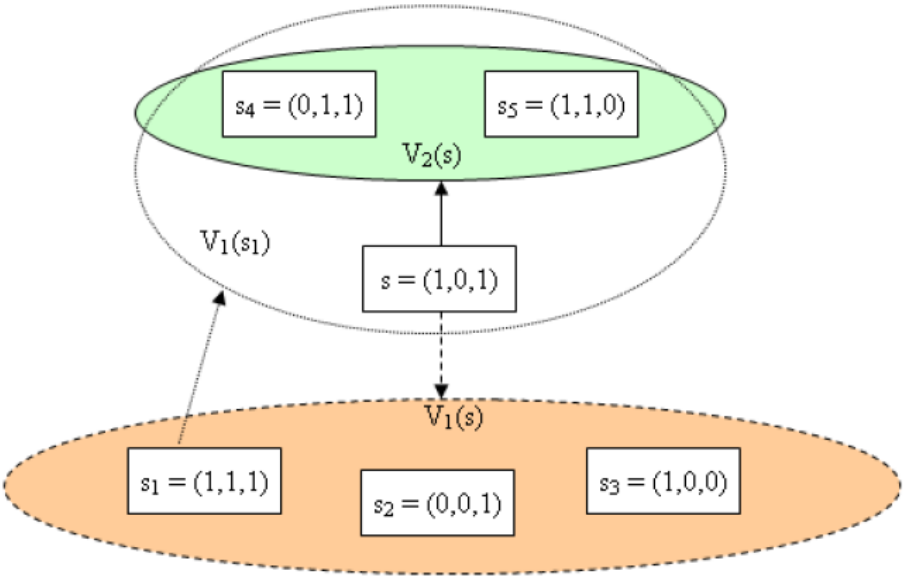


Figure 10 : Exemple de structures de voisinage

Selon Duhamel [64], une structure de voisinage se caractérise par 3 propriétés principales :

1. La complexité spatiale : Relation entre la taille du voisinage et la taille de l'espace de recherche du problème.
2. La complexité temporelle : Temps nécessaire pour évaluer toutes les configurations voisines.
3. La portée : Permet de mesurer le degré de transformation entre la configuration courante et les configurations voisines. Trois degrés de portée sont proposés :
    a. Portée locale : peu de modifications dans la configuration courante.
    b. Portée régionale : une plus grande quantité de modifications est introduite dans les configurations voisines par rapport à la configuration courante.
    c. Portée globale : Transforme beaucoup les caractéristiques de la configuration courante.

La portée permet de différencier deux structures de voisinage en fonction de l'éloignement de l'ensemble de configurations accessibles par rapport à la configuration initiale.

La portée d'une structure de voisinage $V(S)$ est définie par la distance maximale des configurations appartenant à $V(S)$ par rapport à la configuration S .

Il existe plusieurs manières de calculer cette distance [65]. La distance de Hamming est l'exemple le plus courant, elle correspond au nombre de composantes différentes entre deux configurations voisines. Cette distance est parfaitement adaptée aux configurations composées de variables binaires comme les configurations de la Figure 10.

Définition 19 : Distance de Hamming

La distance de Hamming (baptisée du nom de Richard Hamming), désigne le nombre d'éléments différents entre deux ensembles de même taille.

Mathématiquement, la distance de Hamming d_H entre deux configurations S et S' est calculée de la façon suivante :

$$d_H(S,S') = \sum_{i=1}^{N} \delta_{S(x_i),S'(x_i)}$$

Où $\delta_{i,j}$ est le symbole de Kronecker :

$$\delta_{i,j} = \begin{cases} 1 & \text{si i = j} \\ 0 & \text{sinon} \end{cases}$$

et $S(x_i)$ la valeur de la composante x_i (variable) dans la configuration S, et N le nombre de variables du problème.

On peut maintenant définir d'une manière précise les notions d'optimum local et global.

Définition 20 : Optimum local

Une configuration S est un optimum local si et seulement si il n'existe pas une autre configuration $S' \in V(S)$ de fonction de coût strictement meilleure.

$$\forall S' \in V(S) \begin{cases} f(S) \leq f(S') & \text{dans le cas d'un problème de minimisation} \\ f(S) \geq f(S') & \text{dans le cas d'un problème de maximisation} \end{cases}$$

Avec $V(S)$ l'ensemble des configurations voisines de S.

La définition d'un optimum local est liée à la notion de voisinage. Un optimum local pour une structure de voisinage donnée n'est pas forcément un optimum local pour une autre structure de voisinage.

Définition 21 : Méthode de descente

Une méthode de descente est une méthode dont le principe consiste à choisir à chaque étape une configuration voisine meilleure que la configuration courante. Quand une méthode de descente n'est plus applicable, un optimum local est atteint.

Une variante est la méthode de plus forte descente qui choisit à chaque étape la meilleure configuration voisine parmi celles qui améliorent la solution courante.

Définition 22 : Optimum global

Une configuration S qui est un optimum local pour toutes les structures de voisinage du problème est appelée optimum global.

2.2.3.3. Quelques méthodes

Les méthodes de recherche approchées sont aussi parfois appelées méthodes de trajectoire. Il existe un grand nombre de méthodes [60] [61] [62] [63], nous présentons ci-après un choix de trois méthodes de recherche locale en lien avec nos travaux.

Le recuit simulé [66] [69] :

Proposée par trois chercheurs de la société IBM en 1983 [66], le recuit simulé est une méthode de recherche locale inspirée d'un processus utilisée en métallurgie.

On sait que la méthode de descente n'accepte qu'une configuration qui améliore la configuration courante et donc s'arrête dans le premier minimum local rencontré. Le recuit simulé accepte toujours une solution qui améliore la meilleure solution trouvée mais utilise aussi un critère de sélection relaxé qui peut accepter une configuration de qualité moindre en fonction de la dégradation par rapport à la meilleure solution trouvée. Le niveau de dégradation dépend d'un paramètre *température* qui décroit au cours du temps.

Il s'agit d'une méthode adaptative dans la mesure où la courbe de descente de la température, qui conduit à la réduction progressive du voisinage, est estimée au début de l'algorithme après quelques itérations [67]. Le paramètre de température est utilisé pour restreindre progressivement, et indépendamment de l'historique de recherche, l'ensemble des solutions voisines qui dégradent la fonction de coût. L'écart à la meilleure solution trouvée est important au début pour favoriser l'exploration et progressivement de plus en plus faible pour intensifier la recherche sur l'optimum local.

Cet algorithme est toujours très utilisé notamment dans le domaine des réseaux [68].

La recherche à voisinage variable [70] [71] [73] :

Proposée par Hansen et Mladenovic en 1997 [73], la méthode de recherche à voisinage variable de base VNS (*Variable Neighborhood Search*) est une métaheuristique récente qui exploite le changement de structure de voisinage, que ce soit en descente vers un optimum local ou pour s'échapper de ces optimums locaux.

L'idée de base est de faire varier les structures de voisinage en se basant sur le fait qu'une configuration *S* reconnue comme un optimum local pour une structure de voisinage donnée peut ne pas être un optimum local pour une autre structure de voisinage.

Cette méthode fonctionne en plusieurs étapes. Tout d'abord elle nécessite la définition de différentes structures de voisinage, notées $V_1,...,V_{k_{\max}}$, forcément différentes dans leurs actions. Une relation d'ordre sur la portée entre les structures de voisinage est à définir en fonction du problème. A partir d'une configuration complète de départ *S*, une voisine de *S* est sélectionnée en utilisant la structure de voisinage V_1 a priori de plus faible portée pour favoriser une recherche locale. Si la configuration obtenue est meilleure que la configuration de départ alors la recherche reprend à partir de cette configuration. Sinon, une autre configuration complète peut être sélectionnée en utilisant un autre voisinage de portée supérieure, soit ici le voisinage V_2 ; la même règle est appliquée sur ce deuxième choix, une nouvelle recherche locale est effectuée sur cette configuration ou on passe à un autre voisinage. A noter que les règles de changement de structure peuvent être plus complexes.

Cette méthode permet de sortir d'un optimum local si celui-ci est un optimum local pour un voisinage V_i avec $i < k_{\max}$.

Avanthay et al. [74] ont utilisé cette méthode pour résoudre le problème de *k*-coloration de graphe. Pour toutes les instances DIMACS testées, la méthode VNS obtient de meilleurs résultats que la méthode *Tabucol* utilisée seule. La méthode *Tabucol* est une recherche Tabou proposée par Hertz et al. [56] [75] et appliquée principalement sur les problèmes de coloration.

Il peut être donc intéressant d'utiliser des algorithmes combinant plusieurs structures de voisinage pour la recherche dans les problèmes disposant de nombreux minimums locaux. Il existe plusieurs exemple récents sur des méthodes de ce type, par exemple la méthode de recherche locale réitérée (ILS) [76] [72], ou encore la méthode multi-voisinage multi-opérateurs [77].

Les algorithmes génétiques en sont aussi un exemple en combinant la plupart du temps une structure de courte portée via la mutation et une autre de portée plus longue via le croisement [59].

Recherche Tabou [56] [57] [58] :

Proposée par Glover [51] [52] et Hansen [53], la recherche Tabou est une méthode de recherche locale qui construit un voisinage excluant un certain nombre de configurations récemment visitées ou de mouvements récemment utilisés. La méthode permet de gérer les cycles en les interdisant complètement ou pas selon les options choisies. L'idée initiale consiste à se déplacer d'une configuration à une autre tout en s'interdisant de revenir sur une configuration déjà rencontrée dans un nombre donné d'itérations antérieures (éventuellement depuis le début). Pour se faire, une liste Tabou a été introduite. Elle contient les configurations vers lesquelles il est interdit de se déplacer pendant un certain nombre d'itérations.

Cependant stocker et gérer des configurations entières est coûteux en espace et en temps de calcul. Pour éviter ces inconvénients, la méthode Tabou consiste aussi à interdire un ensemble de mouvements récemment utilisés, ces mouvements peuvent ramener à une configuration déjà visitée ou non. Dans ce cas, il n'y a plus de garantie sur les boucles mais la gestion de la mémoire est facilitée. Pour ne pas interdire d'accéder par un mouvement Tabou à des configurations de qualité supérieure à la meilleure configuration rencontrée, un critère d'aspiration a été introduit qui permet d'enlever le statut Tabou sur certains mouvements jugés utiles à un moment donné.

La durée du statut Tabou peut être soit statique soit dynamique. En durée Tabou statique, on utilise une durée fixe et constante durant toute la recherche [55]. En durée Tabou dynamique, on utilise une durée qui peut varier au cours de l'exécution. Plusieurs études comme celles réalisées par Hao et al. [54] sur les listes Tabou dynamiques ont montré que les résultats obtenus pouvaient être plus intéressants que les résultats obtenus avec une liste statique. La durée Tabou est un paramètre essentiel et important qui influence énormément les performances de la méthode ; il est donc très crucial de bien choisir les règles de gestion de la durée en fonction du problème traité. [104][105] proposent des techniques d'apprentissage pour définir empiriquement les paramètres de durée.

Afin d'améliorer l'efficacité et les performances de la recherche Tabou, Glover et Laguna [50] ont proposé plusieurs mécanismes. L'intensification et la diversification sont deux exemples.

L'idée de base de l'intensification est d'exploiter une ou des solutions prometteuses. L'intensification se fonde sur les notions liées à l'apprentissage en considérant que les attributs souvent présents dans les configurations d'élite connues sont favorables, il faut donc

les exploiter plus que les autres ; par contre les attributs rarement présents dans les configurations d'élite connues sont peu favorables, il faut donc les éviter. De nombreuses techniques ont été proposées :

- Relancer la recherche à partir des bonnes solutions déjà rencontrées.
- Reconstruire une solution de départ qui tente de combiner des attributs souvent rencontrés dans les meilleures configurations.
- Fixer certains attributs qui ont souvent été présents dans les configurations d'élite.

A contrario, la diversification est un mécanisme qui oblige la recherche à s'éloigner de zones où les solutions sont connues pour se diriger temporairement vers d'autres zones inexplorées proposant des catégories de solutions inconnues. De nombreuses techniques ont été proposées. La technique la plus répandue consiste à construire une nouvelle solution de départ et à relancer la recherche, la solution de départ peut être la plus différente possible des solutions déjà visitées. Les changements de voisinage évoqués précédemment sont aussi une forme de diversification permettant d'atteindre de nouvelles solutions.

2.2.4. Hybridation de propagation de contraintes et recherche locale

Dans cette section, nous présentons l'hybridation de techniques issues de la propagation de contraintes avec la recherche locale. Cette hybridation donne naissance à une nouvelle famille de méthodes globalement plus complexes mais aussi plus puissantes de par leurs résultats sur divers CSP que les méthodes précédentes considérées séparément. Cette partie est directement associée à nos contributions puisque nous avons élaboré une nouvelle méthode hybride basée sur la propagation de contraintes et les *nogoods* d'une part, et la recherche locale via les notions de voisinage, de statut tabou, d'intensification et de diversification d'autre part.

2.2.4.1. Analyse de l'existant et motivations pour l'hybridation

L'idée d'hybrider des techniques issues de la propagation de contraintes avec la recherche locale vient d'un ensemble d'observations sur les intérêts de chaque technique étayées notamment dans [89] [85].

D'un côté, les algorithmes basés sur le *backtracking* manipulent des configurations partielles consistantes. Ces algorithmes procèdent par extension d'une configuration partielle consistante afin d'obtenir une configuration complète qui satisfait toutes les contraintes du problème à résoudre. Ils explorent l'espace de recherche d'une manière systématique qui garantit de trouver une solution réalisable, si celle-ci existe, mais le processus peut être très long. Les combiner avec des techniques de filtrage travaillant sur des instanciations partielles des variables permet de réduire l'espace de recherche et de diminuer le temps total de calcul (utilisation de FC [27], MAC [82]…).

D'un autre côté, les algorithmes de recherche locale manipulent des configurations complètes tout en utilisant des mécanismes d'intensification afin d'explorer les zones prometteuses de l'espace de recherche et des mécanismes de diversification permettant d'échapper aux optimums locaux. Ils explorent l'espace de recherche d'une façon stochastique ne garantissant pas la découverte d'une solution. Mais l'intérêt de ces méthodes réside dans le fait qu'elles peuvent être beaucoup plus rapides que les

méthodes de recherche systématiques en utilisant la descente avec des mécanismes d'intensification et de diversification, surtout sur des problèmes de grandes tailles.

Si les algorithmes de recherche locale manipulent des configurations partielles, ils peuvent être combinés avec des techniques de filtrage et de propagation de contraintes. Sans être exhaustif, on peut par exemple éliminer des valeurs dans les domaines des variables qui ne conduisent pas à une solution complète réalisable et contribuer globalement à la réduction de l'espace de recherche pour la recherche locale. On peut aussi compléter des solutions partielles plus rapidement en évitant une démarche systématique. Ces réflexions ont conduit les chercheurs à concevoir des méthodes qui tirent leurs avantages de la formulation du problème en un CSP et de l'efficacité de la propagation d'une part, et des processus d'intensification et de diversification de la recherche locale d'autre part.

De nombreux travaux [83] [84] [85] [86] [87] [88] ont étudié la coopération entre les mécanismes que nous avons présentés dans les sections précédentes. Jussien et Lhomme [91] les ont classifiés en 3 approches :

1. La recherche locale comme outil d'amélioration de solutions partielles, comme c'est le cas dans [89] [90].
2. La recherche locale comme outil de réparation de solutions partielles, comme c'est le cas dans [87] [91] [92] [94].
3. La recherche locale sur un voisinage consistant, comme c'est le cas de *CN-Tabu* [93].

Dans notre travail, nous nous sommes intéressés particulièrement à deux méthodes hybrides qui nous paraissent intéressantes de part leurs mécanismes et leurs performances sur les CSP, la méthode *CN-Tabu* et la méthode *Decision-Repair*. Ces deux méthodes ont été appliquées sur des problèmes de natures différentes et ont montrées de très bonnes performances.

2.2.4.2. CN-Tabu (Tabu Search on a Consistent Neighbourhood)

Vasquez et al. [93] proposent une méthode de recherche Tabou qui travaille dans l'espace des configurations partielles consistantes. Cette méthode appelée *CN-Tabu* pour *Tabu Search on a consistent Neighbourhood* appartient à la troisième approche des méthodes hybrides. Le voisinage considéré à chaque itération comporte toutes les solutions atteignables à partir de la solution courante en appliquant un double opérateur :

- Instanciation d'une variable non encore affectée à une valeur non taboue de son domaine.
- Puis réparation éventuelle de la solution afin de maintenir la consistance ; les variables en conflit avec la nouvelle affectation sont désaffectées.

L'originalité de la méthode concerne l'évaluation de la qualité d'une solution. La fonction de coût utilise en effet le nombre de variables instanciées comme critère à maximiser. A chaque itération, l'instanciation non taboue choisie est celle qui induit le moins de variables à désaffecter pour le maintien de la consistance.

La liste Tabou, gérée de façon dynamique, est maintenue à l'issue de chaque mouvement par l'ajout des valeurs des domaines des variables non instanciées qui entreraient en conflit

avec la dernière affectation réalisée. Ainsi, le dernier mouvement réalisé est préservé durant un certain nombre d'itérations. Un critère d'aspiration permet de lever le statut Tabou d'un mouvement permettant d'améliorer la valeur de la fonction objectif.

Cette méthode ne fait jamais appel à des mouvements de type recherche arborescente. L'aspect hybride est ici apporté par le travail sur les configurations partielles et les techniques de filtrage (voir Figure 11).

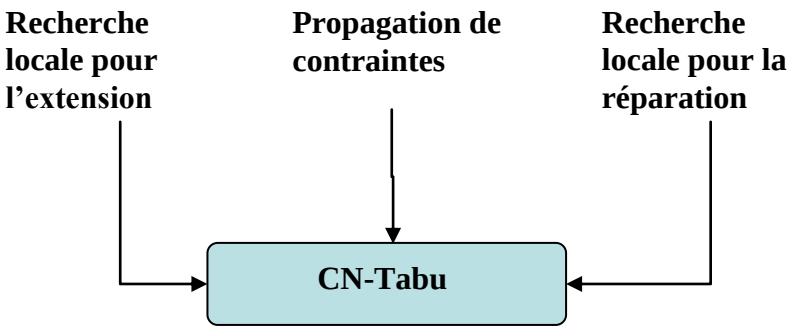


Figure 11 : *CN-Tabu* - Hybridation de la propagation de contraintes avec la recherche locale

2.2.4.3. Decision-Repair

Proposée par Jussien et Lhomme [94] [91], cette méthode effectue une recherche locale sur des instanciations partielles ce qui lui permet d'utiliser des techniques de filtrage et des techniques de retour (look-back) basées sur les conflits (ou *nogoods*) pour guider efficacement la recherche. Les *n* derniers *nogoods* identifiés pendant la recherche sont stockés dans une liste Tabou Γ des *nogoods*, cette liste sert à de ne pas retomber dans les mêmes échecs pendant la recherche.

Decision-Repair bénéficie ainsi de deux avantages : réduire l'espace de recherche par une procédure de filtrage et réparer les éventuelles premières erreurs d'affectation (instanciation conduisant nécessairement à un blocage) faites par la recherche locale à partir des *nogood*s produits pendant le filtrage.

Decision-Repair est un schéma générique de fonctionnement, dans sa version *Tabu-Decision-Repair*, on procède comme suit :

1. La méthode tente d'étendre la configuration partielle courante en appelant la procédure d'extension. Cette procédure choisit une variable non encore instanciée et une valeur légale qui maintient la consistance. Le couple (variable, valeur) choisi doit respecter les *n nogood*s stockés dans la liste Tabu des *nogood*s. Autrement dit, aucun *nogood* stocké dans la liste Tabou des *nogood*s ne doit figurer comme un sous ensemble de la configuration étendue. Le premier couple trouvé qui remplit cette condition est choisi.
2. Après l'extension de la configuration partielle courante, un algorithme de filtrage est lancé afin d'éliminer les valeurs inconsistantes dans les domaines des variables non instanciées :
    a. Si un domaine d'une variable devient vide => Un *nogood ng* est identifié, il sera stocké dans la liste Tabou Γ de taille *n* (FIFO : First In First Out), et une procédure de réparation en fonction du *nogood ng* identifié est lancée.

Cette procédure consiste à sélectionner une affectation parmi celle présente dans la configuration partielle courante afin d'être annulée.

b. Sinon :

   i. Si la solution est complète => solution trouvée

   ii. Sinon, on appelle à nouveau la procédure d'extension.

Dans la procédure d'extension d'une configuration partielle, *Decision-Repair* agit classiquement comme n'importe quelle approche basée sur le retour. Cependant, lors d'un échec, *Decision-Repair* identifie un *nogood ng* responsable de l'inconsistance et fait appel à une procédure pour réparer la configuration partielle courante. Cette procédure est responsable de la sélection d'une configuration voisine consistante. Cette configuration voisine est obtenue en désaffectant une variable de la configuration partielle courante qui appartient à *ng*.

Nous remarquons que *Decision-Repair* dans la version proposée dans [91] [94], reste une méthode qui explore plus ou moins l'espace de recherche d'une manière systématique et qui ne bénéficie pas de la diversification apportée par une recherche Tabou classique car la variable désaffectée n'est pas mémorisée. Cette méthode hybride 3 principales composantes (Figure 12), elle utilise la propagation de contraintes afin de réduire l'espace de recherche, le *Nogood-Recording* afin d'éviter la redondance dans la recherche et la recherche locale afin de réparer une configuration partielle non consistante.

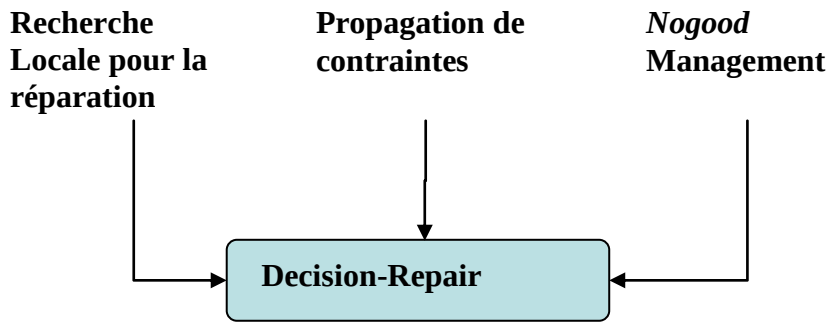


Figure 12 : *Decision-Repair* : Hybridation de recherche locale, propagation de contraintes et *Nogood-Recording*

2.2.4.4. *Réflexions sur l'hybridation des méthodes*

Recherche locale et recherche systématique sont deux paradigmes de résolution très différents. La première est une approche incomplète mais robuste, c'est une technique d'optimisation qui a une certaine liberté pour parcourir l'espace de recherche ; elle est capable de fournir toujours une solution *approchée* et la plupart du temps de bonne qualité en temps raisonnable. La deuxième est complète mais sa complexité est exponentielle, elle est capable de bénéficier de la réduction de l'espace de recherche en exploitant activement les contraintes du problème ; cependant, la tâche d'optimisation est difficile à intégrer dans ce genre d'approches puisque l'ordre d'instanciation des variables est rigide et on se déplace difficilement dans l'espace de recherche d'une façon horizontale. La recherche systématique peut aussi bénéficier de l'intégration de mécanismes d'apprentissage en stockant des *nogoods*

et/ou des *goods* identifiés. Ces outils peuvent permettre de guider efficacement la recherche vers des zones prometteuses.

La question qui se pose est la suivante : pour résoudre un CSP, quel type d'approches faut-il utiliser ? Actuellement personne ne peut donner une réponse convaincante et définitive à cette question générale. Cependant, Freuder [106] a posé une question intéressante : au lieu de faire la compétition, peut-on coopérer ?

Dans la littérature, l'hybridation des méthodes que ce soit dans un cadre collaboratif (en les lançant les unes après les autres ou parallèlement) ou dans un cadre intégratif (en exploitant au sein d'une nouvelle méthode les points forts de méthodes combinées) a montré son efficacité[89][91][93][95], en particulier l'hybridation entre la PPC et la recherche locale [83][87][89][91][93]. La méthode hybride *CN-Tabu* proposée par Vasquez [93] est une de ces méthodes ; elle a prouvé son efficacité et sa rapidité en résolvant à l'optimalité 13 instances de satisfiabilité sur les 14 proposées dans le cadre du projet européen CALMA [119][120], et 38 instances sur les 44 traitées dans le cadre du challenge ROADEF 2001 [122]. De même la méthode hybride *Decision-repair* proposée par Jussien [94][91] a fait ses preuves d'efficacité et de rapidité sur des problèmes d'*open-shop* difficiles ; elle s'est montrée nettement meilleure [91] que les meilleures approches connues sur ce type de problème (2 Recherche Tabou et un algorithme génétique).

Bien entendu pour ces méthodes, comme pour d'autres, on peut aussi trouver des inconvénients ou des défauts dans leur fonctionnement qui peuvent ouvrir quelques pistes nouvelles dans l'hybridation des méthodes. Mais restons modestes, qui dit pistes nouvelles, ne dit pas forcément améliorations garanties ! Encore faut-il faire ses preuves lors de la mise au point de nouveaux mécanismes. Nous citons ci-après quelques points de progrès qui nous semblent intéressants.

CN-Tabu manipule des configurations partielles afin de profiter de la réduction de l'espace de recherche et elle applique une Recherche Tabou classique sur un voisinage consistant. Cette méthode bénéficie de la réduction de l'espace de recherche apportée par la manipulation des configurations partielles et de la liberté de la Recherche Tabou à se déplacer dans l'espace de recherche en utilisant les mécanismes d'intensification et de diversification. Cependant, *CN-Tabu* souffre du caractère aveugle de certains de ses choix et notamment pour réparer l'inconsistance. Par exemple *CN-Tabu* peut tomber plusieurs fois dans les mêmes erreurs pendant la recherche (exploitation redondantes de certaines branches) ; elle n'exploite pas les informations rencontrées afin de guider sa recherche vers de nouvelles branches prometteuses.

Decision-Repair utilise légèrement la recherche locale dans la réparation d'une configuration partielle inconsistante. Cette méthode applique un algorithme de filtrage sur chaque nœud de l'arbre de recherche afin de réduire l'espace de recherche et stocke les coupes non prometteuses afin de ne pas refaire les mêmes erreurs ; les coupes sont par la suite utilisées pour guider d'une manière efficace la méthode vers les branches prometteuses. Cependant, cette méthode n'est pas libre dans le déplacement dans l'espace de recherche, elle souffre de son caractère systématique qui peut rendre difficile la diversification de sa recherche.

Cependant l'hybridation est un point fort des deux méthodes. Sur la base de l'analyse de l'existant dans le domaine de la propagation de contraintes, de la recherche locale et de l'apprentissage des solutions, nous proposons une nouvelle méthode qui bénéficie à la fois de la réduction de l'espace de recherche (configurations partielles et algorithmes de filtrage sur chaque nœud de l'arbre), de l'expérience acquise pendant la recherche afin de guider

efficacement la méthode (*nogoods*), et de la liberté offerte par la recherche Tabou pour naviguer dans l'espace de recherche (liste Tabou).

2.3. *Tabu-NG* : *Tabu Search based on NoGoods*, une nouvelle méthode hybride

2.3.1. Principes généraux de la méthode proposée

L'idée de *Tabu-NG* (*Tabu Search based on NoGood*) est d'hybrider le *backtracking*, la recherche locale et les techniques de filtrage issues de la PPC. *Tabu-NG* se base sur des principes utilisés par les méthodes *CN-Tabu* et *Decision-Repair*, elle essaye de mixer les deux afin de donner naissance à une nouvelle méthode dont on attend qu'elle exploite les aptitudes de mémorisation et d'apprentissage venant à la fois des principes du *Nogood-Recording* (interdits sur les configurations partielles non-consistantes) et des listes Tabou (interdits sur les configurations partielles déjà visitées).

Tabu-NG utilise deux listes Tabou :

1. Une première liste Tabou des *nogood*s nommée Γ (la même que celle utilisée par *Decision-Repair*) qui contient les *nogoods* identifiés au fur et à mesure de la recherche ; on peut imposer à la liste de stocker les *n* (paramètre à préciser) derniers *nogoods* identifiés pendant la recherche afin d'éviter la croissance exponentielle de l'utilisation de la mémoire pour le stockage. Ainsi, *Tabu-NG* utilise cette liste pour mémoriser les échecs et éviter de revisiter des branches déjà explorées.
2. Une deuxième liste Tabou nommée Δ (jouant le même rôle que dans *CN-Tabu*) qui contient pour chaque variable ou pour chaque instanciation *(variable, ressource)* le numéro de l'itération à partir de laquelle la variable ou l'instanciation est autorisé. Autrement dit c'est une liste qui permet de vérifier le statut Tabou d'une variable ou d'une instanciation (paramètre à préciser dans la méthode). Cette liste est utile pour pousser la recherche à diversifier ces choix d'instanciation, ce qui permet à la méthode de changer plus facilement de branche dans l'arbre de recherche.

Avec la manipulation des configurations partielles, l'intérêt de la deuxième liste est d'éviter les cycles de recherche comme pour une Recherche Tabou classique, de rendre l'exploration de l'espace de recherche moins systématique et de bénéficier de la diversification apportée par ce genre de liste Tabou dans les Recherches Tabou classiques. Cette liste est intéressante pour les problèmes de grande taille ou la diversification est un élément important de l'efficacité de la méthode de résolution.

Tabu-NG est une méthode constructive qui se base fondamentalement sur le maintien permanent de la consistance de configurations partielles successives jusqu'à atteindre une configuration complète, solution du CSP. Elle utilise trois mécanismes complémentaires : la propagation de contraintes afin de réduire l'espace de recherche, le *Nogood-Recording* pour apprendre de la recherche passée et ainsi éviter la redondance, et la recherche locale pour la réparation et pour l'extension d'une configuration partielle (voir Figure 13).

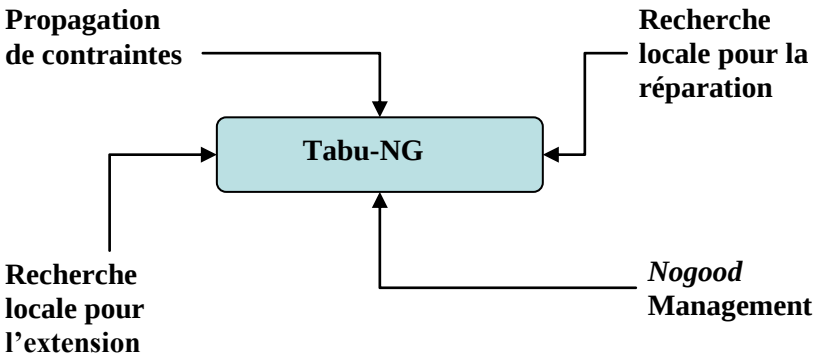


Figure 13 : *Tabu-NG* : Une hybridation de la propagation de contraintes, de *Nogood-Recording* et de recherche locale

L'organigramme de la Figure 14 montre le fonctionnement global de *Tabu-NG*.

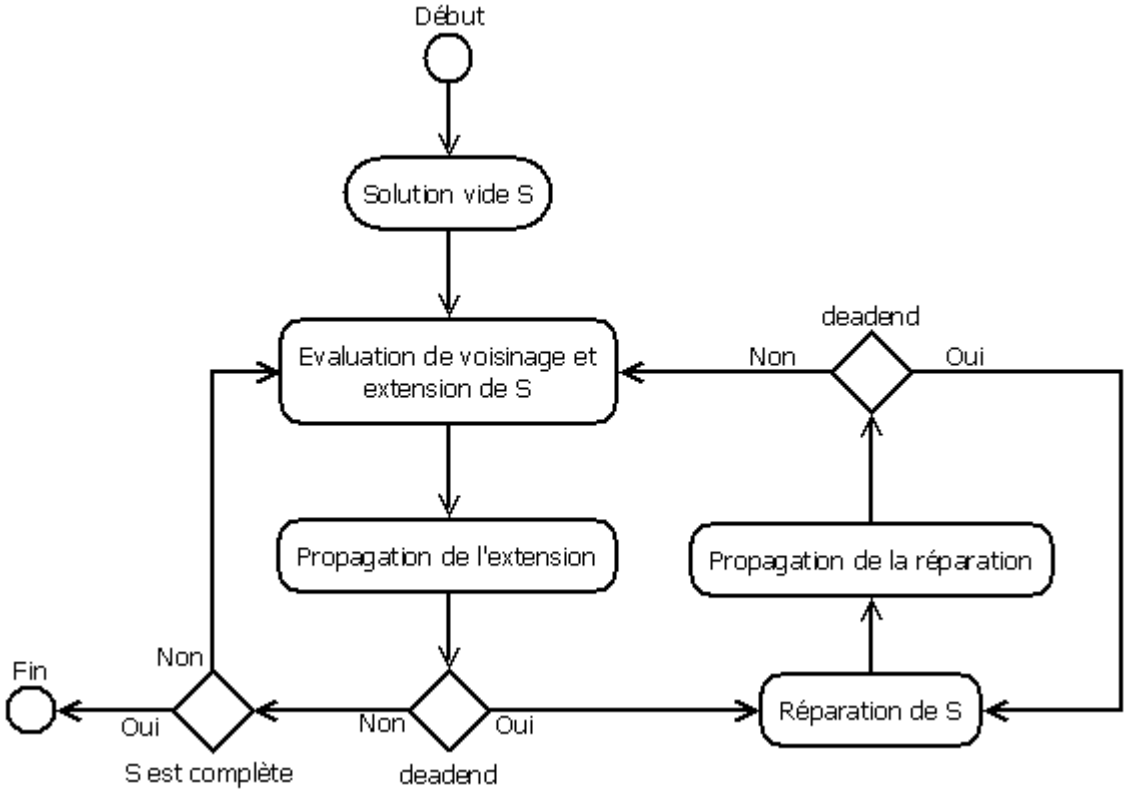


Figure 14 : Organigramme global de *Tabu-NG*

Cette méthode commence la recherche par une configuration partielle vide *S*. A chaque itération, le voisinage est évalué et la configuration est étendue si possible vers une configuration voisine consistante. La notion de *meilleure* fait référence à un critère d'optimisation spécifié. Ensuite, une phase de propagation démarre afin de propager l'effet de l'extension faite sur *S* qui peut aboutir à l'une des deux réponses suivantes :

1. La configuration partielle courante *S* donne lieu à un *deadend* avec un *nogood* responsable de cet échec. Le *nogood* identifié est stocké dans Γ . Puis la méthode lance une phase de réparation de la configuration partielle courante *S*. Cette phase annule une ou plusieurs instanciations déjà faites. Les instanciations annulées, ou les variables de ces instanciations selon la liste Tabou utilisée, seront considérées

comme Tabou pour un nombre donné d'itérations. Après la phase de réparation, une procédure de propagation est lancée afin de propager l'effet de la réparation faite sur *S*.

Aucun domaine de variable n'est vide. On teste si la nouvelle configuration est complète, si elle l'est la méthode arrête la recherche avec une solution trouvée *S*, sinon la méthode continue la recherche en essayant à nouveau d'étendre *S*.

Dans les sections suivantes nous voyons en détail le fonctionnement de chaque étape de la méthode. Ce fonctionnement est générique, indépendant de tout problème particulier de type CSP. Les deux chapitres suivants nous permettront d'aborder la méthode dans un contexte précis pour voir comment elle s'adapte.

2.3.2. Evaluation du voisinage et extension d'une configuration

A partir d'une configuration partielle consistante, la méthode *Tabu-NG* peut adopter une heuristique afin de choisir une configuration voisine intéressante comme pour une méthode de recherche locale classique. La méthode peut aussi choisir une configuration voisine de telle façon que la méthode devienne une méthode de recherche systématique. Dans ce cas il suffit d'agir exactement comme *Decision-Repair* et ne pas tenir compte de la liste Tabou Δ.

Cette partie explique comment choisir une configuration voisine pour obtenir une exploration non systématique. A noter que l'heuristique du voisinage pour *Tabu-NG* est un paramètre de la méthode.

2.3.2.1. Procédure d'extension

La procédure de l'extension vise à choisir une nouvelle instanciation, un couple (variable, ressource) pour l'insérer dans la configuration partielle courante. Autrement dit, on cherche à instancier une variable libre. Les configurations voisines d'une configuration partielle courante sont aussi des configurations partielles (sauf dans le cas où il ne reste qu'une seule variable libre, dans ce cas une solution réalisable est trouvée).

Etant donné qu'une procédure de filtrage est lancée après chaque extension de la solution conduisant à éliminer les valeurs inconsistantes dans les domaines des variables libres, alors n'importe quelle valeur choisie dans le domaine filtré d'une variable libre respecte absolument la consistance de la configuration partielle courante.

Afin de garder l'aspect constructif de notre méthode et respecter l'architecture en graphe du problème, nous avons défini deux règles pour le choix de la nouvelle affectation :

1) La variable *v* à affecter est choisie selon une heuristique définie comme par exemple *MRV* pour *Minimum Remaining Values* ou *HD* pour *High Degree*. Le choix de cette heuristique est un paramètre de la méthode.

   Dans le cas où la variable seule est un attribut dans la liste Tabou Δ, une deuxième condition s'ajoute :

   $$\Delta[v] < IterationCourante$$

   Il ne faut pas que la variable choisie soit interdite.

Dans le cas où la liste Tabu Δ manipule des affectations (variable, ressource) une deuxième condition s'ajoute :

$$\Delta[v,a] < IterationCourante$$

Il ne faut pas que le couple (variable, ressource) choisi soit interdit.

2) Le choix de la ressource a à affecter est fait de la façon suivante :

$$a \in D_v$$
$$\forall ng \in \Gamma, ng \not\subset (S \bigcup (v = a))$$

Avec S la configuration partielle courante.

Littéralement aucun *nogood* stocké dans Γ ne doit figurer dans la nouvelle configuration partielle.

Afin de bien définir le voisinage et afin d'éviter que la méthode boucle sur des choix non corrects, une condition supplémentaire est vérifiée à chaque extension de la configuration courante :

Si toutes les valeurs a appartenant à D_v vérifient la condition :

$$\exists ng \in \Gamma / ng \subset (S \bigcup (v = a))$$

On considère que l'extension de la configuration partielle courante est impossible et que la taille du voisinage est égale à 0. Dans ce cas, un nouveau *nogood* est repérée et la procédure de réparation de la configuration partielle courante est lancée.

2.3.2.2. Critère d'aspiration

Nous utilisons la procédure d'aspiration de la Recherche Tabou pour gérer des cas particuliers lors de l'extension de la configuration courante et éviter les blocages de l'algorithme de choix.

1) Dans le cas où le couple (variable, ressource) est pris en compte dans la liste Tabou Δ :
   - Si une variable est très visitée, tous les choix possibles d'affectation la concernant peuvent devenir tabous. Ce qui nécessite un basculement vers une autre variable.
   - Si pour toutes les variables libres, tous les couples (variable, ressource) sont tabous, alors on annule le statut Tabou pour tous les couples de Δ.
2) Dans le cas où la variable seule est dans la liste tabou Δ :
   - Si toutes les variables libres sont en statut Tabou, alors on annule le statut Tabou de toutes les variables de Δ.

2.3.2.3. Algorithme détaillé

La procédure d'extension d'une configuration partielle est donnée dans Algorithme 6. Il décrit l'algorithme nécessaire pour étendre une configuration partielle *S* avec MRV comme heuristique de choix de la variable et une liste Tabou Δ qui gère des variables sans leur affectation. Dans ce cas Δ est un vecteur de taille *nombre de variables*.

La fonction *GetVarMRVNonTabu* retourne une variable non Tabou (selon Δ) et qui possède le plus petit domaine. Cette fonction gère aussi le critère d'aspiration.

nbreMvt est un vecteur (de taille *nombre de variables*) qui comptabilise le nombre de fois où une variable est choisie par la procédure d'extension. Autrement dit, le nombre de fois où une variable a été visitée. Il s'agit d'une mesure de fréquence d'utilisation de la variable qui reflète l'intensité de la recherche autour de cette variable.

Si Δ gérait des couples, *nbreMvt* serait une matrice (de taille *nombre de variables * taille maximale des domaines*) qui stockerait le nombre de fois où une instanciation est choisie par la procédure d'extension. Cette matrice nous aidera à définir une durée Tabou adaptative pour Δ.

```
Function EtendreConfiguration(X, D, C, S, Δ, Γ)
1.    ListeVarsLibres = GetAllVarLibres(X)
2.    var = GetVarMRVNonTabu(ListeVarsLibres, Δ)
3.    foreach a in D_v do
4.          if isInListeNogood(S, var, a, Γ) == false then
5.                nbreMvt(var) += 1;
6.                return (var, a);
7.    endforeach
8.    ReparerConfiguration(S, ng)
9.    EtendreConfiguration(X, D, C, S, Δ, Γ)
```

Algorithme 6 : Etendre une configuration partielle

La fonction *isInListeNogood* (voir Algorithme 7) vérifie l'existence d'un *nogood* dans la configuration partielle *S*. Cette fonction a pour complexité *O(t*tng)* avec *t* la taille de la liste Γ et *tng* la taille maximale d'un *nogood ng* appartenant à Γ. Dans la pratique, sur nos expérimentations cette complexité maximale n'est jamais atteinte ; en réalité, la complexité moyenne est beaucoup moins importante.

```
Function isInListeNogood(S, var, a, Γ)
1.    bool isInListe = true;
2.    foreach ng in Γ do
3.          foreach couple in ng do
4.                if not couple in S ∪ {var,a} then // O(1)
5.                      isInListe = false; break;
6.                endif;
7.          endforeach;
8.          if isInListe == true then return ng;
9.    endforeach;
10.   return NULL;
```

Algorithme 7 : Vérifier l'appartenance d'une configuration partielle à la liste des *nogoods*

L'organigramme de la Figure 15 montre le déroulement global de la phase d'extension d'une configuration partielle *S*. La procédure d'extension de *S* vise à étendre *S* en choisissant un nouveau couple (variable, ressource). La variable sélectionnée *var* doit être libre et non interdite i.e. elle n'est pas en statut Tabou dans la liste Δ. La ressource *a* qui doit être affectée à *var* doit respecter la liste Γ des *nogoods*. Autrement dit, aucun *nogood* de Γ ne doit figurer comme un sous-ensemble de la nouvelle configuration étendue.

Dans le cas où pour une variable choisie *var*, aucune ressource disponible ne respecte les règles définies, alors un nouveau *nogood* est repéré, la phase de réparation est lancée et ensuite la phase d'extension est appelée à nouveau.

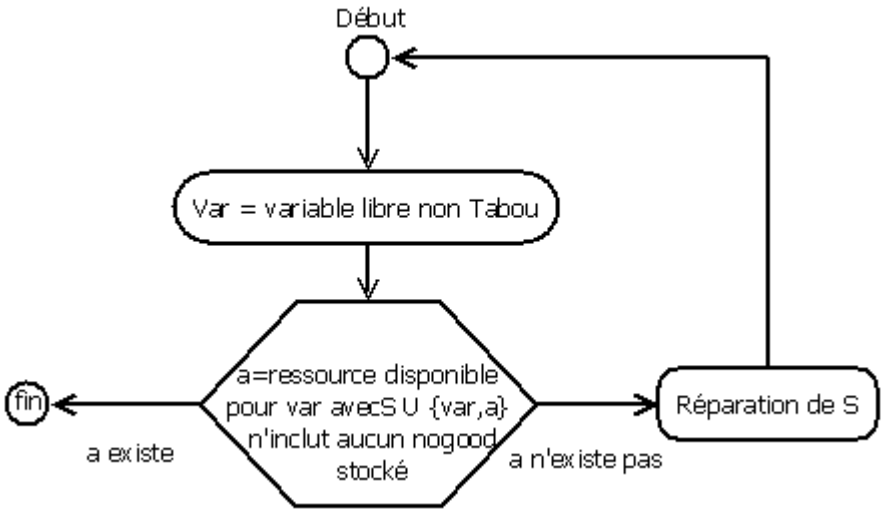


Figure 15 : Fonctionnement de l'extension d'une configuration partielle

2.3.3. Filtrage des domaines

L'algorithme de filtrage est utilisé par notre méthode après chaque extension d'une configuration partielle. Cet algorithme est responsable de la propagation de l'effet de l'ajout d'une nouvelle affectation à la configuration partielle courante *S* sur l'ensemble des variables libres. L'algorithme de filtrage donne deux réponses possibles :

1. Le domaine d'une variable libre est vide alors un n*ogood* est identifié.
2. Aucun domaine n'est vide.

Pour des *CSP* sur des domaines finis avec des contraintes binaires seulement, il est commun d'utiliser des algorithmes de filtrage par arc-consistance (AC3, AC-4, AC-6, etc.).

L'algorithme de filtrage sera choisi dans la méthode selon le type des contraintes à traiter. Parfois on peut propager d'une manière très efficace des contraintes complexes grâce à leurs propriétés mathématiques [10].

Dans la partie applicative de cette thèse, nous détaillerons les algorithmes de filtrage utilisés pour chaque type de contraintes.

2.3.4. Réparation d'une configuration partielle

2.3.4.1. Etape de désaffectation

L'algorithme de filtrage s'applique après l'extension de la configuration partielle courante *S*. Lorsqu'un *deadend* est atteint, un *nogood* est identifié et la solution courante doit être réparée.

La réparation de *S'*se fait en se basant sur le *nogood* identifié car il contient la suite d'instanciation ayant conduit à cet échec.

Le but de la procédure de réparation est de choisir la, ou les instanciations à annuler parmi celles présentes dans le *nogood* identifié. Plusieurs alternatives sont offertes : soit annuler l'instanciation la plus récente, soit annuler toutes les instanciations appartenant au *nogood* identifié, ou bien choisir aléatoirement une décision à annuler. L'heuristique du choix d'une ou plusieurs instanciations à annuler est donc un paramètre de la méthode. Nous allons voir dans la partie applicative de cette thèse les heuristiques que nous avons testées.

Pour les problèmes d'affectation de fréquences, nous avons utilisé la même heuristique de réparation que celle utilisée par la méthode *Decision-Repair* [91] : un poids est associé à chaque couple (variable, ressource), ce poids est calculé d'une façon dynamique pendant la recherche (structure de données maintenue pendant la recherche). Après chaque détection d'un *nogood*, le poids de tous les couples (variable, ressource) appartenant au *nogood* détecté est mis à jour selon la formule suivante, sachant que le poids initial (au début de la recherche) de chaque couple est nul :

$$poids(\text{var}, a) = poids(\text{var}, a) + 1/|nogood|$$

La cardinalité du *nogood* correspond simplement au nombre d'instanciations qu'il contient. Après chaque *deadend* l'instanciation de plus grand poids dans le *nogood* courant est annulée, la variable est rendue libre. Si plusieurs instanciations sont de même poids on prend la première rencontrée.

Malgré que cette heuristique ait montré de bonne performance sur les problèmes d'affectation de fréquences, ce n'est pas toujours suffisant d'annuler une seule instanciation à la fois. Nous avons parfois besoin, surtout pour des problèmes structurés et de grandes tailles, d'annuler plusieurs instanciations à la fois afin de pousser la méthode à diversifier sa recherche et s'échapper des optimums locaux. Pour les problèmes de coloration de graphe, une heuristique de réparation de ce type a été nécessaire afin d'améliorer l'efficacité de *Tabu-NG*.

2.3.4.2. Etape de mémorisation des affectations annulées

A ce stade nous utilisons la liste Tabou Δ pour mémoriser la ou les affectations annulées par la phase de réparation ; elles sont considérées interdites ou taboues pour un nombre d'itérations donné. Cette durée est un paramètre de la méthode.

Nous utilisons une durée adaptative qui est égale au nombre de fois où la même affectation a été choisie par la procédure d'extension, et cela indépendamment des configurations partielles visitées. Nous allons voir que cette durée varie en fonction du problème à résoudre et d'une instance à une autre.

$$\Delta(\text{var}, a) = iterationCourante + nbreMvt(\text{var}, a)$$

Dans le cas où la liste tabou Δ manipule seulement des variables, la durée taboue sera de la forme suivante :

$$\Delta(\text{var}) = iterationCourante + nbreMvt(\text{var})$$

L'algorithme de réparation d'une configuration partielle *S* en fonction d'un n*ogood ng* identifié est donné dans Algorithme 8. La phase de réparation est lancée sur une configuration partielle non consistante *S*. Cette phase doit établir à nouveau la consistance de *S* en annulant au moins une instanciation faite. L'instanciation à annuler doit absolument appartenir au *nogood* responsable de l'inconsistance. L'algorithme décrit dans Algorithme 8 prend en entrée la configuration partielle inconsistante *S* à réparer, la durée Tabou ainsi que le *nogood ng* responsable de l'échec. Dans un premier temps, on choisit l'affectation à annuler à partir de celles appartenant au *nogood ng*. Cette instanciation (ou seulement la variable correspondante) est mise en durée Tabou. Enfin la configuration *S* est réparée en annulant l'instanciation choisie.

Function ReparerConfiguration(*S, Δ, ng*)
1. *(var,a)* = GetAffectationPourAnnuler(*ng*)
2. *Δ(var, a) = iterationCourante + nbreMvt(var, a)*
3. *S = S\(var, a)*
4. return *(var, a)*

Algorithme 8 : Réparer une configuration partielle

2.3.4.3. Etape de mémorisation des nogoods identifiés

La liste Γ contient tous, ou les *n* derniers, *nogoods* identifiés pendant la recherche. Si on stocke dans Γ tous les *nogoods* identifiés pendant la recherche, on pourra garantir la complétude de *Tabu-NG* quel que soit le type de voisinage utilisé. Et par la suite, *Tabu-NG* sera capable de donner une réponse exacte sur la satisfiabilité d'un problème donné. Si un problème est irréalisable, on trouvera à la fin de la recherche un seul *nogood* de taille = 0 qui représente la totalité de l'espace de recherche.

La taille de la liste tabou Γ croit d'une façon exponentielle en fonction du temps pendant la recherche, mais en utilisant la notion de dominance entre les *nogoods* pendant l'ajout des *nogoods* dans la liste Γ, on pourra gérer la taille de Γ et rendre sa croissance en taille mémoire en fonction du temps beaucoup moins importante. Ce point sera étudié de près dans le chapitre dédié à la résolution des problèmes d'affectation de fréquences par *Tabu-NG*.

L'algorithme d'ajout d'un nouveau *nogood* à la liste Γ est donné dans Algorithme 10. L'idée est de supprimer de Γ tous les *nogoods* qui sont dominés par le nouveau *nogood* à insérer. Le nouveau *nogood ng* identifié pose seulement deux cas :

1. *ng* domine un ensemble de *nogoods* stockés dans Γ. Dans ce cas tous les *nogoods* dominés sont supprimés de Γ, et seul *ng* est ajouté dans Γ.
2. *ng* ne domine aucun *nogood* stocké dans Γ. Dans ce cas *ng* est ajouté dans Γ.

Dans *Tabu-NG*, on ne peut pas tomber sur un *nogood* lui-même dominé par des *nogoods* déjà stockés dans Γ puisque la phase d'extension d'une configuration partielle *S* choisit une voisine de *S* selon une règle qui interdit ce comportement.

Définition 23 : Dominance des *nogoods*

Un *nogood ng'* domine un autre *nogood ng,* ou *ng* est dominé par *ng'* si et seulement si :

$$|ng'| < |ng| \text{ et } ng' \subset ng$$

```
Function AjouterNogood(Γ, nog)
1.    bool estInclus = true;
2.    foreach ng in Γ do
3.        if |ng|>|nog| then
4.            foreach couple in nog do
5.                if not couple in ng then // O(1)
6.                    estInclus = false;
7.                    break;
8.                endif;
9.            endforeach;
10.           if isInListe == true then
11.               Γ= Γ\ng
12.           endif;
13.       endif;
14.   Γ.push(nog)
15.   endforeach;
```

Algorithme 9 : Ajouter un *nogood* à la liste des *nogoods*

Pour ajouter un nouveau *nogood ng* à *Γ*, il faut chercher tous les *nogoods* dominés par *ng* et les supprimer avant l'ajout de *ng*. L'Algorithme 9 réalise les étapes ci-après. Pour chaque *nogood ng* stocké dans *Γ* on teste si *ng* est dominé par le nouveau *nogood nog* à ajouter. Cependant, si la taille de *ng* est plus grande que la taille de *nog* alors on est sûr que la dominance n'est pas assurée entre les deux *nogoods*, et il faut tout de suite passer au *nogood* suivant dans *Γ*. Cette condition diminue d'une façon considérable la complexité moyenne de l'algorithme d'ajout.

2.3.5. *Tabu-NG* : Algorithme général

Combinant tous les éléments introduits précédemment, la méthode *Tabu-NG* est décrite dans l'Algorithme 10. L'algorithme prend en paramètre d'entrée l'ensemble des variables *X*, l'ensemble des domaines *D*, l'ensemble des contraintes *C* et une configuration partielle vide de départ.

L'algorithme commence par une phase de prétraitement en appelant une fonction *Algo-filtrage* qui élimine toutes les valeurs inconsistantes avant de commencer la phase de recherche. Si un domaine d'une variable devient vide, alors le problème est identifié comme irréalisable. Il s'agit donc d'une phase de prétraitement, qui est parfois très utile pour les problèmes réels. Cette phase peut réduire la taille du problème et par la suite l'espace de recherche avant de lancer une méthode de résolution.

Après cette étape, la méthode tente d'étendre la configuration partielle courante *S*. Ensuite une fonction appelée *AlgoFiltrageApresExtension*, qui propage seulement l'effet de l'ajout d'une nouvelle instanciation, est appelée. Cette fonction est aussi capable d'identifier le *nogood* responsable de l'échec en cas de *deadend*. Si la nouvelle configuration étendue n'est

pas un *deadend*, *Tabu-NG* continue à étendre S afin d'obtenir une configuration complète. Sinon, l'extension est un *deadend*, alors un *nogood ng* est identifié et *Tabu-NG* tente une réparation.

Après la réparation de *S*, on lance l'algorithme *AlgoFiltrageApresReparation* qui propage l'effet du retrait d'une affectation de *S*. Nous savons qu'à chaque fois qu'une nouvelle instanciation est ajoutée à la configuration partielle *S*, on propage l'effet de cette instanciation sur le reste du graphe. De la même façon, lorsqu'une instanciation est annulée, il faut propager l'effet de son annulation sur le reste du graphe.

Les algorithmes de filtrage seront détaillés dans les parties propres à l'application de la méthode sur les problèmes traités.

```
Procedure Tabu-NG(X, D, C, S)
1.    iterationcourante = 0
2.    //S est La solution initiale = ensemble vide.
3.    Resultat = AlgoFiltrage(X, D, C)
4.    if Resultat = EMPTY_DOMAIN then
5.        return Infeasible;
6.    endif
7.    DF = D //DF est le domaine filtré des variables appartenant à X
8.    repeat
9.     if Conditions d'arrets sont atteints then
10.        return SolutionNonTrouvée
11.    else
12.        if ng not Null then  //deadend et identification d'un nogood
13.            AjouterNogood(Γ, ng)
14.            (var,a) = ReparerConfiguration(S, Δ, ng)  // chercher un voisinage
15.            AlgoFiltrageApresReparation (DF, S, C, var, a, ng, Δ)
16.        else if S est complète then return S //problème réalisable
17.        else
18.            (var, a) =EtendreConfiguration(X, DF, C, S, Δ, Γ)
19.            ng = AlgoFiltrageApresExtension(DF, S, C, var, a)
20.        endif
21.    endif
22.    iterationcourante++;
23.   until false
```

Algorithme 10 : *Tabu-NG* : Algorithme général

Tabu-NG est une méthode qui possède des paramètres : la taille de la liste *Γ*, la durée Tabou utilisée par la liste *Δ*, le type du voisinage utilisé, l'heuristique de réparation d'une configuration partielle et les algorithmes de filtrage utilisés… *Tabu-NG* est une méthode de résolution d'un CSP qui est composé de plusieurs composants, il faut adapter ses composants en fonction du problème à résoudre, de ses propriétés physiques et mathématiques comme pour une métaheuristique.

Nous verrons la résolution de différents problèmes avec *Tabu-NG* dans les parties qui suivent. Ces résolutions laissent à penser que *Tabu-NG* est paramétrable facilement pour résoudre des problèmes de natures différentes.

2.4. Synthèse

Dans ce chapitre, nous avons dans un premier temps donné une introduction sur les problèmes de satisfaction de contraintes ou *Constraint Satisfaction Problem* (CSP), décrit le modèle général CSP, ainsi que quelques exemples de problèmes connus avec leurs modèles CSP équivalents. Dans notre travail nous nous intéressons plus particulièrement aux méthodes de résolution hybrides des CSPs, et plus précisément à la combinaison des techniques issues de la programmation par contraintes avec la recherche locale.

La programmation par contraintes est un paradigme de programmation qui sépare la modélisation d'un problème de sa résolution et se base principalement sur la propagation de contraintes pour réduire l'espace de recherche. Nous avons donc introduit les méthodes exactes de résolution des CSP, en détaillant tout d'abord les techniques de propagation de contraintes qui sont des techniques de réduction de l'espace de recherche et non des techniques de résolution. Par la suite nous avons introduit le *backtracking*, un algorithme de recherche systématique qui est la base de presque toutes les méthodes exactes de résolution des CSPs. Nous avons aussi introduit les approches d'amélioration du *backtracking*. la première approche (appelée *look-back*) consiste à rendre le *backtracking* intelligent en revenant sur des variables critiques dans l'arbre de recherche au lieu de retourner toujours à la dernière variable instanciée en analysant et en exploitant la source de conflit. Cette approche peut utiliser le *Nogood-Recording* qui consiste à stocker des configurations partielles (appelées *nogood*) conduisant à des *deadends* dans une liste afin d'éviter la redondance dans la recherche. La deuxième approche (appelée *look-ahead*) consiste à découvrir l'inconsistance d'une configuration partielle le plus tôt possible en utilisant les techniques de propagation de contraintes, et par la suite éviter d'explorer des branches ne contenant pas de solutions réalisables. Nous avons présenté par la suite une approche qui hybride le *look-back* et le *look-ahead*, qui vise à rendre le *backtracking* à la fois intelligent et capable de découvrir l'inconsistance d'une configuration partielle le plus tôt possible.

Ensuite nous avons donné une vue globale sur les métaheuristiques comme méthodes de résolution incomplètes des CSPs. Nous avons expliqué le fonctionnement de la recherche locale avec les notions de voisinage et nous avons donné quelques méthodes de recherche locale telles que le recuit simulé, les méthodes à voisinage variable et la Recherche Tabou. Les méthodes de recherche locale manipulent des configurations complètes qu'elles essayent de faire évoluer en faisant des mouvements successifs afin de chercher des configurations de meilleure qualité que la configuration courante. Des mécanismes sont ajoutés à ce genre de méthodes, comme la mémoire Tabou, l'intensification et la diversification, afin d'améliorer leur efficacité et leur performance. Ces méthodes ne gèrent pas la consistance des solutions générées contrairement aux méthodes de propagation de contraintes en général. Par contre elles sont assez réactives pour les problèmes de grande taille en pouvant alterner des phases d'intensification et d'exploration de la recherche.

L'hybridation des techniques issues de la programmation par contraintes et la recherche locale est ensuite présentée. L'idée consiste à hybrider principalement les avantages apportés par les méthodes manipulant des configurations partielles et la possibilité d'utiliser des techniques de propagation afin de réduire l'espace de recherche et les avantages apportés par la recherche locale qui s'illustre dans sa rapidité à trouver une bonne solution et dans sa capacité à diversifier la recherche. Nous avons détaillé dans ce cadre deux méthodes intéressantes et différentes, probablement les plus connues. La première appelée *CN-Tabu* fait une Recherche Tabou mais maintient un voisinage consistant par une réparation des conflits. L'aspect hybride se définit dans cette méthode par la combinaison de la Recherche Tabou

avec des configurations partielles. La deuxième méthode est le *Decision-Repair* qui se base principalement sur la propagation de contraintes et le *Nogood-Recording*. Dans cette méthode, La recherche locale se présente légèrement dans la phase de réparation d'une configuration partielle inconsistante. Ces deux méthodes sont très performantes, elles ont su en particulier tirer parti de mécanismes qui rendent compte de problèmes rencontrés lors de la recherche pour orienter les décisions futures.

La dernière section de ce chapitre présente notre première contribution dans le cadre de cette thèse. Cette contribution se résume dans la conception d'une nouvelle méthode hybride appelée *Tabu-NG* qui combine notamment des techniques utilisées par *Decision-Repair* et *CN-Tabu*. *Tabu-NG* combine principalement la propagation de contraintes, le *Nogood-Recording*, et la recherche locale pour étendre et pour réparer une configuration partielle. *Tabu-NG* est comme *Decision-Repair* et *CN-Tabu* une méthode générique et qui possède des paramètres à adapter en fonction du problème traité. *Tabu-NG* utilise des techniques de propagation qui doivent être adaptées en fonction des contraintes, une liste Tabou des *nogoods* dont la taille est à préciser et une mémoire Tabou dont la durée est à fixer.

Tabu-NG a été testée sur deux problèmes différents. Le problème d'affectation de fréquences qui correspond à une application réelle avec la présence de plusieurs types de contraintes, unaires, binaires et n-aires. Le problème de *k*-coloration qui est très utilisé comme problème académique dans la littérature. Le travail réalisé et les résultats obtenus sur ces deux problèmes sont présentés en détail dans les 2 chapitres suivants.

3. *Tabu-NG* et affectation de fréquences

Dans le but de synthétiser et formaliser les différentes notions et technologies relatives à l'Affectation de Fréquences Discrètes (AFD) pour les réseaux tactiques militaires, nous avons effectué un projet sur le développement de modules algorithmiques d'allocation de fréquences pour le compte du CELAR (*Centre ELectronique de l'ARmement de la DGA*). La modélisation, qui a été proposée par le CELAR, est caractérisée par un haut degré de spécification permettant de distinguer différentes sous-classes de problèmes avec des types de contraintes, de variables et d'objectifs variés.

Dans ce cadre nous avons traité des problèmes de type optimisation sous contraintes avec des contraintes dures et des contraintes molles ; ce chapitre porte sur les algorithmes que nous avons conçus et développés pour leur résolution. Nous commençons dans un premier temps par décrire la problématique d'affectation de fréquences discrètes dans les réseaux militaires proposée par le CELAR. En deuxième partie nous présentons les deux principaux modèles utilisés dans la littérature, coloriage de graphe et satisfaction de contraintes, et nous donnons une description de quelques méthodes proposées dans la littérature (les meilleures) pour résoudre ces problèmes.

Dans la troisième partie, nous présentons la méthode *Tabu-NG* appliquée au problème AFD classique (scénarios publics fournis par le CELAR et l'université de Delft), qui ne comporte que des contraintes binaires, ainsi qu'une analyse critique des résultats obtenus. En quatrième partie nous présentons la méthode *Tabu-NG* adaptée à la problématique d'affectation de fréquences avec polarisation et contraintes n-aires pour les réseaux militaires (scénarios privés fournis par le CELAR). Nous donnons aussi les résultats obtenus sur les scénarios privés ainsi qu'un ensemble d'analyse et de critiques issues des résultats obtenus et comparés avec les résultats fournis par le CELAR. Ce travail a fait l'objet de plusieurs publications dans des conférences internationales dont nous donnons la liste à la fin du manuscrit.

Sommaire

3. *Tabu-NG* et affectation de fréquences 66

3.1. Affectation de Fréquences Discrètes pour les réseaux tactiques militaires 67
3.2. Modèle de référence et méthodes de résolution 84
3.3. Méthode *Tabu-NG* pour le problème AFD classique 93
3.4. Méthode *Tabu-NG* pour AFDP avec contraintes n-aires 110
3.5. Synthèse 139

3.1. Affectation de Fréquences Discrètes pour les réseaux tactiques militaires

3.1.1. Contexte

Les interventions militaires de la France se placent dans un cadre de plus en plus large à la fois interarmées et interalliés. Le théâtre d'opérations dans ce contexte international voit cohabiter sur des zones géographiques parfois très limitées une quantité grandissante de systèmes radioélectriques civils et militaires (systèmes de communications mobiles, satellite, radars, systèmes de guerre électronique et de surveillance…). L'accroissement des besoins militaires se traduit par une augmentation de la ressource spectrale nécessaire. Or cette dernière n'est pas extensible et est de plus en plus convoitée par tous les opérateurs privés et publics avec des enjeux économiques et stratégiques très importants.

L'emploi des bandes de fréquences doit donc être optimisé afin de permettre le déploiement opérationnel des forces dans un environnement radioélectrique de plus en plus dense, tout en garantissant des performances aux systèmes en fonction des objectifs de chaque unité.

Dans ce contexte, il est nécessaire de doter les outils de planification et de gestion du spectre d'algorithmes d'allocation de fréquences performants dont le rôle est de déterminer la meilleure allocation possible en fonction de différents critères d'optimisation : utilisation spectrale minimale, qualité de communication optimale, etc.

Les travaux présentés dans ce chapitre s'inscrivent dans le cadre d'un projet qui portait sur la réalisation d'un module algorithmique pour l'allocation des fréquences de systèmes de transmission fonctionnant en mono porteuse et à fréquence fixe.

Avant de détailler les contraintes et les objectifs relatifs au problème d'affectation de fréquences susmentionné tels qu'ils ont été définis par le CELAR (Centre ELectronique de l'ARmement), nous présentons successivement la problématique générale d'allocation de fréquences discrète d'un point de vue physique puis mathématique.

3.1.2. Description physique du problème

Pour le problème d'allocation de fréquences que nous avons traité, deux systèmes sont visés :

- Les systèmes de faisceaux hertziens
- Les systèmes de réseaux de radios

Les liaisons hertziennes sont des liaisons point-à-point entre deux émetteurs/récepteurs généralement fixes. L'allocation de ressources pour une liaison doit être réalisée pour chaque sens de transmission appelé *trajet*. Une liaison hertzienne peut être constituée de un ou plusieurs trajets (voir Figure 16). La ressource est constituée d'une fréquence et, en fonction du type d'antenne utilisé, éventuellement d'une polarisation. La polarisation décrit une valeur binaire généralement notée *verticale* ou *horizontale*. Un réseau militaire à faisceaux hertziens peut être d'infrastructure ou tactique. Un réseau d'infrastructure n'évolue quasiment pas sur du court ou moyen terme tandis que la configuration d'un réseau tactique peut changer au cours d'une même journée. Cette spécificité rend les délais disponibles pour l'analyse et

l'optimisation des réseaux tactiques plus réduits comparés aux réseaux d'infrastructure. La qualité de l'optimisation étant dans ce cas partiellement sacrifiée pour une meilleure adaptabilité aux changements du système.

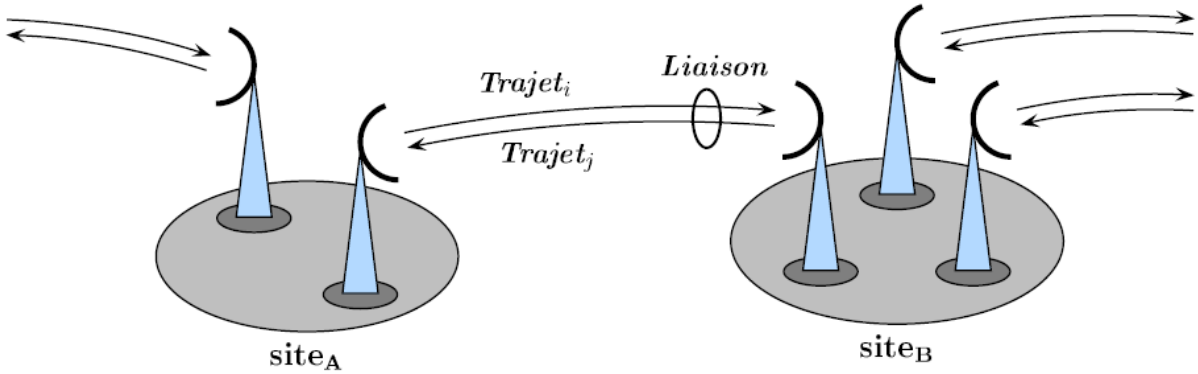


Figure 16 : Liaison hertzienne entre 2 sites

Un réseau de radio peut quant à lui être cellulaire ou non. Un réseau cellulaire est constitué d'un ensemble de stations de base qui assurent chacune la couverture radioélectrique d'une zone géographique sur laquelle évoluent des terminaux mobiles (systèmes GSM, UMTS…). Les liaisons radios sont exclusivement bilatérales entre les stations de base et les terminaux. Dans ce cas, l'allocation consiste à attribuer un ensemble de fréquences à chaque station de base afin qu'elle communique avec les terminaux de sa zone. Les contraintes que l'on retrouve pour ce type de réseau sont principalement des restrictions de ressources entre stations ou localement au niveau de chaque station. Pour les réseaux de radios non cellulaires, le réseau est constitué d'un ensemble de terminaux mobiles qui communiquent directement sans infrastructure (système PR4G). Il n'y a donc pas de notion de cellule ou de zone de couverture parfaitement délimitée. L'allocation de ressources consiste à attribuer une ou des fréquences à chacun des réseaux (réseau de sécurité, réseau de section, réseau de commandement…) en tenant compte des contraintes de compatibilité électromagnétique et éventuellement des priorités entre les réseaux. Les terminaux radios d'un réseau utilisent la fréquence du réseau pour communiquer entre eux dans les deux sens de la liaison.

L'objectif du projet est de fournir un module algorithmique qui attribue les valeurs des fréquences à utiliser afin que les communications soient de meilleure qualité possible. Pour cela, il est nécessaire de résoudre un certain nombre de contraintes. Ces contraintes sont présentes pour différentes raisons :

- Elles peuvent être imposées par les équipements.
- Elles peuvent être réglementaires.
- Elles peuvent être calculées à partir de critères de performance en tenant compte de l'environnement radioélectrique afin de limiter les interférences.

On peut distinguer deux grandes familles de contraintes : les contraintes unaires et les contraintes non-unaires.

Les contraintes unaires sont les contraintes qui ne concernent qu'un seul trajet. Elles imposent une plage de valeurs possibles pour un trajet en raison du matériel, de la réglementation… Elles s'expriment sous la forme d'appartenance ou d'interdiction. Par exemple, une fréquence affectée à un trajet devra appartenir à un intervalle particulier à cause du matériel utilisé ; ou à cause de la réglementation certaines fréquences ne peuvent pas être affectées à certaines liaisons car elles sont utilisées par une autre unité.

Les contraintes binaires et n-aires sont des contraintes de compatibilité électromagnétique qui concernent plusieurs trajets radio. Elles permettent de garantir que les fréquences affectées à chaque trajet n'interfèrent pas avec les fréquences d'autres trajets situés à proximité. Les contraintes non-unaires expriment les exigences qui doivent permettre de garantir un fonctionnement sans interférences. Elles peuvent avoir plusieurs formes en fonction de leur origine :

- Un *écart duplex* entre les fréquences d'émission et de réception d'une même liaison se traduira par une contrainte d'écart minimal entre le trajet montant et le trajet descendant.
- La cohabitation de plusieurs équipements sur un même site se traduira par des *contraintes co-sites* qui imposent des écarts minimaux entre les fréquences. Ces écarts sont calculés en fonction des caractéristiques des équipements et de leur implantation sur site. Ces contraintes sont en fonction des situations et des données disponibles :
  - Des *contraintes binaires d'écart minimal* en fréquences entre deux liaisons qui se perturbent.
  - Des *contraintes binaires d'harmoniques* : quand un trajet utilise une fréquence donnée, de l'énergie est aussi présente autour des fréquences multiples entières de celle-ci et peut créer des perturbations sur ces fréquences.
  - Des *contraintes n-aires de type sommation de perturbateurs*, où la fréquence d'un trajet est contrainte par les fréquences d'un ensemble de trajets susceptibles de le perturber.
  - Des *contraintes n-aires de type intermodulation* : quand plusieurs trajets en situation de proximité émettent chacun sur des fréquences données, de l'énergie peut apparaître sur des fréquences correspondant à des combinaisons linéaires entières de celles-ci, pouvant ainsi créer des perturbations.
- Pour des équipements plus éloignés, le calcul de *perturbation en champ lointain* génère aussi des contraintes sur les fréquences. Les types de perturbations peuvent être du rayonnement direct, hors bande, des harmoniques… On retrouve des contraintes comparables à celles définies ci-dessus dans le cas co-sites :
  - Des contraintes binaires d'écart minimal
  - Des contraintes binaires d'harmoniques
  - Des contraintes n-aires de type sommation de perturbateurs

3.1.3. Description mathématique du problème

3.1.3.1. Les variables et les ressources

La problématique d'affectation de fréquences que nous avons traitée consiste à allouer des ressources discrètes à des variables. Une variable peut correspondre à un trajet appartenant à une liaison. Les liaisons peuvent correspondre à des liaisons point-à-

point (allocation de faisceaux hertziens), des liaisons point-à-zone (réseaux cellulaires)… selon les systèmes et donc selon les instances de problèmes à traiter. Nous donnons tout d'abord quelques définitions de base.

Définition 24 : Variables cibles et hors cibles

Soit V l'ensemble des variables d'un problème donné. Une variable est un trajet auquel on doit affecter soit une *fréquence*, soit un couple (*fréquence, polarisation*). L'ensemble V est partitionné en deux sous-ensembles :

- T : ensemble des variables de la **cible**. Il contient toutes les variables qui doivent faire l'objet d'une affectation.
- $V \backslash T$: ensemble des variables **hors cible**. Il contient toutes les variables qui disposent d'une ressource initiale qui ne doit pas être modifiée lors du processus d'allocation mais qui interviennent dans le calcul des contraintes et des objectifs.

Définition 25 : Ressource

Il y a deux types de ressources selon la nature des variables considérées. Une ressource est une *fréquence* correspondant à la fréquence porteuse du signal transmis. Ou une ressource est un couple (*fréquence, polarisation*) dont les composants sont associés respectivement à la fréquence porteuse du signal transmis et à la polarisation de l'onde.

La polarisation est une ressource bivalente. La polarisation p_i attribuée à la variable $i \in V$ peut prendre les valeurs +1 (polarisation verticale) ou –1 (polarisation horizontale):

$$p_i \in \{+1, -1\}$$

Un problème est dit homogène si toutes les ressources sont soit uniquement de type *fréquence* soit uniquement sous la forme d'un couple (*fréquence, polarisation*) qui revient à (*fréquence*, -1) ou (*fréquence*, +1). Afin de simplifier, par la suite nous nous placerons par défaut dans le cas le plus général (*fréquence, polarisation*) sauf mention contraire.

Définition 26 : Domaine de ressources

Pour chaque trajet de la cible $i \in T \subset V$, on définit un ensemble de ressources qui sont susceptibles de lui être affectées ; cet ensemble est appelé domaine de ressources du trajet, il est noté D_i.

Définition 27 : Allocation de ressources

Soit A l'ensemble des allocations possibles pour un ensemble de variables V. Un élément a de A est une ressource r_i par variable i de l'ensemble V. Par abus de notation, on considèrera que :

$$a(V) = \{a(i) = r_i = (f_i, p_i), i \in V\}$$

On notera F_a l'ensemble des fréquences allouées à toutes les variables :

$$F_a = \{f : \exists i \in V, a(i) = (f, p_i)\}$$

Et F_a^X la restriction à un sous-ensemble *X* de variables :

$$F_a^X = \{f : \exists i \in X, a(i) = (f, p_i)\}$$

Allouer des ressources consiste pour l'opérateur à trouver un couple (*fréquence*, *polarisation*) pour chaque variable de la cible, pris dans le domaine de ressources associé à cette variable et satisfaisant un ensemble de contraintes.

3.1.3.2. Les contraintes

Une contrainte traduit en général la satisfaction d'un critère lié à la qualité des communications. Elle peut également traduire un impératif lié au matériel, à la réglementation ou simplement une exigence de l'allocateur.

On note *C* l'ensemble des contraintes qui doivent être respectées. On définira trois types principaux de contraintes :

- Les contraintes unaires, qui ne concernent qu'une seule variable ; par exemple imposer ou exclure une fréquence ou une polarisation pour une variable donnée.
- Les contraintes binaires qui concernent exactement deux variables différentes, ce sont typiquement les contraintes de compatibilité électromagnétique (CEM).
- Les contraintes n-aires qui concernent deux variables et plus. Les contraintes de sommation de perturbateurs ou d'intermodulation entrent dans ce périmètre.

Par la suite, on distinguera les contraintes unaires des autres contraintes qui seront qualifiées de non-unaires. On note *CU* l'ensemble des contraintes unaires et *CN* l'ensemble des contraintes non-unaires. On a $C = CU \bigcup CN$ l'ensemble de toutes les contraintes.

Une pondération peut être associée aux différentes contraintes. Cette pondération est représentée par un entier strictement positif :

$$(\omega_c)_{c \in C}$$

Par défaut, toutes les pondérations sont identiques et égales à l'unité.

3.1.3.3. Contraintes unaires

Comme leur nom l'indique, elles ne concernent qu'une seule variable. Différents types de contraintes unaires doivent être gérés :

- $r_i = r$ ou $r_i \neq r$, ressource imposée ou interdite pour une variable *i*.
- $f_i = f$ ou $f_i \neq f$, fréquence imposée ou interdite pour une variable *i*.
- $p_i = p$ ou $p_i \neq p$, polarisation imposée ou interdite pour une variable *i*.

- $f \le f_i \le g$, la fréquence allouée à la variable *i* doit appartenir à un certain intervalle, ou plus généralement à une réunion d'intervalles.
- r_i appartient à un ensemble défini de ressources exprimées sous la forme d'une énumération de valeurs discrètes ou d'ensembles plus généraux.

En toute généralité, ces contraintes unaires peuvent toujours s'exprimer sous la forme ensembliste, $r_i \in \Phi_i$, tel que la ressource allouée au trajet *i* doit appartenir à un ensemble Φ_i. Il est à noter que Φ_i ne fait que traduire la contrainte unaire ; la ressource allouée à la variable *i* devra de toute façon être dans le domaine de ressource D_i associé à *i*. Ainsi on n'a pas nécessairement $\Phi_i \subset D_i$ mais on devra toujours avoir $r_i \in \Phi_i \cap D_i$.

3.1.3.4. Contraintes binaires

Les contraintes binaires prennent en compte exactement deux variables. Elles correspondent en règle générale à des contraintes de compatibilité électromagnétique. Plusieurs types de contraintes binaires doivent être gérés :

- Contrainte d'égalité ou d'inégalité de fréquences entre variables : $f_i = f_j$ ou $f_i \neq f_j$
- Contrainte d'égalité ou d'inégalité de polarisation : $p_i = p_j$ ou $p_i \neq p_j$
- Contrainte d'égalité ou d'inégalité de ressources : $r_i = r_j$ ou $r_i \neq r_j$
- Contrainte d'égalité ou d'inégalité d'un écart de fréquences : $\left|f_i - f_j\right| = \varepsilon_{ij}$ ou $\left|f_i - f_j\right| \neq \omega_{ij}$
- Contrainte d'écart en fréquences minimal : $\left|f_i - f_j\right| \ge \varepsilon_{ij}$ et plus généralement, on impose à un écart en fréquences d'appartenir à un ensemble explicite (valeurs discrètes, intervalle, union d'intervalles) : $\left|f_i - f_j\right| \in \Psi_{ij}$

  Exemple : $\left|f_i - f_j\right| \in \left[\varepsilon_{ij}^1, \varepsilon_{ij}^2\right] \cup \left[\varepsilon_{ij}^3, \varepsilon_{ij}^4\right]$ où $\varepsilon_{ij}^k \in \ddot{\Re}^+ = [0,+\infty]$
- Contrainte d'écart en pourcentage : $\left|f_i - f_j\right| \ge \pi_{ij} \times \max\left(f_i, f_j\right)$
- Contrainte d'harmonique : $\left|f_i - nf_j\right| \ge h_{ij}$ et $\left|f_j - nf_i\right| \ge h_{ij}$
- Contraintes d'écart en fréquences à deux états : $\left|f_i - f_j\right| \ge \begin{cases} \gamma_{ij} & \text{si } p_i = p_j \\ \delta_{ij} & \text{si } p_i \neq p_j \end{cases}$

  Cette dernière traduit le fait que l'écart en fréquences nécessaire entre deux trajets qui se perturbent dépend des polarisations qui leur sont respectivement affectées. En règle générale, on aura une contrainte en polarisation croisée plus faible qu'en polarisation identique : $\gamma_{ij} \ge \delta_{ij}$.

  Plus généralement, en fonction de l'égalité ou non des polarisations, on imposera à l'écart en fréquence d'appartenir à des ensembles différents :

$$\left|f_i - f_j\right| \in \begin{cases} \Gamma_{ij} & \text{si } p_i = p_j \\ \Delta_{ij} & \text{si } p_i \neq p_j \end{cases}$$

- Contrainte d'écart minimal en fréquences à quatre états :

$$\left|f_i - f_j\right| \geq \begin{cases} \gamma_{ij}^{+1} & \text{si } p_i = p_j = +1 \\ \gamma_{ij}^{-1} & \text{si } p_i = p_j = -1 \\ \delta_{ij}^{+1} & \text{si } p_j \neq p_i = +1 \\ \delta_{ij}^{-1} & \text{si } p_j \neq p_i = -1 \end{cases}.$$

Cette dernière traduit le fait que l'écart en fréquences nécessaire entre deux trajets qui se perturbent dépend des polarisations qui leur sont respectivement affectées.

Plus généralement, en fonction de la valeur de la polarisation et de leur égalité ou non, on imposera à l'écart en fréquence d'appartenir à des ensembles différents :

$$\left|f_i - f_j\right| \in \begin{cases} \Gamma_{ij}^{+1} & \text{si } p_i = p_j = +1 \\ \Gamma_{ij}^{-1} & \text{si } p_i = p_j = -1 \\ \Delta_{ij}^{+1} & \text{si } p_j \neq p_i = +1 \\ \Delta_{ij}^{-1} & \text{si } p_j \neq p_i = -1 \end{cases}$$

3.1.3.5. Contraintes n-aires

Les contraintes n-aires concernent au moins deux variables, deux types de contraintes n-aires sont à considérer :

- Contrainte n-aire de type *sommation de perturbateurs* : cette contrainte permet de gérer la prise en compte simultanée des effets de *n* perturbateurs sur un même récepteur ρ : $\sum_{i=1}^{n} \beta_{i\rho} \times T_{i\rho}\left(\left|f_i - f_\rho\right|\right) \leq B_\rho$

  — Les valeurs $B_{i\rho}$ sont des coefficients linéaires qui pondèrent la contribution du perturbateur *i* en prenant en compte la distance entre l'émetteur perturbateur et le récepteur, les gains en émission et en réception.

  — La fonction $T_{i\rho}()$ est une fonction tabulée calculée pour chaque valeur possible de l'écart en fréquences $f_i - f_\rho$. Elle correspond à la convolution entre la densité de puissance de l'émetteur perturbateur et la fonction de sélectivité du récepteur. La fonction $T_{i\rho}()$ est paire et décroissante dans le domaine positif.

— B_ρ est une valeur propre au récepteur ρ ; elle intègre toutes les caractéristiques de la liaison utile, la puissance de bruit du récepteur et la valeur retenue du critère de qualité.

NB : Comme pour les contraintes binaires d'écart minimal, la valeur de $B_{i\rho}$ dépend des polarisations attribuées aux trajets i et ρ. On a donc des contraintes de sommation de perturbateurs à deux et quatre états avec autant de valeurs de $B_{i\rho}$.

- Contrainte n-aire de type *intermodulation* : cette contrainte permet de décrire des effets d'intermodulation sur un récepteur ρ à la fréquence f_ρ générée par deux perturbateurs aux fréquences f_i et f_j :

$$\left| f_\rho - \left(\pm\alpha_i f_i \pm \alpha_j f_j \right) \right| \geq \zeta$$

NB : Plusieurs contraintes d'intermodulation faisant intervenir les mêmes variables peuvent apparaître dans un même problème.

3.1.3.6. Relâchement des contraintes

On définit une solution réalisable comme l'allocation d'une ressource à chaque trajet en satisfaisant l'ensemble des contraintes imposées. On définira par la suite plusieurs critères d'optimisation des solutions réalisables. On appelle $\Omega \subset A$ l'ensemble des solutions réalisables.

Les problèmes d'affectation de fréquences n'ont pas toujours une solution réalisable. Des domaines de ressources trop restreints ou des contraintes trop fortes peuvent conduire à une telle situation. L'allocateur ne pouvant se satisfaire d'un tel constat peut vouloir rechercher des solutions *dégradées obtenues en relâchant des contraintes* ; on les appellera *solutions de compromis*.

Pour cela, on définit deux types de contraintes :

- Les contraintes fortes ou impératives : les contraintes à satisfaire obligatoirement.
- Les contraintes non impératives sur lesquelles on autorisera un relâchement progressif et contrôlé. Pour cela on introduit la notion de *niveau de repli*. Chaque contrainte de ce type est définie par différents niveaux de repli. Le niveau de repli 0 correspondra par convention au problème initialement posé (niveau nominal). Le passage d'un niveau de repli au suivant se traduit par un relâchement de tout ou partie des contraintes non-impératives. Le niveau maximal de relâchement est le niveau 10. Les contraintes de niveau 10, correspondant au niveau de relâchement maximal peuvent être considérées comme des contraintes impératives pour le problème.

3.1.3.7. Relâchement des contraintes unaires

Toute contrainte unaire sur une variable i peut s'exprimer sous la forme $(f_i, p_i) \in \Phi_i$ avec Φ_i la traduction ensembliste de la contrainte unaire. La notion de relâchement d'une

contrainte unaire s'exprime donc sous la forme d'une suite de sous-ensembles $\Phi_i^{(l)}$ définis comme suit :

$$(f_i, p_i) \in \Phi_i^{(0)} \subset \Phi_i^{(1)} \subset \cdots \subset \Phi_i^{(l)} \subset \cdots \subset \Phi_i^{(10)}$$

Si $\Phi^{(l)}$ n'est pas fourni, on considèrera qu'il est implicitement égal à $\Phi^{(l+1)}$ s'il existe et ainsi de suite récursivement jusqu'à $\Phi^{(10)}$.

Exemple :

On considère un relâchement d'une contrainte unaire associée à la variable *i*.

- Au niveau nominal, on souhaite que la ressource soit exactement égale au couple (*a*, +1), avec (*a*, +1) appartenant à D_i, soit $f_i=a$ et $p_i=+1$, ce qui se traduira par : $\Phi_i^{(0)} = \{(a,+1)\}$
- Au niveau 1, on autorise la fréquence à varier entre *b* et *c* mais sans modifier la polarisation, on aura alors :

$$\Phi_i^{(1)} = \{(f_i, p_i) \in D_i / b \leq f_i \leq c \text{ et } p_i = +1\}$$

- Au niveau 2, on autorise la polarisation à varier, la fréquence restant dans le même intervalle que précédemment : $\Phi_i^{(2)} = \{(f_i, p_i) \in D_i / b \leq f_i \leq c\}$
- Au niveau 10, on autorise tous les couples du domaine de ressource, soit $\Phi_i^{(10)} = D_i$

Dans cet exemple, il n'y a pas de relâchement défini du niveau 3 au niveau 9 inclus, on considèrera alors que pour ces niveaux le relâchement à prendre en compte est celui du prochain niveau supérieur, ici le 10. Ainsi on aura : $\Phi_i^{(3)} = \Phi_i^{(4)} = \cdots = \Phi_i^{(9)} = \Phi_i^{(10)}$.

3.1.3.8. Relâchement des contraintes binaires

Toute contrainte binaire entre deux variables *i* et *j* peut s'exprimer sous la forme $|f_i - f_j| \in \Psi_{ij}$ avec Ψ_{ij} la traduction ensembliste de la contrainte binaire. La notion de relâchement d'une contrainte binaire s'exprime donc sous la forme d'une suite de sous-ensembles $\Psi_{ij}^{(l)}$ définis comme suit :

$$|f_i - f_j| \in \Psi_{ij}^{(0)} \subset \Psi_{ij}^{(1)} \subset \cdots \subset \Psi_{ij}^{(l)} \subset \cdots \subset \Psi_{ij}^{(10)}$$

Si $\Psi^{(k)}$ n'est pas fourni, on considèrera qu'il est implicitement égal à $\Psi^{(k+1)}$ s'il existe et ainsi de suite récursivement comme pour les contraintes unaires.

Exemple avec une contrainte d'écart minimal :

Pour les contraintes d'écart minimum à deux états en fréquences, le relâchement se traduit par un relâchement de tout ou partie des écarts fréquentiels définissant chaque contrainte ; ainsi pour une contrainte entre les variables *i* et *j*, on aura :

$$\left|f_i - f_j\right| \geq \begin{cases} \gamma_{ij}^{(0)} \geq \gamma_{ij}^{(1)} \geq \ldots \geq \gamma_{ij}^{(k)} \geq \ldots \geq \gamma_{ij}^{(10)} & \text{si } p_i = p_j \\ \delta_{ij}^{(0)} \geq \delta_{ij}^{(1)} \geq \ldots \geq \delta_{ij}^{(k)} \geq \ldots \geq \delta_{ij}^{(10)} & \text{si } p_i \neq p_j \end{cases}$$

Avec $\gamma_{ij}^{(k)}$ et $\delta_{ij}^{(k)}$, les écarts à respecter entre les variables *i* et *j* pour le niveau de repli *k*.

Ou de manière équivalente : $\left|f_i - f_j\right| \geq \frac{\left|p_i + p_j\right|}{2}\gamma_{ij}^{(k)} + \frac{\left|p_i - p_j\right|}{2}\delta_{ij}^{(k)}, k = 0,1,\ldots,10$

3.1.3.9. Relâchement des contraintes n-aires

Le principe de relâchement des contraintes n-aires d'intermodulation est comparable à celui défini pour les contraintes unaires et binaires. On définit une suite décroissante de valeurs $\zeta^{(k)}$: $\left(\zeta^{(k)}\right)_{k=0..10}, \zeta^{(k)} \geq \zeta^{(k+1)}$.

Pour le relâchement des contraintes de sommation de perturbateurs, on définit une suite de contraintes de type sommation de perturbateurs, chacune correspondant à un niveau de replis, avec le seuil B_ρ prenant des valeurs croissantes.

3.1.4. Les objectifs

La problématique d'affectation de fréquences est définie suivant différent modes d'optimisation relatifs aux objectifs de qualité recherchés. Ces critères de qualité traduisent des contraintes à satisfaire (éventuellement relâchées) et des objectifs d'optimisation spectrale (utilisation des fréquences disponibles). Ce qui donne 9 modes de recherche que nous détaillons ci-dessous après quelques définitions se rapportant aux notions de réalisabilité. Chacun des modes doit être traité séparément, il s'agit d'autant de problèmes différents, il ne s'agit pas d'optimisation multi objectif.

Définition 28 : Solution *l*-unaire-réalisable

On définit une solution ***l*-unaire-réalisable** comme une allocation d'une ressource à chaque variable satisfaisant l'ensemble des contraintes fortes, des contraintes non unaires de niveau 10 et de toutes les contraintes unaires de niveaux *l* et supérieurs. Soit $U_l \subset A$ l'ensemble des solutions *l*-unaires-réalisables. On dira alors que le problème est ***l*-unaire-réalisable** si l'ensemble U_l n'est pas vide.

Définition 29 : Solution *k*-non-unaire-réalisable

On définit une solution ***k*-non-unaire-réalisable** comme une allocation d'une ressource à chaque variable qui satisfait l'ensemble des contraintes fortes, des contraintes unaires de niveau 10 et de toutes les contraintes non unaires de niveaux *k* et supérieurs. Soit $N_k \subset A$ l'ensemble des solutions *k*-non-unaires-réalisables. On dira alors que le problème est ***k*-non-unaire-réalisable** si N_k n'est pas vide.

Définition 30 : Solution (*l*,*k*)-réalisable

On définit une solution **(*l*,*k*)-réalisable** comme une allocation d'une ressource à chaque variable qui satisfait l'ensemble des contraintes fortes, toutes les contraintes unaires de niveau l et supérieurs, et toutes les contraintes non-unaires de niveaux k et supérieurs. Soit $W_{l,k} \subset A$ l'ensemble des solutions (l,k)-réalisables. On dira alors que le problème est **(*l*,*k*)-réalisable** si $W_{l,k}$ n'est pas vide. Par ailleurs, il y a des contraintes impératives à respecter par toutes les solutions indépendamment des niveaux (l,k), ces contraintes sont de différents types.

Neuf modes de recherche sont à prendre en compte :

- Un mode de recherche AFD de n solutions réalisables (n fixé), i.e. satisfaisant l'ensemble de toutes les contraintes pour un niveau fixé de relâchement unaire l et non-unaire k.
- Cinq modes de recherche AFD d'une solution réalisable optimisant un critère d'utilisation du spectre pour un niveau fixé de relâchement unaire l et non-unaire k.
- Trois modes de recherche AFD d'une solution de compromis conduisant à un relâchement de contraintes, en faisant varier le niveau de relâchement unaire l, ou non-unaire k ou les deux niveaux de relâchement.

Pour l'ensemble des définitions à venir, nous rappelons que Ω est l'ensemble des solutions réalisables et T est l'ensemble des variables cibles .

3.1.4.1. Pb1 : Recherche de solutions réalisables

L'objectif de l'algorithme est de trouver des solutions réalisables. Le nombre de solutions demandées ***n*** ainsi que les valeurs de niveau de relâchement unaire ***l*** et non-unaire ***k*** sont fournis en entrée. Le module algorithmique doit rechercher ***n*** solutions *(l,k)*-réalisables ou prouver qu'il n'en existe pas du tout ou qu'il n'en existe que $\boldsymbol{m}<\boldsymbol{n}$.

3.1.4.2. Pb2 : Optimisation de solutions réalisables avec MinFreq

Pour un niveau de relâchement unaire ***l*** et un niveau de relâchement non-unaire ***k***, l'objectif de l'algorithme de recherche est de trouver la solution *(l,k)*-réalisable qui minimise le nombre de fréquences utilisées pour les variables de la cible sans prendre les polarisations assignées en considération.

$$MinFreq = \min_{a \in \Omega} \left(\left| F_a^T \right| \right)$$

3.1.4.3. Pb3 : Optimisation de solutions réalisables avec MinSpec

Pour un niveau de relâchement unaire ***l*** et un niveau de relâchement non-unaire ***k***, ce mode vise à trouver la solution *(l,k)*-réalisable qui minimise la largeur totale du spectre utilisé par les variables de la cible.

$$MinSpec = \min_{a \in \Omega} \left(\max\left(F_a^T\right) - \min\left(F_a^T\right) \right)$$

3.1.4.4. Pb4 : Optimisation de solutions réalisables avec MinFreqD

Pour un niveau de relâchement unaire ***l*** et un niveau de relâchement non-unaire ***k***, ce mode vise à trouver la solution *(l,k)*-réalisable qui minimise la fréquence supérieure du spectre utilisé pour les variables de la cible. En théorie, la solution sera alors cadrée dans la partie inférieure du spectre disponible.

$$MinFreqD = \min_{a \in \Omega} \left(\max \left(F_a^T \right) \right)$$

3.1.4.5. Pb5 : Optimisation de solutions réalisables avec MaxFreqG

Pour un niveau de relâchement unaire ***l*** et un niveau de relâchement non-unaire ***k***, ce mode vise à trouver la solution *(l,k)*-réalisable qui maximise la fréquence inférieure du spectre utilisé pour les variables de la cible. En théorie, la solution sera alors cadrée dans la partie supérieure du spectre disponible.

$$MaxFreqG = \max_{a \in \Omega} \left(\min \left(F_a^T \right) \right)$$

3.1.4.6. Pb6 : Optimisation de solutions réalisables avec MinEcart

Pour un niveau de relâchement unaire ***l*** et un niveau de relâchement non-unaire ***k***, ce mode vise à trouver la solution *(l,k)*-réalisable qui minimise l'écart entre toute fréquence allouée aux variables de la cible, et une, ou deux, fréquences choisies par l'allocateur dans le spectre disponible. En théorie, les fréquences utilisées seront regroupées autour d'une ou deux fréquences particulières notées ci-dessous f_1 et f_2.

$$MinEcart = \min_{a \in \Omega} \left(\max_{x \in T} \left(\min \left(\left| F_a^x - f_1 \right|, \left| F_a^x - f_2 \right| \right) \right) \right)$$

3.1.4.7. Pb7 : Optimisation de solutions de compromis avec MinCard

Soit *l* un niveau de repli unaire donné. Sur l'ensemble des solutions *l*-unaires-réalisables, ce mode vise à minimiser la somme pondérée des contraintes non-unaires non satisfaites en gérant les niveaux de repli sur les contraintes binaires. On optimisera successivement :

- Rechercher *k**, le plus petit niveau de relâchement tel qu'il existe une solution (l,*k**)-réalisable.
- Minimiser la somme pondérée des contraintes non-unaires non satisfaites au niveau *k**-1, $\sum_{c \in \overline{CN}^{(k^*-1)}} (\omega_c)$.
- Minimiser la somme pondérée des contraintes non-unaires non satisfaites aux niveaux strictement inférieurs à (*k**-1), $\sum_{j=0..(k^*-1)} \sum_{c \in \overline{CN}^{(j)}} (\omega_c)$.

3.1.4.8. Pb8 : Optimisation de solutions de compromis avec MinMob

Soit *k* un niveau de repli non-unaire donné. Sur l'ensemble des solutions *k*-non-unaires-réalisables, ce mode vise à minimiser le nombre de contraintes unaires non satisfaites en gérant les niveaux de repli sur les contraintes unaires. On optimisera successivement :

- Rechercher *l** le plus petit niveau de relâchement tel qu'il existe une solution (*l**,*k*)-réalisable.
- Minimiser la somme pondérée des contraintes unaires non satisfaites au niveau *l**-1, $\sum_{c\in\overline{CU}^{(l^*-1)}}(\omega_c)$.
- Minimiser la somme pondérée des contraintes unaires non satisfaites aux niveaux strictement inférieurs à (*l**-1), $\sum_{j=0..(l^*-1)}\sum_{c\in\overline{CU}^{j}}(\omega_c)$.

3.1.4.9. Pb9 : Optimisation de solutions de compromis avec MinMax

Soit *l* un niveau de repli pour les contraintes unaires et soit *k* un niveau de repli pour les contraintes non-unaires. Dans l'ensemble des solutions (*l*-10)-réalisables, on recherchera à minimiser la non-satisfaction uniforme des contraintes non-unaires de niveau supérieur ou égal à *k*.

$$\min_{a\in W_{l,10}}\left(\max_{c\in\overline{CN}^{(k)}(a)}\xi_c\right)$$

ξ_c est l'évaluation de la non-satisfaction de la contrainte *c*.

Le *MinMax* est pris par rapport à un ensemble homogène de contraintes (contraintes d'écart minimal, de sommation de perturbateurs…). Pour réaliser une optimisation de type *MinMax* sur plusieurs sous-ensembles homogènes de contraintes, le *MinMax* global s'exprimera comme une somme pondérée des *MinMax* élémentaires.

3.1.5. Définition des classes d'optimisation

Nous avons réalisé une classification des différents modes d'optimisation sur la base du résultat de la recherche de la réalisabilité et de l'objectif concerné (cf. Figure 17). La position de chaque problème est déterminée par la réponse ou les réponses (cas itératif de la recherche du niveau de relâchement global) à la question de réalisabilité. Sur la base de cette réponse et de la nature du mode d'optimisation un arrêt de la recherche, une optimisation spectrale ou une recherche de compromis est entreprise.

En termes de méthodologie de travail, ceci permet de perfectionner une même approche, éventuellement avec plusieurs variantes, utilisable pour les différents cas à traiter.

3.1.5.1. Classe A

La problématique consiste uniquement à une recherche de réalisabilité. La recherche s'arrête dès qu'une solution complète consistante est trouvée (cas du mode d'optimisation 1) ou suite à la preuve de l'inconsistance du problème (cas des modes d'optimisation 1 à 6 et 7 à 9) car alors il n'y a plus d'optimisation spectrale.

3.1.5.2. Classe B

L'algorithme de recherche poursuit l'optimisation après l'obtention de la solution réalisable afin d'améliorer l'utilisation spectrale selon l'un des cinq modes 2-6. Cette étape fera suite à un cycle de recherche de réalisabilité suivant la méthode de classe A.

3.1.5.3. Classe C

L'algorithme procède à l'optimisation de la satisfaction des contraintes via une recherche de solutions de compromis (modes 7 à 9). Cette étape fera suite à un cycle de recherche de réalisabilité suivant la méthode de classe A, permettant à la méthode de fixer le niveau de relâchement général sur les contraintes.

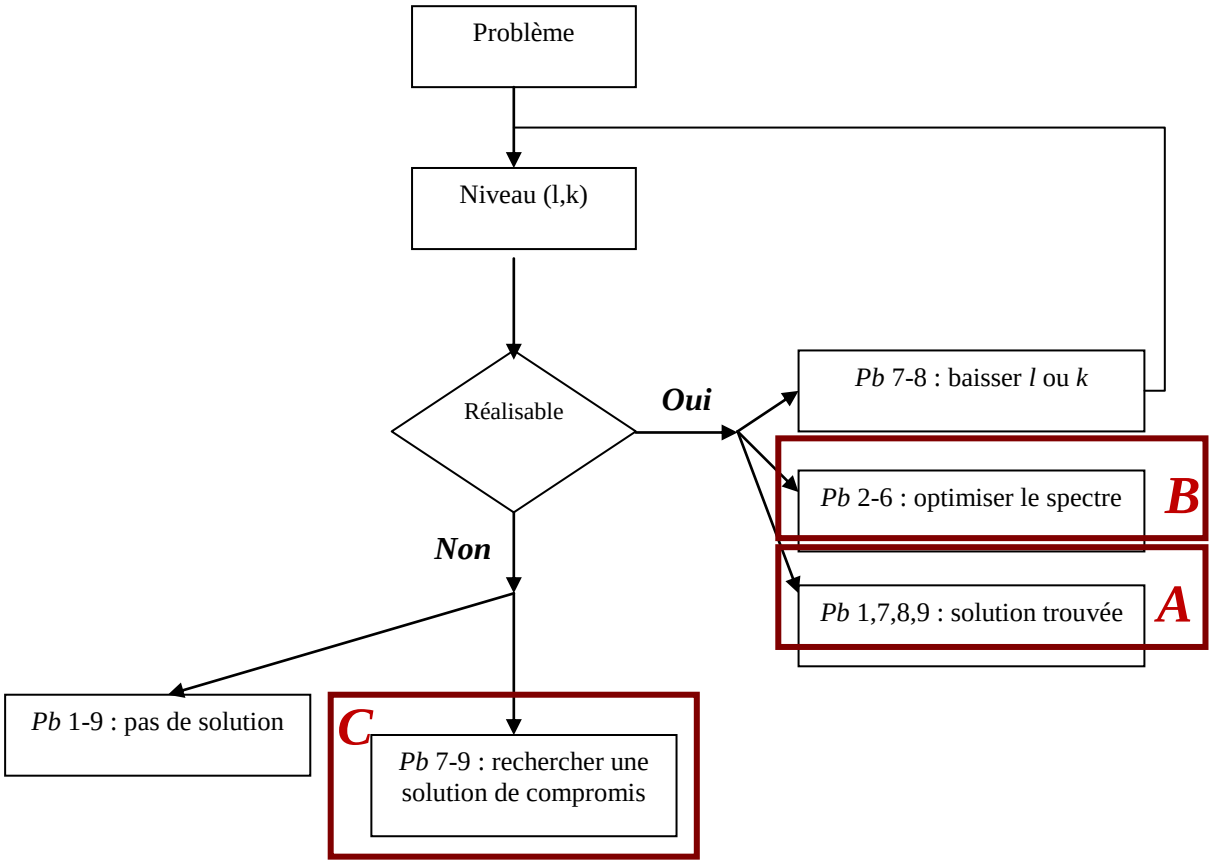


Figure 17 : Identification des classes de problèmes correspondant aux 9 modes d'optimisation

3.1.6. Les instances

Deux groupes de jeux de tests ont été utilisés dans notre travail, l'un privé fourni par le CELAR dans le cadre du projet, l'autre public fourni par le CELAR et l'université de Delft et publié dans le cadre du projet européen CALMA [119] [120]. Les détails de ces instances sont présentés dans cette partie. En ce qui concerne les jeux privés, nous ne donnerons que certaines caractéristiques correspondant aux propriétés des instances publiques.

Les instances publiques sont de taille moyenne et ne comportent que des contraintes binaires. Les instances privées sont de très grande taille, elles correspondent à la définition complète du problème et comprennent différents types de contraintes et objectifs à optimiser. Les instances privées sont plus difficiles à résoudre que les instances publiques et ceci a été confirmé par le CELAR qui possède un ensemble d'outils d'affectation utilisant différentes méthodes de résolution.

3.1.6.1. Les instances publiques

Pour améliorer notre analyse des performances de la solution algorithmique proposée dans le cadre du projet avec le CELAR, il est nécessaire de se baser sur des instances publiques connues dans la littérature et proches du problème défini dans le projet. Les instances du projet européen CALMA (*Combinatory ALgorithms for Military Applications*) [119] [120] répondent exactement à ces besoins. Plusieurs équipes de chercheurs ont travaillé sur les mêmes instances de problème : 11 instances CELAR fournies par le CELAR et 14 instances GRAPH fournies par l'Université de Technologie de Delft [121]. Les instances CELAR sont toutes déclinées à partir d'un seul et même problème contenant 916 liaisons et 5744 contraintes **binaires,** de type contraintes d'écart en fréquences minimal ou contraintes d'égalité (cf. section 3.1.3.4), réparties dans 11 composantes connexes. Elles avaient été conçues, à l'origine, pour évaluer différents outils de programmation par contraintes. Les instances GRAPH (*Generating Radio link frequency Assignment Problems Heuristically*) ont été générées aléatoirement par van Benthem [121] et possèdent les mêmes caractéristiques que les instances CELAR.

Chacune des variables des problèmes CELAR et GRAPH prend ses valeurs de fréquences dans l'un des sept domaines possibles définis dans un même fichier commun à tous les problèmes. Selon le problème, une variable est définie par un numéro de domaine de 1 à 7. Le plus grand domaine contient 44 valeurs, le plus petit 6.

Dans un premier temps, nous nous sommes concentrés sur la réalisabilité. Par conséquent, nous ne présentons ici que les instances qui possèdent une solution réalisable. Le reste des instances a pour objectif la minimisation du nombre de contraintes violées.

Le Tableau 2 détaille les caractéristiques de chaque instance :

- Le nombre de variables ou *trajets*
- Le nombre total de valeurs ou *fréquences*

- Le nombre de contraintes binaires
- Le nombre de trajets pré-affectés (interdit de les changer)
- L'objectif visé : *MinFreq* pour minimiser le nombre de fréquences et *MinSpec* pour minimiser la largeur du spectre (lignes grises)
- Les meilleurs résultats connus selon les deux objectifs (incertitude sur le graph08)

Les valeurs sont suivies d'une étoile lorsqu'elles sont prouvées optimales. Pour plus de détails sur le projet CALMA, les instances, les méthodes et leurs résultats, le lecteur pourra se référer au lien web suivant : http://fap.zib.de/problems/CALMA/.

Problème	**nb var**	**nb val**	**nb ctr**	**nb aff**	**objectif**	***MinFreq***	***MinSpec***
CELAR01	916	36200	5548	0	*MinFreq*	16∗	680
CELAR02	200	80004	1235	0	*MinFreq*	14∗	394
CELAR03	400	15892	2760	0	*MinFreq*	14∗	666
CELAR04	680	26856	3967	280	*MinFreq*	46∗	-
CELAR05	400	15768	2598	0	*MinSpec*	-	792∗
CELAR11	680	26856	4103	0	*MinFreq*	22∗	-
graph01	200	6920	1134	0	*MinFreq*	18∗	408
graph02	400	14624	2245	0	*MinFreq*	14∗	394
graph03	200	7820	1134	0	*MinSpec*	-	380
graph04	400	15592	2244	0	*MinSpec*	-	394∗
graph08	680	25628	3757	0	*MinFreq*	16 ou 18	652
graph09	916	36092	5246	0	*MinFreq*	18∗	666
graph10	680	26980	3907	0	*MinSpec*	-	394∗
graph14	916	36716	4638	0	*MinFreq*	8	352

Tableau 2 : Caractéristiques et objectifs des instances CELAR et GRAPH

3.1.6.2. Les instances privées

Les 30 instances privées seront notées *SCENx* avec *x* une valeur entière comprise entre 1 et 30. Le Tableau 3 détaille les caractéristiques de chaque instance :

- Le nombre de variables ou *trajets*
- Le nombre de valeurs, ou *canaux,* correspondant à la taille du domaine global représentant l'union de tous les domaines des variables. Il n'y a pas un seul domaine par problème, il s'agit plutôt de plusieurs ensembles de spectre de fréquences. Un domaine d'une variable pourra être l'union de plusieurs ensembles. Nous rappelons que chaque fréquence peut-être polarisée donnant lieu à deux valeurs possibles *verticale* et *horizontale*.
- Le nombre de contrainte unaire
- Le nombre de contrainte binaire avec tous les types de contraintes présentées dans la section 3.1.3.4

- Le nombre de contraintes n-aires qui sont les contraintes d'intermodulation et de sommation de perturbateurs, cf. section 3.1.3.5
- Le nombre total de contraintes

Problème	nb var	nb val	Ctr unaires	Ctr binaires	Ctr n-aires	nb ctr
SCEN1	50	92	0	444	0	444
SCEN2	41	196	0	555	0	555
SCEN3	40	100	0	1252	139	1391
SCEN4	16	662	47	56	0	103
SCEN5	50	514	32	1229	0	1261
SCEN6	50	75	0	6562	270	6832
SCEN7	50	49	0	483	0	483
SCEN8	50	49	0	507	0	507
SCEN9	48	55	48	1238	155	1441
SCEN10	50	49	0	536	0	536
SCEN11	770	125	0	33909	3060	36969
SCEN12	702	72	0	39123	0	39123
SCEN13	182	700	0	24756	1036	25792
SCEN14	300	506	192	4121	0	4313
SCEN15	1500	155	0	143186	0	143186
SCEN16	115	1921	0	1814	90	1904
SCEN17	568	71	0	46554	0	46554
SCEN18	2000	95	0	146184	0	146184
SCEN19	154	100	0	16298	1414	17712
SCEN20	221	1720	0	6108	598	6706
SCEN21	1088	500	0	273060	14934	287994
SCEN22	1466	71	0	93516	0	93516
SCEN23	1930	125	0	74419	4410	78829
SCEN24	1314	91	0	76875	0	76875
SCEN25	364	841	0	9834	198	10032
SCEN26	2166	460	0	74070	7820	81890
SCEN27	144	801	0	17837	410	18247
SCEN28	894	91	2936	42003	0	44939
SCEN29	2454	400	0	66629	8474	75103
SCEN30	3000	91	0	182739	0	182739

Tableau 3 : Caractéristiques des instances privées

Les scénarios privés sont très différents les uns des autres, on peut néanmoins les ranger en 2 classes. Les scénarios 1 à 10 sont des petits scénarios avec 50 variables et 7000 contraintes au plus. Les scénarios 11 à 30 sont des scénarios de taille très importante avec 115 à 3000 variables et jusqu'à 300 000 contraintes.

14 scénarios, parmi les 30, contiennent des contraintes n-aires mettant en jeu plus de 2 variables (lignes grises). Une contrainte n-aire de type sommation de perturbateurs peut porter sur un nombre important de variables ; par exemple pour le SCEN9, nous avons des contraintes qui portent sur 15 variables, soit 1/3 du nombre total de variables du problème.

3.1.7. Complexité

Un problème d'affectation de fréquences peut être représenté par un graphe de contraintes, également appelé graphe d'interférences. Les sommets correspondent aux trajets du réseau. Un arc reliant deux trajets désigne une contrainte entre les deux variables. Cependant cette représentation est incomplète par rapport aux problèmes que nous traités car elle n'exprime pas les contraintes unaires et n-aires ainsi que les données relatives à chaque contrainte comme l'écart demandé, le poids ou la priorité, le niveau de relâchement...

Prenons un cas particulier pour étudier la complexité des problèmes évoqués ci-dessus. Supposons qu'il n'y ait que des contraintes d'écart minimum, que les écarts demandés égalent 1, et que les domaines fréquentiels des trajets soient tous identiques et égaux à l'ensemble des entiers entre 1 et k, la question de savoir s'il existe une solution qui satisfasse toutes les contraintes revient à rechercher une *k-coloration valide* d'un graphe G, ce qui est un problème NP-complet pour $k > 2$. Les problèmes que nous avons à traiter sont donc particulièrement difficiles.

3.2. Modèle de référence et méthodes de résolution

Dans la littérature, il existe des problèmes théoriques qui s'approchent du problème d'affectation de fréquences proposé par le CELAR : les problèmes de coloration de graphe et les problèmes de satisfaction de contraintes. Dans cette partie, nous décrivons ces deux modèles ainsi que leurs similitudes avec le problème que nous traitons.

3.2.1. AFD et coloration de graphe

Dans les problèmes de référence en affectation de fréquences, les contraintes électromagnétiques entre deux liaisons radios sont exprimées sous forme d'écart à respecter entre les fréquences allouées aux deux liaisons concernées. Le problème ainsi produit est exclusivement constitué de contraintes binaires mettant en jeu uniquement deux variables à chaque fois et exprimant un brouillage simple avec un unique brouilleur c'est-à-dire d'une liaison par une autre. Par conséquent, les contraintes binaires dans le problème d'allocation de fréquences sont souvent représentées par un graphe non orienté dont les sommets désignent des liaisons hertziennes ou des cellules et les arcs reliant les sommets représentent les contraintes binaires. Cette représentation donna lieu à une multitude d'approches qui ramène le problème d'allocation de fréquences à un problème de coloration de graphe. Il s'agit alors d'assigner à chaque sommet une couleur de façon à satisfaire les contraintes d'écart. Cependant, la modélisation de l'AFD par un problème de coloration a nécessité l'utilisation de variantes plus élaborées du modèle. Ces variantes visent à introduire de nouvelles contraintes ne pouvant pas être prise en compte par le modèle de base. Pour plus de détail, nous renvoyons l'auteur au livre de Rahoual et Siarry [109], à la thèse de Devarenne [126] et à la thèse de Gondran [127].

3.2.1.1. Coloration de graphe

Dans ce modèle, seules les contraintes binaires d'inégalité de ressources sont considérées. Le problème revient à colorier les sommets du graphe, de façon à ce que

chaque paire de sommets adjacents soit de couleurs différentes. Dans ce cas précis les contraintes électromagnétiques prises en compte font référence à un brouillage simple (brouilleur unique) et en co-canal exclusivement, c'est-à-dire la réutilisation de la même fréquence entre une liaison et ses voisines [142].

3.2.1.2. T-coloration de graphe

Pour modéliser les contraintes binaires d'écart qui tiennent compte des brouillages simples (brouilleur unique) en canaux adjacents, en plus du co-canal, entre les liaisons voisines, les arcs du graphe sont munis de poids. Deux trajets, u et v, sont reliés par un arc de poids d si et seulement si leurs couleurs doivent être séparées d'au moins d. Ceci sous-entend qu'un ordre doit être établi entre les couleurs. Cet ordre correspond à l'ordre des fréquences codées dans le spectre. Dans le cas où les poids des arcs seraient égaux à 1, le problème revient à un problème de coloriage classique. Dans de nombreux travaux sur le problème d'allocation de fréquence, les espacements inter-fréquence entre les liaisons sont représentés sous forme de matrice carrée de réutilisation M [107] [108].

3.2.1.3. Liste T-coloration

Ce modèle introduit la contrainte unaire de ressources bloquées. En plus d'associer un poids à chaque arc du graphe, chaque sommet est désigné par un ensemble de couleurs inutilisables. Cet ensemble peut éventuellement être vide.

3.2.1.4. T-coloration avec ensembles

Plusieurs émetteurs/récepteurs peuvent être affectés à une même liaison. C'est notamment le cas pour les réseaux cellulaires avec ou sans saut fréquences [129][130][131]. Du fait que le voisinage des émetteurs/récepteurs d'une même liaison soit identique leur regroupement au sein du même sommet est un moyen intéressant pour réduire la complexité du problème. Dans ce cas, un ensemble de couleurs doit être affecté à chaque sommet du graphe dépendant notamment, mais par exclusivement, du nombre d'émetteurs/récepteurs que le sommet contient. Pour maintenir les contraintes binaires d'inégalité de ressources entre les sommets, les couleurs affectées à un couple de sommets doivent respecter deux à deux les contraintes binaires d'écart entre ces sommets et en plus les couleurs affectés à un sommet doivent respecter deux à deux des contraintes d'écart entre elles (relation réflexive sur le sommet).

3.2.1.5. T-coloration avec ensembles sur hypergraphe

La question de la prise en compte du brouillage multiple faisant intervenir plusieurs brouilleurs a été abordée par des approches de coloration sur hypergraphe.

Comme le problème de T-coloration de graphe ne permet d'exprimer que les interférences binaires, [127] a généralisé sa définition aux hypergraphes pour rendre compte des interférences multiples. En effet, la notion de graphe est restrictive car elle correspond à des relations binaires sur les ensembles qui ne prennent pas en compte les interférences multiples. Pour simplifier l'étude du phénomène d'interférence, il est de coutume de considérer les sources d'interférences indépendantes les unes des autres. Ainsi, il est plus facile d'estimer l'impact d'un signal interférent sur le signal porteur de

l'information comme s'il était l'unique interférent. Cependant, c'est l'action conjuguée de tous les signaux interférents qu'il faut aussi considérer pour être plus proche de la réalité physique.

La T-coloration d'hypergraphe représente plus finement la réalité du phénomène. Les hyperarêtes de l'hypergraphe auront pour fonction de représenter les contraintes n-aires qui lient plus de deux variables à la fois. Le modèle de T-coloration d'hypergraphe proposé dans [127] est une approximation du problème d'affectation de fréquences initial, il est équivalent au problème initial sous certaines conditions restrictives. L'intérêt du passage à l'hypergraphe a été confirmé numériquement par des tests pour lesquels la T-coloration d'hypergraphe a donné de bien meilleurs résultats que la T-coloration de graphe sur des problèmes d'affectation de fréquences dans les réseaux sans fils [117]. Les approches algorithmiques utilisées pour traiter ce problème ont été basée sur l'utilisation de recherche locale. Il n'y a pas eu de généralisation de ce modèle pour l'instant, les références actuelles sont spécifiques à un type de système radio.

3.2.1.6. Discussion

Les approches à base de coloriage de graphe et hypergraphe constituent une bonne base théorique pour l'étude du problème d'allocation de fréquences. Cependant, ces modèles ne reflètent pas tous les aspects relatifs au problème réel tels que la polarisation des fréquences ou les phénomènes d'intermodulation. De plus, la matrice de réutilisation ne permet pas d'exprimer à elle seule les notions de poids des contraintes et des niveaux de relâchement qui leurs sont associés. Enfin les contraintes électromagnétiques qui ne s'expriment pas par un écart en fréquences fixe, telles que les contraintes harmoniques où une fréquence interfère avec des fréquences adjacentes et des fréquences multiples, ne peuvent pas être représentées facilement dans un graphe. Pour l'ensemble de ces raisons nous nous sommes orientés vers une modélisation en problème de satisfaction de contraintes qui offre un cadre formel avec plus de flexibilité pour modéliser le problème AFD.

3.2.2. AFD et satisfaction de contraintes

Comme tout problème d'affectation de ressources, le problème d'affectation de fréquences se modélise naturellement sous forme d'un problème de satisfaction de contraintes.

Le problème AFD est modélisé par un ensemble de variables représentant des émetteurs ou des liaisons radio exigeant chacun une ressource qui est un canal de fréquence ou un couple (*canal, polarisation*). Les variables du problème sont liées entre elles par des contraintes de compatibilité électromagnétique établissant les conditions de succès des communications.

Le Tableau 4 propose la transposition d'un AFD en un CSP. Afin de transformer le domaine bidimensionnel des ressources radio (*fréquence, polarisation*) d'un trajet en un domaine de scalaires, nous avons représenté le domaine d'une variable i comme une union de deux domaines : le premier noté Dh_i pour domaine de fréquences avec polarisation horizontale et le deuxième noté Dv_i pour domaine de fréquences avec polarisation verticale. On a donc $D_i = Dh_i \cup Dv_i$. Nous avons choisi cette représentation

pour des raisons de facilité et de rapidité d'accès d'une part, et d'autre part en fonction de la méthode de résolution que nous avons proposée.

CSP	AFD
X *:* l'ensemble de variables	L'ensemble des variables cibles et hors cibles
D **:** l'ensemble des domaines	L'ensemble des ressources pour les variables
C : l'ensemble des contraintes	L'ensemble des contraintes unaires, binaires et n-aires

Tableau 4 : Rapport entre CSP et AFD

3.2.3. Méthodes de résolution

Les problèmes de T-coloration ont été très largement étudiés dans la littérature [129][130][131], cependant ils ne reflètent pas tous les aspects du problème d'affectation de fréquences pour les réseaux. Dans cette section, nous présentons des méthodes de résolutions qui ont été appliquées à des problèmes d'affectation de fréquences semblables à celui traité dans le cadre du projet CELAR, notamment les instances CALMA [119][120] et les instances ROADEF [122]. Le problème AFD traité dans le projet CALMA a été enrichi au cours de l'année 2001 par le CELAR pour prendre en compte les notions de polarisations et de relâchement contrôlé des contraintes de compatibilité électromagnétique. Cette extension a donné lieu aux instances ROADEF. Cependant les instances ROADEF ne traitent toujours pas des contraintes n-aires d'intermodulation et de sommation de perturbateurs que nous avons traités.

Pour plus de détails et pour une vision plus complète, nous recommandons la lecture de l'article d'Aardal [116].

3.2.3.1. Affectation de fréquences : problème binaire sans polarisation

Le problème d'affectation de fréquences avec contrainte binaire et sans polarisation est le plus classique. Il a été très étudié dans la littérature et de nombreuses méthodes ont été adaptées pour traiter ces problèmes. La plupart de ces méthodes ont été appliquées sur les instances CALMA. Nous présentons ici les meilleures parmi celles recensées et comparées sur la page web de *Zuse Institute Berlin* dédiée aux instances CALMA, http://fap.zib.de/problems/CALMA.

Deux objectifs d'optimisation spectrale ont été traités, le *MinSpec* et le *MinFreq*, que nous expliquons ci-après.

Optimisation *MinSpec* (*Minimiser le Spectre*) : L'optimisation d'une instance du problème revient à trouver une affectation de fréquences satisfaisant toutes les contraintes et réduisant l'écart entre la plus basse et la plus haute fréquence allouée. Généralement l'optimisation de l'étendue du spectre des fréquences utilisées s'effectue par une démarche itérative. A chaque itération le problème est durci en désaffectant les variables utilisant la plus basse ou la plus haute fréquence de la solution consistante trouvée à l'étape précédente. Nous comparons ici brièvement les 8 méthodes présentées sur la page web de ZIB ainsi que la méthode *CN-Tabu* de Vasquez [123] qui est une des

meilleures méthodes référencées dans la littérature pour la résolution des problèmes d'affectation de fréquences.

Sans redonner tous les résultats, le Tableau 5 présente une synthèse des résultats obtenus par les meilleures méthodes appliquées sur 4 instances issues du projet CALMA. Pour chaque méthode en colonne nous donnons le nom du premier auteur de la méthode, le principe algorithmique de la méthode ainsi que le nombre des instances résolues à l'optimalité.

Critère	Vasquez PPC + Recherche Tabou	Aardal Programmation linéaire	Tiourine Programmation quadratique	Kolen Programmation par Contraintes	Tiourine Recherche Tabou	Boujou Recherche Tabou	Boujou Algorithme génétique	Warners Réduction de potentiel	Pasechnik Réduction de potentiel
Nb inst. résolues	4	1	1	4	4	1	1	2	4

Tableau 5 : *MinSpec* – Synthèse des résultats sur 4 instances CALMA

Les instances CALMA *MinSpec* sont réputées pour être faciles à résoudre. Les égalités entre les résultats obtenus rendent difficile de conclure quant à la qualité des méthodes appliquées. Nous pouvons cependant noter que les méthodes utilisant des techniques issues du monde de la programmation par contraintes (PPC) (Kolen et Vasquez) fournissent de bonnes performances.

Optimisation *MinFreq* (*Minimiser le nombre de Fréquences*) : L'objectif est ici de fournir une affectation de fréquences consistante utilisant le moins de fréquences possible. La démarche itérative d'optimisation du critère spectral s'avère ici plus complexe. A chaque itération plusieurs stratégies de choix ont été utilisées : retirer la fréquence maximale, la moins utilisée, la plus utilisée, alterner les choix… Le Tableau 6 présente une synthèse des résultats obtenus par les meilleures méthodes connues appliquées aux 10 instances du projet CALMA. Nous donnons pour chaque méthode le nombre d'instances résolues à l'optimalité.

Critère	Tiourine Clique Bound	Kolen PPC	Aardal Relaxation linéaire	Pasechnik Descente de gradient	Tiourine Recherche Tabou	Bouju Recherche Tabou	Warners Réduction de potentiel	Bouju Algorithme Génétique	Dupont CN-Tabu
Nb inst. résolues	8	8	7	9	7	4	6	6	9

Tableau 6 : *MinFreq* - Synthèse des résultats sur 10 instances CALMA

Les méthodes utilisant des techniques de propagation et de filtrage, telles que la méthode PPC de Kolen ou *CN-Tabu* de Dupont, ont montré de très bonnes performances sur les instances CALMA avec *MinFreq* comme objectif.

A noter qu'il n'y a pas une stratégie efficace connue pour optimiser ce critère. La plupart du temps, et c'est le cas dans *CN-Tabu*, l'optimisation de ce critère consiste à lancer la méthode de résolution d'une façon itérative tout en essayant avant chaque lancement de supprimer une ou plusieurs ressources utilisées par la meilleure solution trouvée.

Nous renvoyons le lecteur intéressé par les détails de ces méthodes à [116] [26] qui présentent une vue d'ensemble des différentes approches.

3.2.3.2. Affectation de fréquences : problème binaire polarisé (AFDP)

Nous décrivons dans cette partie le problème d'affectation de fréquences polarisées avec relâchement de contraintes tel qu'il a été présenté lors du challenge ROADEF 2001. Il s'agit dans ce cas d'allouer des fréquences polarisées à un ensemble de variables reliées par des contraintes de compatibilité électromagnétique.

Chaque contrainte du problème est décrite par une hiérarchie de niveaux de relâchement. Le niveau maximal de relâchement correspond à l'expression de la contrainte avec le niveau minimal d'exigence, soit les contraintes les plus faciles. Ce niveau d'exigence étant par conséquent le plus prioritaire. La contrainte de compatibilité électromagnétique entre les variable i et j au niveau du relâchement k s'écrit :

$$\left| f_i - f_j \right| \geq \begin{cases} \gamma_{ij}^k & \text{si } p_i = p_j \\ \delta_{ij}^k & \text{si } p_i \neq p_j \end{cases}$$

Où f_i et f_j (respectivement p_i et p_j) désignent les fréquences (respectivement les polarisations) allouées aux variables i et j.

L'objectif de la résolution du problème est de trouver une solution consistante avec le niveau de relâchement minimum k, soit le moins de relâchement possible sur les contraintes. Ensuite, pour un niveau de relâchement réalisable donné, on préférera la solution qui minimise la somme des contraintes violées au niveau k-1. Enfin, pour une qualité identique sur le deuxième critère, on préférera la solution qui minimise la somme des contraintes violées aux niveaux strictement inférieurs à k-1.

Le Tableau 7 présente une synthèse des résultats obtenus par les 6 méthodes finalistes sur les 27 participantes selon le nombre d'instances résolues au niveau optimal k* par chaque méthode. En trouvant 36 solutions sur 44 de niveau k*, *CN-Tabu* a obtenu les meilleurs résultats de la compétition.

Critère	*TS-VN*	*MH+PPC*	*RL-CC*	*LNS*	*Tabu*	*CN-Tabu*
Nb inst. résolues	32	35	28	19	20	36

Tableau 7 : Synthèse des résultats sur les 44 instances AFDP

Dans le challenge ROADEF, les méthodes de résolution finalistes peuvent se résumer à une dominance quasi totale de la Recherche Tabou (RT) hybridée avec la Programmation Par Contrainte (PPC) ; il n'y a pas eu de méthode à base de population (algorithmes évolutionnaires, fourmis, agents…) parmi les méthodes finalistes. Les résultats obtenus par ces deux types de méthode sont une des raisons qui a poussé nos travaux vers ce genre d'approches. Ci-dessous nous donnons une brève description des 6 méthodes finalistes.

Recherche Tabou sur voisinage variable (TS-VN) :

Développée par l'équipe canadienne de S. Bisaillon, P. Galinier, M. Gendreau et P. Soriano [133], la méthode TS-VN (*Variable Neighborhood Tabu Search*) est une Recherche Tabou avec voisinage variable. Dans l'optique de minimiser le niveau de relâchement, elle procède par résolution successive de trois sous-problèmes différents selon les trois critères énoncés :

- Trouver une solution réalisable au niveau k (satisfaisant toutes les contraintes des niveaux supérieurs ou égal à k).
- Sinon, trouver une solution *k+1-réalisable* qui satisfasse autant que possible des contraintes de niveau k.
- Sinon, trouver une solution *k+1-réalisable* qui minimise le nombre de contraintes violées de niveau k et inférieur.

Combinaison de méta-heuristiques et de PPC (MH+PPC) :

L'algorithme MH+PPC, développé par Y. Caseau, combine les principes de propagation de contraintes avec les méta-heuristiques. Plus précisément, partant d'un niveau de relâchement k=0 (pas de relâchement), il filtre les domaines à chaque niveau en utilisant une consistance forte, ce qui permet d'obtenir comme borne inférieure le plus petit niveau k consistant, que nous notons *kconst*. Une procédure notée *Limited Discrepancy Search* [134] est alors exécutée avec pour niveau de relâchement k égal à *kconst*. Si le nombre de contraintes non satisfaites est trop grand, le niveau de relâchement k est incrémenté. Dans le cas contraire, une recherche sur un large voisinage (*Large Neighbourhood Search* [135]) tente, à son tour, d'obtenir une solution réalisable.

Recherche locale guidée par le coût des contraintes (RL-CC) :

La troisième méthode développée par H. Gavranovic de l'IMAG à Grenoble [136] est une recherche locale guidée par le coût des contraintes (RL-CC). Elle se base sur une construction d'arbres de polarisations déterminés par les valeurs γ_{ij}^k et δ_{ij}^k des contraintes CEM au niveau de relâchement k donné. Puis les contraintes CEM sont progressivement insérées et propagées tant qu'il y a des arbres de polarisations.

Recherche à voisinage large (LNS) :

L'équipe de *P. Michelon* a proposé de résoudre l'AFDP en utilisant une métaheuristique basée sur une recherche dans un voisinage large (*Large Neighbourhood Search*) ainsi que sur des mécanismes de propagation de contraintes (cf. [143] pour plus de détails).

La méthode travaille en trois phases. La première étape calcule la borne inférieure *kconst* en relaxant des contraintes pour obtenir un arbre de couverture maximal (*Maximal Cover Tree*). Ensuite, il cherche une solution *k-réalisable*, à partir de la configuration obtenue à l'étape précédente, en essayant de satisfaire la totalité des contraintes. Pour ce faire, il augmente de 10% en 10% le nombre de contraintes à satisfaire par la configuration courante, jusqu'à obtenir une solution de niveau k. Pour finir, l'algorithme tente d'améliorer cette solution en optimisant les deux autres critères sur les viols de contraintes. Ainsi, si la solution est de niveau optimal (elle est de même

niveau que la borne inférieure), il considère simultanément les deux derniers critères. Sinon, il tente de diminuer le nombre de contraintes non satisfaites au niveau $k-1$.

Recherche Tabou seule :

Une recherche Tabou classique, a été implémentée par Schindl, Zufferey et Hertz de l'École Polytechnique Fédérale de Lausanne [137]. Un prétraitement par filtrage arc-consistant sur chaque niveau est d'abord mis en œuvre sur les contraintes impératives et sur les contraintes de compatibilité électromagnétiques CEM_k permettant d'obtenir la borne inférieure *kconst*. A chaque niveau *k*, deux algorithmes différents de Recherche Tabou sont alors utilisés d'une manière séquentielle ; le premier pour affecter les fréquences et le second pour affecter les polarisations. Dès qu'une solution est trouvée, *k* est décrémenté.

Recherche Tabou sur voisinage consistant (CN-Tabu) :

CN-Tabu (*Consistent Neighbourhood in a Tabu Search*) a été développée par M. Vasquez pour le challenge ROADEF 2001 [122]. La méthodologie mise en œuvre résout successivement chaque problème de niveau *k* en utilisant en prétraitement une procédure standard de filtrage arc-consistant. Cela permet de supprimer les valeurs qui ne sont pas ou plus cohérentes avec les écarts γ^k et δ^k. Elle décrémente les niveaux dès lors que *CN-Tabu* trouve une solution de niveau k. Plus de détails sur la méthode CN-Tabu ont été donnés dans le chapitre précédent.

3.2.3.3. Affectation de fréquences : problème n-aire polarisé

Les contraintes n-aires dans les AFD faisant état de contraintes d'intermodulation ou de sommation de perturbateurs, ont été abordées récemment dans la littérature par quelques études [114] [115] [117].

Les premières méthodes de résolution datent de l'année 2003. Un algorithme de Recherche Tabou basé sur le principe de voisinage consistant a été développé par Vlasak et Vasquez [111], une procédure de recuit simulé par Sarzeaud [112], une méthode exacte hybride entre programmation par contraintes et programmation linéaire PPC-PLNE par Oliva [125] et une méthode de grands voisinages LSSPER par Palpant [110]. Ces méthodes ont été développées dans le cadre d'un projet bilatéral avec le CELAR et sur des instances privées.

La seule méthode exacte basée sur la programmation linéaire en nombre entier proposée par Oliva [125] a montré ses limites ; les solutions trouvées par cette méthode sur les instances traitées sont très loin de celles obtenues par les méthodes approchées. Dans sa thèse, Oliva a combiné la Programmation Linéaire avec la PPC. Elle cherche tout d'abord à déduire des contraintes binaires à partir des contraintes n-aires, puis elle utilise la PL sur le problème relaxé (contraintes binaires seulement) afin de prouver sa réalisabilité. Si le problème est réalisable, une recherche arborescente (PPC) est lancée avec une borne supérieure ayant pour évaluation la valeur de la solution trouvée. Si le problème est irréalisable une recherche arborescente est lancée avec une borne inférieure calculée à partir de l'évaluation d'une contrainte n-aire.

Les méthodes approchées (recuit simulé, Recherche Tabou) ont utilisées les contraintes n-aires d'une manière classique en les intégrant dans la fonction d'évaluation pour le choix d'un voisinage. Ces méthodes ont montré une bonne performance sur

l'ensemble des instances traitées. Dans le cadre du projet bilatéral avec le CELAR, il n'y a pas eu de méthodes approchées meilleures que d'autres.

3.2.4. Discussion et orientation

A travers cette recherche, nous constatons que l'état de l'art sur le problème qui nous est présenté dans le cadre du projet est très fourni mais à travers une multitude de sous-problèmes. Une des spécificités de notre problème est le rassemblement de composants jusqu'alors traités séparément qui rendent les choses encore plus complexes : allocation de fréquences discrète, notion de polarisations, contraintes et objectifs avec palier de relâchement, liste de critères très différents à prendre en compte et présence de contraintes n-aires. Tout cela dans un contexte où la combinatoire est gigantesque, les environnements d'exécution parallèle non autorisés et les ressources machine ainsi que le temps de calcul limité. Résumons en soulignant que nous n'avons pas de référence sur le traitement du problème complet que nous venons d'évoquer.

Le problème d'affectation de fréquences que nous avons décrit précédemment se modélise naturellement comme un CSP. Pour les contraintes binaires, la propagation de contraintes est très efficace vue leurs natures physiques et leurs formulations mathématiques ; nous avons constaté qu'une fréquence dans le domaine d'une variable donnée possède souvent peu de fréquences supports dans le domaine de la variable opposée. La présence des contraintes unaires qui réduisent d'une manière efficace l'espace de recherche est aussi un bon atout pour l'utilisation de propagation de contraintes. Seule la nature des contraintes n-aires et leurs aspects cumulatifs rendent les choses difficiles ; dans ce contexte les méthodes de propagation de contraintes ne sont pas encore au point même s'il existe plusieurs solutions dans les solveurs les plus connus (ILOG, CHOCO, COMET…). Du point de vue de la combinatoire et des objectifs d'optimisation à traiter en sus de la satisfaction des contraintes, la capacité de diversification de la Recherche Tabou peut apporter un réel supplément comme cela a été le cas pour les problèmes de référence que nous avons cités (CALMA et ROADEF2001). Chercher la meilleure solution pour un objectif donné tout en respectant les contraintes est une tâche extrêmement difficile pour laquelle un couplage de PPC avec une autre méthode est nécessaire.

En l'état actuel des connaissances, il n'est pas possible de concevoir une méthode exacte pour l'optimisation du fait de la taille des problèmes et de la nature de certains objectifs non linéaires ; et les méthodes exactes sont non efficaces pour les instances AFD de grandes tailles, en témoignent les travaux de C. Oliva [125] qui soulignent les limitations de ce type d'approche. Il n'est pas possible non plus d'envisager une méthode trop gourmande en temps de calcul comme les métaheuristiques à base de populations du fait des temps de recherche restreints (colonies de fourmis, algorithmes évolutionnaires, essaims particulaires, systèmes multi-agents…). Bien qu'il y ait d'autres formes d'hybridation, les méthodes hybridant des techniques issues de la PPC avec la Recherche Tabou se sont montrées les plus performantes sur des problèmes proches du nôtre.

Ces constatations nous ont amené à concevoir une méthode qui tire ses avantages de la formulation du problème en un *CSP,* de l'efficacité de la propagation des contraintes et de la rapidité et la capacité de diversification de la recherche Tabou. Par ailleurs nous avons vu un réel intérêt pour les méthodes à base de *nogood* de part les indicateurs d'explication qu'elles peuvent fournir lorsque la recherche est dans une impasse

(*deadend*). Nous croyons que les *nogoods* peuvent parfaitement s'insérer dans un processus de recherche en harmonie avec la programmation par contraintes et la Recherche Tabou. La difficulté avec les *nogood* est la gestion de ces éléments : comment les trouver et que décider à partir d'eux ? Tout en restant dans une approche efficace, c'est-à-dire que cette gestion doit apporter plus que ce qu'elle ne coûte dans la recherche de solutions au problème considéré.

L'association de l'ensemble de ces briques algorithmiques est le challenge que nous avons relevé en proposant une nouvelle méthode hybride appelée *Tabu-NG* pour une Recherche Tabou couplée avec un système de gestion de *nogood*. Dans les deux parties suivantes nous traiterons successivement l'allocation de fréquences discrètes puis le problème complet avec contraintes n-aires pour les différents objectifs.

3.3. Méthode *Tabu-NG* pour le problème AFD classique

Nous allons au cours de cette section décrire l'application de la méthode *Tabu-NG,* que nous avons introduite globalement en fin de chapitre précédent pour la résolution du problème d'affectation de fréquence proposé par le CELAR dans le cadre du projet européen CALMA (cf. section 3.1.6.1).

3.3.1. Prétraitement

3.3.1.1. Filtrage des domaines et réduction de la taille du problème

L'objectif de cette phase préliminaire est de réduire la taille du problème, et par la suite la taille de l'espace de recherche, en supprimant les valeurs inconsistantes des domaines des variables. Cette partie s'avère nécessaire et profitable de par la nature de ses instances issues d'application réelles.

Problème	nb var	Densité du graphe	nb ctr	nb aff	nb val	Résultat AC
CELAR01	916	0,0132	5548	0	36200	AC
CELAR02	200	0,062	1235	0	80004	AC
CELAR03	400	0,0345	2760	0	15892	AC
CELAR04	680	0,0171	3967	280	26856	13868 (51,6%)
CELAR05	400	0,0325	2598	0	15768	12046 (76,4%)
CELAR11	680	0,0177	4103	0	26856	AC
graph01	200	0,057	1134	0	6920	AC
graph02	400	0,0281	2245	0	14624	AC
graph03	200	0,057	1134	0	7820	340 (4,3%)
graph04	400	0,0281	2244	0	15592	776 (4,9%)
graph08	680	0,0163	3757	0	25628	AC
graph09	916	0,0125	5246	0	36092	AC
graph10	680	0,0169	3907	0	26980	386 (1,4 %)
graph14	916	0,0111	4638	0	36716	AC

Tableau 8 : Réduction de domaines sur 15 instances

La dernière colonne du Tableau 8 montre le résultat de la phase de prétraitement, elle donne le pourcentage de valeurs éliminées. AC signifie que le problème est arc-

consistant tel qu'il est proposé et que la phase de prétraitement n'a supprimé aucune valeur dans le domaine des variables.

Aucune réduction de domaine n'est possible sur la majorité des instances fournies dans le cadre du projet CALMA. Cette réduction n'est effective que pour 5 des 14 instances (lignes grises) et de façon efficace sur un nombre encore plus restreint, ainsi qu'on peut le constater sur le Tableau 8 où deux instances seulement ont des réductions importantes (> 50% de valeurs éliminées).

Cette procédure est réalisée une seule fois avant le lancement de la phase de recherche effective d'une solution du problème. Dans le cas où toutes les contraintes du problème sont binaires, cas des AFD classiques, un algorithme qui force l'arc-consistance est nécessaire et suffisant pour éliminer les valeurs inconsistantes dans les domaines des variables.

La procédure de filtrage choisie se base sur l'algorithme AC-3 choisi en raison de sa simplicité d'implémentation et son usage réduit de la mémoire. La procédure de filtrage nommé *Algofiltrage* est décrite par le code Algorithme 14.

L'Algorithme 11 commence par traiter les contraintes unaires du problème qui porte sur une seule variable. Pour cela le domaine valide de la variable concerné par une contrainte unaire est réduit aux seules valeurs respectant la contrainte. Ensuite, une procédure de FC (*Forward Checking*) est effectuée (Algorithme 2). Cette phase sert à propager l'instanciation des variables hors cibles (variables pré-affectées) aux variables voisines qui partagent des contraintes en commun avec elles. Par exemple, si une variable hors cible x avec une fréquence f est reliée à une variable y par une contrainte d'écart minimum d, les fréquences entre f-d et f+d sont à éliminer du domaine de la variable y.

La dernière étape consiste à filtrer les domaines des variables en utilisant l'arc-consistance (Algorithme 3). L'algorithme AC-3 n'étant utilisé qu'une seule fois en prétraitement, la lenteur de la procédure n'est donc pas préjudiciable à la rapidité de l'algorithme de recherche. La complexité temporelle de l'algorithme de filtrage proposé est de l'ordre de $O(d^3)$ avec d la taille maximale des domaines.

```
Function AlgofiltageUnaire(X, D, C)
1.    C_Unaires = C.GetAllContraintesUnaires();
2.    foreach c in C_Unaires do
3.          var = c.GetVar();
4.          foreach val in D_var do
5.                if not c.verify(val) then Remove val from D_var
6.          endforeach
7.    endforeach
```

Algorithme 11 : Prétraitement et filtrage par contraintes unaires

```
Function ForwardChecking(S, X, D, C)
1.    FutureVarsFC = Les variables instanciées dans S
2.    while FutureVarsFC is not Empty do
3.          varFC = FutureVarsFC.pop()
4.          if (checkFC(varFC) == EMPTY_DOMAIN) then
5.                return EMPTY_DOMAIN;
6.    endwhile;
```

```
Function checkFC(varFC)
1.    ctrList = Constraints of varFC
2.    foreach c in ctrList do
3.        otherVar = the 2nd variable of constraint c;
4.        if otherVar is not assigned then
5.            foreach v_o in Dom_otherVar do
6.                if v_o is inconsistent with varFC then
7.                    Remove v_o from Dom(otherVar);
8.                    if Dom_otherVar is Empty then
9.                        return EMPTY_DOMAIN;
10.                   endif;
11.               endif
12.           endforeach
13.       endif
14.   endforeach
```

Algorithme 12 : Prétraitement : Forward Checking

```
Function AC3(S, X, D, C)
7.    FutureVarsAC = Les variables libres
8.    while FutureVarsAC is not Empty do
9.        varAC = FutureVarsAC.pop()
10.       if (checkAC(varAC, futureVarsAC) == EMPTY_DOMAIN) then
11.           return EMPTY_DOMAIN;
12.   endwhile;

Function checkAC (varAC, futureVarsAC)
1.    ctrList = Constraints of varAC
2.    foreach c in ctrList do
3.        otherVar = the 2nd variable of constraint c;
4.        if (otherVar is not assigned) then
5.            foreach v_o in Dom_otherVar do
6.                if no support for v_o in Dom_varAC then
7.                    Remove v_o from Dom_otherVar;
8.                    if Dom_otherVar is Empty return EMPTY_DOMAIN;
9.                    futureVarsAC.add(otherVar);
10.               endif
11.           endforeach
12.       endif
13.   endforeach
```

Algorithme 13 : Prétraitement : Filtrage par Arc-Consistance

```
Function Algofiltage(S, X, D, C)
1.    if AlgoFiltrageUnaire(X, D, C) == EMPTY_DOMAIN then
2.        return ProblemeIrrealisable;
3.    if ForwardChecking(S, X, D, C) == EMPTY_DOMAIN then
4.        return ProblemeIrrealisable;
5.    if AC-3(S, X, D, C) == EMPTY_DOMAIN then
6.        return ProblemeIrrealisable;
```

Algorithme 14 : Prétraitement : Algorithme de filtrage

Pour illustrer le fonctionnement de la procédure de filtrage AC-3 prenons l'exemple suivant. Soit un problème AFD avec $V = \{x, y\}$, $D_x = D_y = \{1, 2, 3, 4, 5\}$ et C un ensemble de contraintes avec une seule contrainte $c : f_x > f_y+2$.

Afin d'éliminer les valeurs inconsistantes dans le domaine de la variable x selon la contrainte c, nous avons besoin de vérifier pour chaque fréquence f_x appartenant à D_x l'existence d'au moins un support dans le domaine de la variable y. Prenons la fréquence 1 dans le domaine de la variable x, cette valeur ne possède aucun support dans le domaine de la variable y selon la contrainte c. Cette valeur sera donc supprimée de D_x.

La routine de base nécessaire pour appliquer cette vérification est donnée par la fonction *checkAC* (voir Algorithme 13). La complexité temporelle de cette fonction est $O(d^2)$ avec d la taille maximale des domaines des variables du problème.

3.3.1.2. Arc-Consistance et affectation de fréquences

Les instances du projet CALMA sont décrites par des contraintes binaires d'écart fixe du type $|f_i - f_j| = \varepsilon_{ij}$ et des contraintes binaires d'écart minimal du type $|f_i - f_j| \geq \varepsilon_{ij}$. Supposons que les domaines soient triés par ordre croissant, et que la vérification de l'appartenance d'un élément à un domaine se fait avec une complexité de O(1). Nous proposons dans cette partie, un traitement mathématique de ces contraintes afin d'améliorer les performances de la fonction *checkAC*.

Pour une contrainte binaire d'écart fixe, la vérification de la consistance d'une fréquence f_i appartenant au domaine de la variable i se fait en vérifiant l'existence d'une des deux valeurs $\varepsilon_{ij} - f_i$ et $\varepsilon_{ij} + f_i$ dans le domaine de la variable j. Si ces deux fréquences sont absentes alors la valeur $f_i \in D_i$ est déclarée sans support dans D_j. Dans ce cas la fréquence f_i est supprimée de D_i.

Pour une contrainte binaire d'écart minimal en fréquences, la vérification de la consistance d'une fréquence f_i appartenant au domaine de la variable i se fait en vérifiant la relation suivante: $f_j^{\min} \leq f_i - \varepsilon_{ij}$ ou $f_j^{\max} \geq f_i + \varepsilon_{ij}$. $f_j^{\min}$ désigne la valeur minimale dans D_j et $f_j^{\max}$ la valeur maximale.

Cette condition est suffisante pour vérifier l'existence d'un support pour une fréquence $f_i \in D_i$ d'une variable i dans le domaine de la variable opposé D_j. La complexité algorithmique de la fonction *checkAC* reste inférieure à $O(d)$ avec d la taille maximale des domaines. Par la suite l'algorithme de filtrage proposé aura une complexité de l'ordre de $O(d^2)$.

3.3.1.3. Composantes Connexes

Il est judicieux d'appliquer en prétraitement une phase qui consiste à identifier les différentes composantes connexes dans le graphe de contraintes G, en vue de pouvoir traiter indépendamment les uns des autres les problèmes $P_1, \ldots, P_k$ qu'elles définissent. Cependant toutes les instances que nous avons traitées dans cette thèse ne sont pas décomposables, chaque instance a une seule composante connexe. Nous n'avons donc pas eu de gain lié à cette procédure.

Tabu-NG est une méthode qui se base sur la propagation de contraintes et la recherche Tabou. Le traitement de toutes les composantes connexes d'un problème à la fois, revient à les traiter parallèlement, ce qui peut agir sur la vitesse et donc pour un temps donné sur la qualité de la solution finale. D'une manière générale, on cherche à décomposer un problème en plusieurs sous problèmes afin de faciliter sa résolution.

3.3.2. Propagation de contraintes et identification des *nogoods*

3.3.2.1. Propagation de l'extension d'une configuration partielle consistante

Dans cette partie, nous allons détailler la procédure de propagation de contraintes. Cette procédure intervient suite à chaque extension d'une configuration partielle courante consistante. Cet algorithme aura comme mission l'élimination de toutes les valeurs inconsistantes qui résulte de la propagation de la dernière affectation ajoutée à la configuration partielle courante. Cet algorithme est responsable aussi de l'identification d'un *nogood* lorsqu'un *deadend* a lieu.

Etant donné que les instances du projet CALMA que nous traitons dans cette section portent sur des contraintes binaires et ne contiennent pas de contraintes n-aires, un algorithme de propagation de contraintes par arc-consistance suffit pour éliminer toutes les valeurs inconsistantes dans les domaines des variables libres. Plusieurs algorithmes de propagation de contraintes ont été proposés dans la littérature tels que : AC-3, AC-4, AC-6, AC-2001, etc. Parmi ces algorithmes, seul AC-4 permet d'identifier un *nogood* de bonne qualité en modifiant l'algorithme ; nous en donnons les raisons ci-après. Les travaux expliqués dans cette partie se basent principalement sur l'algorithme MAC-DBT de Jussien [45], l'algorithme AC-4 ainsi que sur les travaux menés sur les explications d'élimination de Jussien [139] [140] [141].

Par définition un *nogood* est une configuration partielle qui ne peut pas faire partie d'une solution réalisable. Après la procédure de propagation de contraintes, si le domaine d'une variable devient vide, un sous ensemble de la configuration partielle courante est considéré comme *nogood*. La question qui se pose est alors comment identifier les explications d'élimination ? Nous passons par des structures de données spécifiques pour marquer les éliminations de données dans les domaines de définition des variables.

Etape 1 : enregistrement des supports par contraintes

Dans le domaine de chaque variable, nous associons à chaque valeur, un ensemble noté *explications d'élimination* qui contient les affectations responsables de l'élimination de la valeur. Le stockage des explications d'élimination pour toutes les valeurs des variables est d'une complexité spatiale de l'ordre de **O(n*n*d)**. Cette complexité est déterminée en associant à chaque variable *d* valeurs possibles donc *n*d* valeurs totales. A chaque association (*variable, valeur*) on pourra, dans le pire des cas, avoir une explication qui inclut toutes les variables du problème.

L'élimination d'une valeur *a* d'une variable *i* se fait selon une contrainte donnée c_{ij}. Cependant la cause de l'élimination de *a* n'est pas toujours la dernière affectation ajoutée à la configuration partielle courante mais la cause de l'élimination de tous les supports de cette valeur dans le domaine de la variable opposée *j* selon la contrainte c_{ij}. Pour cela, nous avons besoin de savoir pour chaque contrainte c_{ij} tous les supports de

chaque valeur de i dans le domaine de la variable j. L'algorithme de filtrage AC-4 est le seul algorithme qui permet d'utiliser une telle structure.

Un algorithme d'initialisation est nécessaire afin de remplir les structures de données nécessaires pour identifier les *nogoods* et pour faciliter le filtrage, ces structures sont les mêmes que celles utilisées par AC-4 [11]. L'algorithme d'initialisation des structures est donné dans Algorithme 15. Le but est de stocker pour chaque valeur a d'une variable i tous les supports et leurs nombres dans le domaine de la variable j selon une contrainte c_{ij}. A noter que la complexité spatiale de cet algorithme est de l'ordre de $O(|C|*d^2)$ avec $|C|$ le nombre de contraintes du problème.

```
Function initialisation()
1.    foreach c_ij ∈ C do
2.    foreach a in Dom_i do
3.        foreach b in Dom_j do
4.            if c.verify(a,b) then
5.                supports[c_ij,a] = supports[c_ij,a] U {(j, b)};
6.                nbSupports[(i,j),a] += 1;
7.            endif;
8.        endforeach;
9.    endforeach;
```

Algorithme 15 : Initialisation et remplissage des structures

Etape 2 : enregistrement de l'explication d'élimination par valeur

Notons par la suite *ElExpl* la matrice qui contient l'explication d'élimination de chaque couple (*variable, valeur*). Par exemple, *ElExpl(i, a)* contiendra l'explication d'élimination de la valeur a dans le domaine de la variable i.

L'algorithme de propagation est appelé après l'extension d'une configuration partielle courante. Il est donc responsable de l'élimination des valeurs inconsistantes dans les domaines des variables libres en propageant seulement l'effet de la nouvelle affectation ajoutée à la configuration partielle. Cet algorithme est responsable aussi de l'identification d'un *nogood* si un *deadend* a lieu.

La procédure de propagation est donnée dans Algorithme 16. Suite à l'assignation de la valeur a à la variable i, le domaine de la variable i est alors restreint à la valeur a tout en désignant l'affectation (i, a) comme explication d'élimination pour toutes les autres valeurs de i, et en propageant ainsi l'effet des valeurs supprimées sur la totalité du graphe de contraintes en appelant la fonction *PropagationSuppression*.

```
Function AlgoFiltrageApresExtension (D, S, C, i, a)
1.    foreach b ∈ D_i and b#a do
2.        ElExpl(i,b) = {(i, a)}; //L'explication est la nouvelle affectation
3.        Q = Q ∪ {(i,a)} ; //Ajout de valeurs supprimées à la liste Q
4.        D_i = D_i \ {b} ; //Supprimer la valeur b de D_i
5.    endforeach
6.    S = S ∪ {(i,a)} //Extension de la configuration partielle
7.    E = PropagationSuppression(Q); //Propagation de la suppression des valeurs
8.    return E;
```

Algorithme 16 : Filtrage après extension de la configuration partielle courante

L'Algorithme 17 décrit la fonction appelée *PropagationSuppression* qui vise à propager les valeurs que l'on vient de supprimer dans le domaine de la nouvelle variable instanciée *i*. L'idée est d'établir l'arc-consistance dans la totalité du graphe suite à l'ajout de la nouvelle affectation. Si la propagation de cette nouvelle affectation vide un domaine d'une variable, cette fonction retourne le *nogood* responsable de cette situation.

La fonction *PropagationSuppression* prend chaque couple (*i*, *a*) de la liste *Q*, puis elle propage le retrait de la valeur *a* du domaine de la variable *i* sur chaque contrainte de *i*. Pour chaque contrainte c_{ij}, pour chaque support *b* de *a* dans le domaine de la variable *j*, on retranche de 1 le nombre de supports de *j*=*b* dans le domaine de la variable *i*. Dans le cas où le nombre de supports de *j*=*b* dans *i* devient 0, on supprime *b* du domaine de la variable *j* et son explication d'élimination sera l'union des explications d'élimination de tous les supports *a'* de *b* dans le domaine de la variable opposée *i*. On ajoute à *Q* le couple (*j*,*b*) afin de propager l'effet du retrait de *b* du domaine de la variable *j*. Dans le cas ou le domaine de la variable *j* devient vide, un *deadend* est alors atteint et un *nogood* est identifié qui est l'union des explications d'élimination de toutes les valeurs dans le domaine de la variable *j*.

Function PropagationSuppression (*Q*)
1. while **Q not Empty** do
2. (i, a) = Un couple dans Q;
3. foreach $c_{ij} \in C$ do // toutes les contraintes de la variable i
4. foreach $b \in D_j$ **and** $b \in$ **Supports(**c_{ij}, a**)** do
5. nbSupports((j,i), b)--; //le nbre de supports de j=b dans i
6. if **nbSupports(*j*, *i*, *b*) == 0** then
7. $D_j = D_j \setminus \{b\}$;
8. $ElExpl(j, b) = \bigcup_{a' \in supports(C_{ji}, b)} ElExpl(i, a')$
9. $Q = Q \cup \{(j, b)\}$
10. if $D_j == \phi$ then return $\bigcup_{a' \in Dom_j} ElExpl(i, a')$ //*nogood*
11. endif ;
12. endforeach;
13. endforeach;
14. endwhile;
15. return NULL

Algorithme 17 : Propagation de l'effet d'une valeur supprimée d'un domaine d'une variable

Les algorithmes de propagation que nous venons de donner sont appelés régulièrement par la méthode *Tabu-NG*, leur complexité spatiale et temporelle ont donc une incidence directe sur les performances de la méthode. Ils doivent être implémentés avec soin.

3.3.2.2. Propagation de la réparation d'une configuration partielle

Dans cette partie, nous détaillons la procédure de propagation appliquée à la suite de la réparation d'une configuration partielle courante. Une réparation est nécessaire lorsque l'algorithme aboutit à un *deadend*.

La réparation d'une configuration partielle S consiste à annuler une affectation parmi l'ensemble des affectations appartenant à S. L'algorithme de propagation lancé après cette phase consiste à rendre arc-consistant le graphe de contraintes du problème en propageant l'effet de l'annulation d'une affectation.

La solution la plus simple du point de vue algorithmique, est de restaurer les domaines de toutes les variables et d'appliquer à nouveau la propagation de chaque affectation restante dans la configuration partielle courante. Cette solution est cependant très coûteuse en termes de temps. Nous avons donc préféré utiliser des structures de données spécifiques pour rendre le graphe de contraintes arc-consistant en propageant l'annulation d'une affectation. La procédure de propagation après réparation est évoquée en détail ci-dessous, elle se base sur la procédure implémentée pour l'algorithme MAC-DBT [45].

La fonction *AlgoFiltrageApresReparation* donnée dans l'Algorithme 18 propage l'effet de l'annulation d'une affectation *(i,a)* de la configuration partielle courante S.

```
Function AlgoFiltrageApresReparation(D, S, C, i, a, ng, Δ)
1.    //Mise à jour des domaines
2.    ng' = MiseAJourDomain({(k,c)}/ (i,a)∈ElExpl(k,c));
3.    //Annulation de l'affectation dans S
4.    S=S\{(i,a)}
5.    //Parfois il existe plusieurs nogoods dans la même solution, ou le nogood
6.    //repéré n'est pas minimal, d'où le lancement de cette phase
7.    if ng' is not NULL then
8.          (j, b) = ReparerSolution(S, Δ, ng')
9.          AlgoFiltrageApresReparation(D, S, C, j, b, ng', Δ)
10.   endif ;
11.   if ng\(i,a) fait encore partie de la configuration partielle courante then
12.         //La cause de l'élimination de l'affectation annulée est le reste du nogood
13.         ElExpl(i,a) = ng\{(i,a)}
14.         //Mise à jour du domaine de la variable i
15.         D_i=D_i\a
16.         Q = Q ∪ {(i,a)}
17.         //Propagation de l'affectation annulée
18.         ng' = PropagationSuppression(Q)
19.         if ng' is not NULL then
20.               (j, b) = ReparerSolution(S, Δ, ng')
21.               AlgoFiltrageApresReparation(D, S, C, j, b, ng', Δ)
22.         endif ;
23.   endif;
```

Algorithme 18 : Filtrage après la réparation d'une configuration partielle courante

Tout d'abord, cette fonction (Algorithme 18) met à jour les domaines en appelant la fonction *MiseAJourDomain* et en passant en paramètre pour cette fonction tous les couples *(k,c)* dont l'affectation *(i,a)* fait partie de son explication d'élimination. *ng'* contient un éventuel nouveau nogood malgré la mise à jour des domaines. Ensuite, l'affectation *(i,a)* est enlevée de la configuration partielle courante S. Si après la mise à jour des domaines, un *deadend* est à nouveau détecté (ligne 7), cela signifie que la réparation faite n'est pas suffisante et on a besoin de lancer une nouvelle phase de

réparation, d'où le lancement récursif de la phase de réparation conduisant à une nouvelle réparation à faire sur un nouveau couple *(j,b)* (lignes 8 et 9 de l'algorithme). Le test de la ligne 11 de l'algorithme vérifie si le *nogood* récupéré sans l'instanciation annulée fait toujours partie de la configuration partielle courante ; si c'est le cas on interdit à la variable *i* qui vient d'être désaffectée de reprendre la valeur *a* puisqu'on sait que le couple *(i,a)* avec le reste du *nogood* existant conduit à un *deadend*. Le reste du *nogood* est donc l'explication d'élimination de *(i,a)* ; *a* est supprimée du domaine de la variable *i* et une phase de propagation du retrait d'une valeur est lancée. Suite à cette phase on peut tomber à nouveau sur un *deadend* qui demande une nouvelle réparation récursive sur une autre affectation *(j,b)*.

Suite à la réparation d'une configuration partielle courante en annulant une affectation, une mise à jour des domaines est nécessaire. La fonction *AlgoFiltrageApresReparation* met à jour les domaines en appelant la fonction *MiseAJourDomain* et en passant en paramètre pour cette fonction tous les couples dont l'affectation annulée fait partie de leurs explications d'élimination. Ces couples sont dans la liste *listeAffectation***.** Pour chaque couple *(i, a)* de la liste *listeAffectation*, la valeur a est réinsérée dans le domaine de la variable *i* (ligne 2 de l'algorithme) puis le nombre de supports dans la variable opposée est mis à jour (lignes 4, 5 et 6). A partir de la ligne 11, l'algorithme parcourt pour chaque couple *(i, a)* de la liste *listeAffectation* toutes les contraintes c_{ij} et vérifie si ce couple possède un support dans le domaine de la variable *j* ; si ce n'est pas le cas le couple *(i,a)* est inséré dans une liste *Q* (ligne 13) afin que la valeur *a* soit supprimée du domaine de la variable *i* (ligne 16).

Function *MiseAJourDomain*(*listeAffectation*)
1. foreach $(i, a) \in$ *listeAffectation* do
2. $D_i = D_i \cup \{a\}$ //Mise à jour du domaine de la variable *i*
3. //Mise à jour des nombres de supports
4. foreach $\boldsymbol{c_{ij} \in C}$ do
5. foreach $\boldsymbol{b \in D_j}$ / $\boldsymbol{b \in supports(c_{ij}, a)}$ do nbsupports(*j*, *i*, *b*)++;
6. endforeach;
7. endforeach;
8. //Si une valeur dans le domaine d'une variable ne possède pas de support
9. //dans le domaine de la variable opposée alors cette valeur doit être supprimée
10. //et il faut propager l'effet de sa suppression
11. foreach $\boldsymbol{(i,a) \in listeAffectation}$ do
12. if $\exists c_{ij} \in C$ /**(*i, a*) n'a pas de support dans D_j pour c_{ij}** then
13. $Q = Q \cup \{(i,a)\}$
14. endif;
15. endforeach,
16. return PropagationSuppression(*Q*)

Algorithme 19 : Mise à jour des domaines après la réparation de la configuration partielle courante

3.3.3. Paramètres de la méthode

Cette partie décrit les paramètres utilisés par la méthode *Tabu-NG* pour résoudre les problèmes d'affectation de fréquences AFD.

3.3.3.1. Taille de la liste Tabou des nogoods Γ

Le stockage des *nogood*s est une des premières questions soulevées par l'implémentation de la méthode *Tabu-NG*. L'idéal serait de stocker tous les *nogoods* identifiés au fur et à mesure de la recherche dans une liste ce qui permet de garantir la complétude de la méthode. Cette première approche est cependant limitée car le nombre de *nogoods* peut croître de manière non contrôlée. L'utilisation de la notion de dominance entre les *nogoods* aide à mieux maîtriser cette croissance mais ne fournit aucune garantie quant aux risques de débordement mémoire. Pour cette raison une taille maximale de la liste des *nogoods* a été définie et fixée empiriquement à 15000 *nogoods* pour tous les problèmes. A noter que cette limite n'a jamais été atteinte pour les AFD que nous avons traités. Le taux de remplissage se situe entre 0% et 15%.

3.3.3.2. Durée Tabou utilisée

La réparation d'une solution consiste à annuler une partie des affectations de la configuration partielle courante ayant conduit à un *deadend*. Comme expliqué dans l'algorithme général en chapitre 1 (cf. section 2.3.5), une affectation annulée est considérée Tabou pour une certaine durée calculée en nombre d'itérations de façon déterministe. La durée Tabou d'une affectation (*i, a*) est fixée au nombre de fois ou la valeur *a* a été assignée à la variable *i*, donc plus une valeur revient souvent plus elle est écartée longtemps. Cette durée est donnée par :

$$\Delta(i,a) = iterationCourante + nbreMvt(i,a)$$

3.3.3.3. Heuristique de choix du voisinage

A chaque itération de la méthode *Tabu-NG*, une variable libre est sélectionnée pour étendre la configuration partielle courante. Nous avons testé séparément deux heuristiques simples pour le choix de la variable à instancier. La première notée *MRV* pour *Minimum Remaining Values* est une heuristique qui choisit la variable qui possède le domaine restant le plus petit au moment du choix de variable. La deuxième connue dans la littérature sous le nom *dom/deg* choisit la variable qui possède le degré le plus fort entre les variables de même cardinalité sur le critère MRV, soit *|MRV|/Degree*.

3.3.3.4. Heuristique de choix de la valeur

Le choix de la valeur *a* à affecter à la variable libre *i* choisie doit obéir à plusieurs règles : la valeur *a* doit appartenir au domaine filtré de la variable choisie *i*, le couple (*i,a*) ne doit pas être en statut tabou et aucun *nogood* stocké dans Γ ne doit figurer comme un sous ensemble de la configuration étendue. La première valeur du domaine parcouru de bas en haut qui vérifie ces conditions est affectée à la variable.

3.3.4. Stratégies d'optimisation

Pour traiter les instances du projet CALMA et leurs deux objectifs *MinFreq* et *MinSpec*, nous déclinons une stratégie par objectif :

- Une approche pour résoudre les instances dont le critère est de minimiser la largeur du spectre utilisé (*MinSpec*). Vu la nature des contraintes binaires d'écart, cet objectif revient à minimiser la plus haute fréquence utilisée dans la solution.
- Une approche pour aborder les instances dont l'objectif est d'utiliser le moins de fréquences différentes dans une solution (*MinFreq*). Trois alternatives de stratégie ont été implémentées notées *MinFreq (a, b* ou *c)* par la suite.

3.3.4.1. MinSpec

Au regard de l'objectif *MinSpec*, une stratégie évidente [124] consiste à relancer la méthode de résolution d'une manière itérative. Après chaque itération la fréquence la plus haute f_{Max} utilisée dans la solution trouvée est supprimée des domaines des variables, ainsi que les fréquences du domaine supérieures à f_{Max}. *Tabu-NG* doit donc résoudre une série de CSP successifs, dont les domaines des variables sont progressivement réduits par le haut. Notons que f_{Max} n'est pas forcement la plus haute fréquence des domaines mais celle utilisée dans la solution.

Afin de profiter au mieux des informations obtenues au cours des recherches précédentes, toutes les variables affectées à f_{Max} dans la solution courante sont désaffectées. La configuration partielle et consistante obtenue sert alors de point de départ à la nouvelle recherche de *Tabu-NG*, plutôt que de repartir d'une configuration vide. La liste Tabu des *nogood*s Γ est également conservée, on supprime de cette liste tous les *nogood*s dont f_{Max} fait partie. Le processus itératif prend donc toute son importance du fait de la reprise de la recherche d'un point de départ issu de la recherche précedente. L'apprentissage de l'algorithme sur les *nogoods* est un bénéfice très important dans cette méthode hybride.

3.3.4.2. MinFreq(a)

Afin de traiter les instances répondant à l'objectif *MinFreq*, nous avons dans un premier temps implémenté une stratégie testée par [124], ainsi la fréquence la moins utilisée dans la solution, que nous notons $f_{MinUsed}$, est supprimée de la solution trouvée et de tous les domaines des variables, puis on relance la recherche à partir de la configuration partielle consistante en conservant comme pour *MinSpec* les *nogoods* trouvés ne contenant pas cette fréquence.

Tabu-NG résout encore une série de *CSP* imbriqués, puisque les variables et les contraintes sont les mêmes, et les domaines de valeurs sont progressivement réduits. La configuration partielle à partir de laquelle *Tabu-NG* reprend sa recherche pour le *CSP* suivant est obtenue en supprimant $f_{MinUsed}$ de la solution. Cette stratégie nécessite de maintenir une structure de données comptabilisant le nombre de fois que chaque fréquence est utilisée.

3.3.4.3. MinFreq(b)

L'approche *MinFreq(b)* alterne des phases de recherche en agissant sur f_{Max} ou $f_{MinUsed}$, cette alternance a été proposée dans [124]. Si une solution est trouvée par *Tabu-NG*, la fréquence correspondant à la stratégie de la phase courante, f_{Max} ou $f_{MinUsed}$, est

supprimée. *MinFreq(b)* change de stratégie toutes les deux phases et commence sa résolution par la supression de $f_{MinUsed}$. L'utilisation de f_{Max} est liée au choix des valeurs de bas en haut du domaine donc f_{Max} est une fréquence peu utilisée mais pas la moins utilisée, ainsi l'alternance apporte plus de diversité sur la méthode de résolution.

3.3.4.4. MinFreq(c)

Dans cette partie, nous proposons une approche un peu plus complexe basée sur une analyse de l'emploi de toutes les fréquences dans les solutions successives dans un contexte de type *generate and test* qui réalise une sorte d'apprentissage sur l'utilité des fréquences vis-à-vis de la réalisabilité, sans toutefois la prouver formellement. Cette stratégie apporte beaucoup plus de diversité sur la gestion du spectre.

La stratégie *MinFreq(c)* conçue pour optimiser l'objectif *MinFreq* repose sur la résolution d'une série de *CSP* et sur une mémorisation de *l'utilité* des fréquences dans deux listes nommées *ListeASupprimer* (fréquence pas utile) et *ListeANePasSupprimer* (fréquence utile).

Les deux listes sont initialisées à vide. *Tabu-NG* est utilisée pour trouver une solution réalisable ; une fois cette solution trouvée, on trie toutes les fréquences de cette solution par ordre croissant de leur taux d'utilisation dans la solution. La fréquence la moins utilisée, et qui n'appartient pas à la liste *ListeANePasSupprimer,* est supprimée de la solution courante, on la note $f_{MinSupp}$ (à noter qu'à la première itération de ce processus $f_{MinSupp}=f_{MinUsed}$). Puis on ajoute à la définition du problème une contrainte unaire temporaire sur toutes les variables, $f_i \# f_{MinSupp}$, interdisant l'utilisation de cette fréquence.

Tabu-NG est relancée pour *m* itérations (*m* constante à définir) sur ce nouveau problème ; si l'on obtient une solution réalisable, à ce stade cette fréquence n'est pas indispensable à la réalisabilité alors on l'ajoute à *ListeASupprimer* et on maintient les contraintes temporaires *;* si l'on n'obtient pas une solution réalisable, alors on ne peut pas conclure sur l'utilité de cette fréquence, on préfère la conserver : on l'ajoute à *ListeANePasSupprimer* et on supprime les contraintes temporaires $f_i \# f_{MinSupp}$. On relance le processus à partir de la procédure de tri en partant de la dernière solution réalisable trouvée pour identifier une nouvelle fréquence à tester.

Cette stratégie permet à la méthode de trouver ou de s'approcher d'un minimum local si *m* est assez grand. Si après avoir parcouru toutes les fréquences utilisées par la dernière solution trouvée cette stratégie n'arrive pas à améliorer le meilleure score obtenu, on vide les deux listes *ListeASupprimer* et *ListeANePasSupprimer* et on supprime une fréquence aléatoirement des domaines et de la solution courante, puis on relance *Tabu-NG* suivant le même processus.

3.3.5. Résultats et analyse

Nous avons implémenté la méthode *Tabu-NG* proposée en C++. Nos expérimentations ont été menées sur un PC Intel PentiumIV dont le processeur est cadencé à 2.4GHZ avec 1Go de RAM.

3.3.5.1. Comparaison des heuristiques de voisinage

Le Tableau 9 a pour but de comparer les deux heuristiques de voisinage sur le choix des variables au sein de la méthode *Tabu-NG*. Pour chaque instance, nous avons supprimé quelques fréquences afin de les rendre difficiles à résoudre et avoir une comparaison des heuristiques plus significative. Nous avons supprimé toutes les fréquences plus grandes que f_{Max} noté dans le Tableau 9 et nous avons testé *Tabu-NG* avec les deux heuristiques MRV et *dom/deg*. Les colonnes *NbItérations* donnent le nombre d'itérations réalisées par *Tabu-NG* avant d'atteindre une solution réalisable.

D'après ces tests, 9 instances sur 12 ont été résolues plus rapidement avec MRV. Globalement, la différence est très légère entre les deux heuristiques mais MRV est légèrement meilleure pour la méthode *Tabu-NG*. A noter que la méthode testée est totalement déterministe.

Dans les parties suivantes, l'heuristique MRV est exclusivement utilisée comme heuristique de voisinage.

Objectif = Chercher une solution réalisable					
Données				**MRV**	***dom/deg***
Problème	**nb var**	**nb ctr**	**f_{Max}**	**NbItérations**	**NbItérations**
CELAR01	916	5548	680	2812	3294
CELAR02	200	1235	394	1114	1296
CELAR03	400	2760	652	2162	2011
CELAR05	400	2598	792	1120	422
graph01	200	1134	408	3383	1023
graph02	400	2245	408	4527	5922
graph03	200	1134	380	1056	2039
graph04	400	2244	394	1136	1183
graph08	680	3757	680	2745	3218
graph09	914	5246	708	9425	10456
graph10	680	3907	394	2310	2337
graph14	916	4638	366	7191	10415

Tableau 9 : Comparaison des heuristiques de voisinage

3.3.5.2. Taille de la liste Tabou des nogoods Γ

Le Tableau 10 montre le nombre de *nogoods* stockés dans la liste Γ. Nous avons lancé *Tabu-NG* sur l'ensemble des instances avec l'objectif *MinSpec*. La colonne f_{Max} donne la fréquence la plus haute retournée par *Tabu-NG* et la colonne *NbNogood* donne le nombre de *nogood*s stockés dans Γ à la fin de l'optimisation.

Le nombre de *nogoods* stockés dans Γ est acceptable, il varie entre 5 et 2000. Une instance difficile à résoudre par *Tabu-NG* engendre un nombre de *nogoods* assez important. On voit qu'il n'y a pas de relation directe entre la taille du problème et la taille de cette liste à la fin de l'exécution ; le problème CELAR01 est l'un des plus gros en nombre de variables et de contraintes, et pourtant la liste de *nogood* à l'issue de la résolution contient seulement 145 ensembles de couples variable/valeur.

La notion de dominance entre les *nogoods* nous permet de bien gérer la taille de Γ. Pour les instances CELAR et graph du projet CALMA, nous n'avons pas rencontré de difficultés pour la gestion de la taille de Γ.

Objectif = *MinSpec*				
Données			***Tabu-NG* + MRV**	
Problème	**nb var**	**nb ctr**	**f_{Max}**	***NbNogood***
CELAR01	916	5548	680	145
CELAR02	200	1235	394	153
CELAR03	400	2760	652	182
CELAR05	400	2598	792	5
graph01	200	1134	408	1787
graph02	400	2245	394	1179
graph03	200	1134	380	50
graph04	400	2244	394	66
graph08	680	3757	652	1473
graph09	914	5246	666	776
graph10	680	3907	394	126
graph14	916	4638	352	1076

Tableau 10 : Taille de la liste Tabou des *nogoods*

3.3.5.3. Résultats sur les instances MinSpec

Nous donnons dans le Tableau 11 les résultats des 8 méthodes rapportés dans le paragraphe 3.2.3.1 sur 4 instances, les résultats de la méthode *CN-Tabu* [124] en première colonne, ainsi que nos propres résultats en dernière colonne.

Les colonnes du tableau indiquent le nom de chaque instance et la fréquence maximale trouvée par chaque méthode. A noter que le pas entre 2 fréquences consécutives est toujours 14. Les huit méthodes présentées sur la page Web du ZIB trouvent une solution optimale avec une fréquence maximale de 792 pour le CELAR05. Mais seulement 3 de ces 8 méthodes présentées, résolvent les instances GRAPH03, GRAPH04 et GRAPH10 et seulement 3 sur 8 résolvent chaque instance à l'optimalité. Les méthodes *CN-Tabu* et *Tabu-NG* résolvent aussi ces instances à l'optimalité.

Tabu-NG résout les 4 instances d'une façon optimale en moins de 10 secondes. Nous avons peu d'informations sur le contexte de résolution des méthodes au sein du projet CALMA : on ne sait pas sur quelles machines les tests ont été effectuées, ni le temps de calcul de chaque méthode, et on ne connaît pas non plus le taux de succès sur *n* exécutions. On retiendra donc seulement que notre méthode est au niveau des meilleures en termes de résolution avec l'objectif *MinSpec*.

Objectif = *MinSpec*										
	Vasquez	Aardal	Tiourine	Kolen	Tiourine	Boujou	Boujou	Warners	Pasechnik	**Dib**
Inst	CN-Tabu	Programmation linéaire	Programmation quadratique	PPC	Recherche Tabou	Recherche Tabou	Algorithme génétique	Réduction de potentiel	Réduction de potentiel	**Tabu-NG**
CELAR05	792	792	792	792	792	792	792	792	792	792
graph03	380	-	-	380	380	-	-	-	380	380
graph04	394	-	-	394	394	-	-	-	394	394
graph10	394	-	-	394	394	-	-	394	394	394
#inst résolues	**4**	1	1	**4**	4	1	1	2	**4**	**4**

Tableau 11 : Comparaison des résultats obtenus sur les instances *MinSpec* avec la littérature

Pour avoir une évaluation plus précise en taux de succès et temps de calcul, le Tableau 12 montre une comparaison entre les résultats de notre méthode avec des résultats publiés en 2005 avec *CN-Tabu* [124] qui est une des meilleures méthodes référencées actuellement sur ce type de problème.

Objectif = *MinSpec*								
Problème	**nb val**	**nb ctr**	**CN-Tabu**	**SR**	**T(s)**	***Tabu-NG***	**SR**	**T(s)**
CELAR01	916	5548	680	-	1	680	100%	1
CELAR02	200	1235	394	-	1	394	100%	1
CELAR03	400	2760	666	-	1	**652**	100%	1
CELAR04	680	3967	792	100%	1	792	100%	1
CELAR05	400	2598	792	100%	1	792	100%	1
graph01	200	1134	408	-	**1**	408	100%	7
graph02	400	2245	394	-	1	394	100%	1
graph03	200	1134	380	100%	1	380	100%	1
graph04	400	2244	394	40%	1	394	**100%**	1
graph08	680	3757	652	-	15	652	100%	**11**
graph09	916	5246	666	-	664	666	100%	**435**
graph10	680	3907	394	25%	17	394	**100%**	**10**
graph14	916	4638	352	-	30	352	100%	**22**
Nb inst. résolues			12			**13**		

Tableau 12 : Comparaison des résultats de *Tabu-NG* et *CN-Tabu* sur les instances *MinSpec*

La méthode *CN-Tabu* a été appliquée sur les instances du Tableau 10 pour *MinSpec* sur une machine AMD 64 3000 avec 1Go de RAM équivalent à un PentiumIV 3GHz. Ce tableau donne d'abord le nom de l'instance, le nombre de variables et de contraintes. Puis les colonnes 4 [CN-Tabu], 5 et 6 sont pour *CN-Tabu* et 7 [*Tabu-NG*], 8 et 9 pour notre méthode. Les colonnes [CN-Tabu] et [*Tabu-NG*] donnent le meilleur résultat (la fréquence maximale utilisée dans le spectre), les colonnes *T* donnent le temps d'exécution en secondes, et les colonnes SR (*Success Rate*) donnent le pourcentage de succès, autrement dit le nombre de fois où la solution optimale est atteinte sur 20 relances. Par exemple SR = 40% signifie que la méthode a réussi à trouver la valeur optimale 8 fois sur 20 lancements pour GR04 avec *CN-Tabu*. Parfois le taux de succès de *CN-Tabu* n'est pas publié pour certaines instances, dans ce cas nous indiquons «-».

Les résultats présentés dans le Tableau 12 montrent la compétitivité de *Tabu-NG*. Lorsque les résultats sont en faveur d'une méthode la cellule est grisée. *CN-Tabu* est ainsi plus performante sur graph01. La méthode *Tabu-NG* est plus performante à 6 reprises : sur CELAR03 en utilisant 1 fréquence de moins, sur graph04 et graph10 en ayant une réussite de 100% sur les 20 exécutions et sur graph08, graph09 et graph14 où elle est plus rapide. Globalement, *Tabu-NG* s'avère compétitive. Dans l'état actuel de nos travaux son apport principal réside dans son comportement déterministe pour *MinSpec* grâce auquel une seule exécution suffit à l'obtention de la meilleure solution trouvée par la méthode.

3.3.5.4. Résultats sur les instances MinFreq

Comparaison des stratégies proposées :

Le Tableau 13 présente les meilleurs résultats obtenus en 20 relances d'une heure, par les trois stratégies que nous avons retenues sur les instances *MinFreq*. Pour chaque

instance et chaque stratégie, nous avons : le nombre de fréquences utilisées par la meilleure solution suivi d'un astérisque s'il est optimal, le temps moyen en seconde nécessaire à l'obtention de la meilleure solution, ainsi que le nombre de fois où l'optimalité est atteinte sur 20 relances.

Le *Success Rate* est toujours à 20 (taux de succès de 100%) pour les stratégies déterministes *MinFreq(a)* et *MinFreq(b)* où la fréquence à supprimer est toujours la même ; rappelons que la méthode *Tabu-NG* dans son essence générale est elle aussi déterministe. Cependant la stratégie *MinFreq(c)* n'est pas déterministe parce qu'après un blocage, on redémarre cette stratégie en choisissant aléatoirement une fréquence à supprimer. Ceci étant ce choix pourrait aussi être rendu déterministe.

MinFreq(c) donne de très bons résultats : cette stratégie trouve la valeur optimale pour 9 des 10 instances. La stratégie *MinFreq(a)* donne des mauvais résultats, elle est généralement loin de l'optimum. Il n'est donc pas efficace de caler systématiquement le choix des fréquences à supprimer sur celles qui sont le moins utilisées. Dans *MinFreq(b)*, le fait de combiner les deux stratégies *MinFreq(a)* et *MinSpec* a permis d'améliorer les résultats de *MinFreq(a)*, surtout en utilisant *MinSpec* une fois sur deux seulement, mais en atteignant seulement 2 fois l'optimalité.

La meilleure stratégie est *MinFreq(c)* qui est basée sur la découverte des propriétés des fréquences vis-à-vis de la réalisabilité et sur un mécanisme de diversification intrinsèque du point de vue de la construction des solutions. Cette stratégie permet de tester les fréquences et le cas échéant de les réinsérer dans le domaine des variables lorsque la réalisabilité n'est pas atteinte. Elle se concentre sur les fréquences les moins utilisées dans les solutions trouvées, comme les autres stratégies, mais elle permet de garder certaines fréquences jugées intéressantes dans la structure des solutions.

Objectif : *MinFreq*												
Problème	***MinFreq* (c)**	T(s)	SR	***MinFreq* (a)**	T(s)	SR	***MinFreq* (b) :** *MinFreq*(a) = 2 *MinSpec* = 2	T(s)	SR	***MinFreq*(b) :** *MinFreq*(a) = 2 *MinSpec* = 1	T(s)	SR
CELAR01	16*	1700	20	30	3	20	30	3	20	22	7	20
CELAR02	14*	90	20	16	1	20	16	1	20	14*	2	20
CELAR03	14*	1502	17	16	3	20	18	2	20	18	2	20
CELAR04	46*	1	20	46*	1	20	46*	1	20	46*	1	20
CELAR11	22*	1605	9	42	2	20	40	8	20	40	8	20
graph01	18*	321	11	36	1	20	36	1	20	36	1	20
graph02	14*	517	16	14*	34	20	20	15	20	16	53	20
graph08	18	820	5	36	490	20	32	654	20	30	712	20
graph09	18*	168	4	26	13	20	24	43	20	22	78	20
graph14	8*	627	14	10	18	20	12	15	20	10	39	20

Tableau 13 : Comparaison des différentes stratégies sur les instances *MinFreq*

Comparaison avec la littérature :

Le Tableau 14 dresse une comparaison entre *Tabu-NG* et les meilleures méthodes présentées sur la page Web du ZIB. *Tabu-NG* trouve la valeur optimale de 9 instances sur 10, elle est au même niveau que les meilleures méthodes connues dans la littérature pour l'objectif *MinFreq*.

Problème	Optimum	Tiourine Clique Bound	Kolen PPC	Aardal Relaxation linéaire	Pasechnik Descente de gradient	Tiourine Recherche Tabou	Bouju Recherche Tabou	Warners Réduction de potentiel	Bouju Algorithme Génétique	Dupont CN-Tabu	Dib Tabu-NG
CELAR01	16	**16**	**16**	**16**	**16**	**16**	**18**	**16**	**16**	**16**	**16**
CELAR02	14	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
CELAR03	14	**14**	**14**	**14**	**14**	**14**	**14**	16	**14**	**14**	**14**
CELAR04	46	**46**	**46**	**46**	**46**	**46**	**46**	**46**	**46**	**46**	**46**
CELAR11	22	**22**	**22**	**22**	**22**	**22**	24	-	24	**22**	**22**
graph01	18	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
graph02	14	**14**	**14**	**14**	**14**	**14**	16	**14**	**14**	**14**	**14**
graph08	16-18	-	-	-	18	20	24	18	22	22	18
graph09	18	-	**18**	-	**18**	22	22	**18**	22	**18**	**18**
graph14	8	-	10	-	**8**	10	12	10	-	**8**	**8**
#inst résolues		7	8	7	9	7	4	6	6	9	9

Tableau 14 : Comparaison des résultats obtenus avec la littérature sur *MinFreq*

Comparaison des temps d'exécution :

Pour avoir une évaluation plus précise en taux de succès et temps de calcul, le Tableau 15 montre une comparaison entre les résultats de *Tabu-NG* avec les résultats de *CN-Tabu*. Les 2 méthodes trouvent la valeur optimale pour le même nombre d'instances, cependant *Tabu-NG* améliore le problème *graph08* de 4 fréquences. Au niveau du taux de succès, il y a une amélioration sur 4 instances et une dégradation sur 3 instances pour un même nombre de fréquences. Finalement en terme de temps de calcul *CN-Tabu* est plus rapide à 6 reprises tandis que *Tabu-NG* est meilleure à 4 reprises y compris sur *graph08*.

	Dupont			**Dib**		
Problème	**CN-Tabu**	**T(s)**	**SR**	***Tabu-NG***	**T(s)**	**SR**
CELAR01	16*	**221**	20	16*	1700	20
CELAR02	14*	**14**	15	14*	90	**20**
CELAR03	14*	1890	10	14*	**1502**	**17**
CELAR04	46*	**0**	20	46*	1	20
CELAR11	22*	1993	2	22*	**1605**	**9**
graph01	18*	1981	7	18*	**321**	**11**
graph02	14*	**1**	**20**	14*	517	16
graph08	22	2324	0	**18**	**820**	**5**
graph09	18*	**8**	**19**	18*	168	4
graph14	8*	**20**	**20**	8*	627	14
Nb inst. résolues	9			9		

Tableau 15 : Comparaison des résultats de *Tabu-NG* et *CN-Tabu* sur les instances *MinFreq*

3.3.6. Discussions

Tabu-NG a montré son efficacité et sa robustesse pour résoudre des problèmes de satisfaction de contraintes parmi les plus expérimentés. Les résultats obtenus sur les problèmes d'affectation de fréquences CALMA sont au niveau des meilleurs résultats connus dans la littérature. Dans cette section, nous avons traité les instances réalisables par raison d'économie de temps puisque notre objectif à moyen terme était de traiter le problème d'affectation de fréquences avec polarisation et contraintes n-aires. Cependant il nous a permis de tester les fondations de la méthode en se comparant à d'autres méthodes sur des benchmarks.

Tabu-NG manipule des configurations partielles mais elle peut se généraliser pour traiter des *Max-CSP* en minimisant le nombre de contraintes violées, une somme pondérée de contraintes violées, etc. Plusieurs possibilités sont envisageables, le plus simple serait d'étendre la meilleure configuration partielle trouvée par *Tabu-NG* et la meilleure solution, dans ce cas, serait la plus grande solution en termes de nombre de variables instanciées. Nous allons traiter des problèmes similaires dans la section suivante et nous verrons donc plus largement les utilisations possibles de cette méthode.

Parmi les points forts de la méthode, nous pouvons souligner que la résolution est relativement indépendante des configurations initiales. *Tabu-NG* commence la résolution avec une configuration partielle vide, mais elle utilise aussi des configurations partielles consistantes comme c'est le cas dans les stratégies d'optimisation pour les critères *MinSpec* et *MinFreq*. Nous devons souligner aussi son aspect déterministe qui en fait une méthode de confiance et facile à utiliser puisqu'une seule exécution suffit.

Parmi les points faibles de la méthode, la complexité spatiale pour identifier les *nogood*s pendant la recherche est assez importante. Pour des CSP de très grande taille, il est impossible de stocker toutes les informations nécessaires pour identifier facilement les *nogoods* pendant la recherche. La solution la plus simple serait de chercher en ligne, pendant le déroulement de la recherche, les supports d'une valeur lorsqu'on en a besoin. Cependant cette solution est très couteuse en termes de temps d'exécution. La propagation des contraintes n-aires est aussi très coûteuse, la complexité temporelle est très importante. Il faut donc gérer différemment les contraintes n-aires, nous verrons par la suite une solution originale.

Dans la section suivante, nous allons présenter comment traiter des CSP de très grandes tailles avec la méthode *Tabu-NG* à travers le problème d'affectation de fréquences avec polarisation et contraintes n-aires.

3.4. Méthode *Tabu-NG* pour AFDP avec contraintes n-aires

3.4.1. Algorithme de filtrage et *nogood* partiel

3.4.1.1. Nogoods partiels

Justification de la relaxation de l'exactitude des *nogoods* :

Tabu-NG est une méthode de recherche dirigée par les *nogoods*. Pour les problèmes de petite et moyenne taille comme nous venons de le voir sur les problèmes ROADEF, la taille de la structure qui contient les supports des valeurs qui permet l'identification des *nogoods* pendant la recherche, reste acceptable. Mais face à des problèmes de très

grandes tailles, il n'est pas possible de stocker toutes les informations requises avec un temps d'accès rapide à ces informations. Cela doit se faire soit au détriment de la vitesse de l'algorithme soit au détriment de la précision des *nogoods*.

Pour traiter ce cas de figure nous avons proposé une autre méthode de gestion des *nogoods*. Un *nogood* même s'il n'est pas exact peut être capable de diriger la recherche en contribuant à l'optimisation d'un problème dans un contexte de solutions réalisables par rapport à des contraintes à satisfaire. Nous introduisons pour cela la notion de *nogood partiel* qui enlève la part exacte d'explication d'un *nogood* sur l'identification d'un *deadend*. En ce sens nous relâchons leur exactitude pour les traiter avec un certain degré d'incertitude sur leur effet dans l'évitement des *deadends*.

Nous définissons un *nogood partiel* comme une partie d'un *nogood* autrement dit un sous-ensemble de *nogood*. Ceci induit qu'un *nogood* partiel est une configuration partielle qui pourra faire partie, ou non, d'une solution réalisable. Etant donné qu'un *nogood* partiel *ngp* est inclut nécessairement dans un *nogood ng* selon notre définition alors la probabilité que *ngp* fasse partie d'une solution réalisable dépendra de la précision de celui-ci, ou de la part d'information qu'il contient, autrement dit de la différence de taille entre *ngp* et *ng*.

Prenons un exemple, imaginons un *nogood ng* qui contient 6 affectations, et un *nogood* partiel *ngp* qui contient 4 affectations parmi celles-ci, soit $ngp \subset ng$, la probabilité que *ng* fasse partie d'une solution réalisable est de 0% alors que la probabilité que *ngp* fasse partie d'une solution réalisable est : (taille de *ng* – taille de *ngp)/*taille de *ng,* soit 33,33 % pour cet exemple. Si la taille du *ng* est égale à la taille du *ngp* avec $ngp \subset ng$ alors *ngp* est exactement un *nogood* et il n'y a pas relâchement.

Dans la suite de nos travaux nous nous ne sommes pas intéressés à l'affectation d'une probabilité aux *nogoods* partiels identifiés bien que cela soit possible. Notre but est simplement d'identifier ces *nogoods* et de les utiliser afin de diriger la recherche dans des zones prometteuses de l'arbre de recherche, ou en tout cas dont le risque de conduire à un *deadend* est limité, sans être exclus. Ainsi nous limitons la complexité de gestion de ces informations.

Dans ce nouveau contexte *Tabu-NG* garde son schéma général. La seule composante qui change est l'algorithme de filtrage utilisé afin d'identifier les *nogoods* partiels. Dans les AFD traités dans la section précédente, nous avons utilisé deux algorithmes de filtrage, l'un après l'extension d'une configuration partielle, et l'autre après la réparation d'une configuration partielle. Dans cette section, nous allons toujours utiliser les deux types d'algorithme en les modifiant.

Comment chercher un *nogood* partiel ?

Dans la section 3.3.2 nous avons expliqué comment identifier un *nogood* pendant la recherche ; il faut associer à chaque valeur *a* dans le domaine d'une variable *i* un ensemble d'affectation *ElExpl* qui contient l'explication d'élimination de cette valeur. L'explication d'élimination de la suppression d'une valeur *a* dans le domaine d'une variable *i* n'est pas seulement la dernière affectation ajoutée à la configuration partielle courante mais l'union de toutes les explications d'éliminations de chaque support de la valeur supprimée *a* dans le domaine de la variable opposée *j* selon une contrainte donnée c_{ij}. Lorsqu'un *deadend* a lieu, le *nogood* identifié est l'union de toutes les explications

d'éliminations des valeurs supprimées dans le domaine de la variable vide, ce qui se traduit par :

$$\bigcup_{a \in Dom_v} ElExpl(i,a).$$

Nous avons vu que l'identification d'un *nogood* nécessite l'utilisation d'une structure qui stocke pour chaque contrainte *c* du problème tous les supports de toutes les valeurs. Pour des problèmes de grandes tailles, il est difficile, voire impossible, de stocker une telle structure. La solution intuitive et la plus simple est de calculer en ligne les supports d'une valeur lorsque c'est nécessaire. Cependant, cette solution est très coûteuse, et la recherche deviendra très lente ; ce calcul peut s'avérer la tâche la plus gourmande de l'algorithme, ce qui est totalement inefficace.

A ce stade, nous décidons de tenter d'éliminer la structure nécessaire pour le stockage des supports, et d'associer une explication d'élimination à chaque variable au lieu de l'associer à chaque valeur. L'explication d'élimination d'une variable sera mise à jour lorsqu'une valeur dans son domaine sera supprimée par un algorithme de filtrage. La mise à jour se fera de la façon suivante :

$$ElExpl(i) = ElExpl(i) \bigcup DerniereAffectation$$

Où *ElExpl*(*i*) est l'explication d'élimination de la variable *i* et *DerniereAffectation* est la dernière instanciation ajoutée à la configuration partielle courante.

De cette façon, si une valeur dans le domaine d'une variable *i* est supprimée, on ajoute la dernière affectation réalisée à la configuration partielle courante à l'explication d'élimination de la variable *i*. Si un *deadend* se produit, le *nogood* probabiliste sera l'explication d'élimination de la variable dont le domaine est devenu vide.

Il s'agit d'une information partielle, incomplète pour conclure définitivement sur l'explication. Ainsi, le *nogood* probabiliste généré est un *nogood* inexact, il pourra faire partie, ou non, d'un véritable *nogood*. Les *nogoods* probabilistes identifiés durant la recherche seront stockés dans la liste Γ, comme précédemment, et *Tabu-NG* fonctionne toujours selon son schéma général décrit jusqu'ici. La diversification de l'algorithme agira toujours mais selon un processus décisionnaire imparfait, bien que toujours déterministe.

A noter que, la suppression de la structure de stockage précédente implique des changements non seulement dans l'algorithme de filtrage après une extension d'une configuration partielle mais aussi dans l'algorithme de filtrage utilisé après une réparation d'une configuration partielle. Nous allons décrire ces algorithmes dans les parties suivantes.

3.4.1.2. Algorithme de filtrage après extension

Dans cette partie, nous avons choisi un algorithme simple à implémenter qui possède des complexités spatiales et temporelles acceptables. Nous avions le choix entre des algorithmes de type AC-3, AC-6 ou AC-2001/3.1. Les autres algorithmes proposés dans la littérature possèdent une complexité spatiale assez importante et ne sont pas recommandés pour notre proposition.

Nous avons choisi d'implémenter un algorithme de filtrage basé sur AC-3, le plus simple à implémenter, et le plus simple à modifier afin d'introduire les inférences des contraintes (i.e. tester la consistance d'un arc selon la nature mathématique de la contrainte) afin de réduire la complexité temporelle (cf. section 3.3.1.2). Prenons un exemple : pour une contrainte de différence (*i#j*), et pour tester si une valeur *a* dans le domaine de la variable *i* possède un support dans le domaine de la variable *j,* il suffit de tester si le nombre de valeurs dans le domaine de la variable opposée est plus grand que 1.

L'algorithme de filtrage donné dans Algorithme 20 décrit comment nous procédons pour filtrer les domaines avec des contraintes binaires et n-aires, et en même temps identifier les *nogoods* probabilistes en cas de *deadend*. L'idée est très simple, une fois que la configuration partielle courante *S* est étendue et qu'une nouvelle affectation (*i, a*) est ajoutée à *S*, on propage l'effet de cette affectation sur les variables voisines de *i*, puis on établit l'arc-consistance ainsi que la consistance du chemin sur le reste du graphe, selon les règles données en 3.4.2.

L'Algorithme 20 propage l'effet de l'ajout de la nouvelle affectation (*i*,*a*) à la configuration courante *S* en appelant la fonction *checkFC* qui fait la propagation sur les variables voisines de la variable *i*. Cette fonction retourne un *nogood* si un *deadend* est atteint (ligne 1 de l'algorithme) ; elle met aussi dans une liste *Q* toutes les variables qui ont subi une modification dans leur domaine. *Q* est nécessaire afin de propager l'effet des modifications faites sur le reste du graphe. Ensuite, on parcourt la liste *Q* (ligne 3) en appelant la fonction *checkAC* sur chaque variable impactée (ligne 5) pour propager la modification de son domaine sur le reste du graphe. La fonction *checkAC* retourne un *nogood* lorsqu'un *deadend* a lieu.

```
Function AlgoFiltrageApresExtension (D, S, C, i, a)
1.      ngp = checkFC(i, a, Q)
2.      if ngp is not NULL return ngp;
3.      while Q is not Empty do
4.              varAC = Q.pop();
5.              ngp = checkAC(varAC, i, a, Q) ;
6.              if ngp is not NULL return ngp ;
7.      endwhile;
```

Algorithme 20 : Filtrage et propagation des contraintes binaires et n-aires après extension d'une configuration partielle courante

La fonction *checkFC* décrite dans l'Algorithme 21 propage l'effet de l'ajout d'une nouvelle instanciation sur les variables voisines de la dernière variable affectée. Cette fonction parcourt toutes les contraintes de la dernière variable instanciée *i* et propage chaque contrainte selon son type : binaire (ligne 3), intermodulation (ligne 15) et sommation de perturbateurs (ligne 18). Pour une contrainte binaire, on parcourt le domaine de la variable libre opposée *otherVar* (ligne 6) et on supprime toutes les valeurs non consistantes avec l'instanciation de *i* (lignes 7 et 8). Si une modification dans le domaine de *otherVar* a lieu, l'explication d'élimination de *otherVar* est mise à jour (ligne 9) et *otherVar* est ajoutée à une liste Q afin de propager ultérieurement la modification de son domaine sur le reste du graphe (ligne 11). On fait la même chose avec les autres types de contraintes en appelant les fonctions *CheckIntemodulation* (ligne 16) et *ChekSP* (ligne 19).

```
Function checkFC (i, a, Q)
1.    ctrList = All constraints of i
2.    foreach c in ctrList do
3.      if c is binary constraint then
4.          otherVar = the 2nd variable of the constraint c;
5.          if otherVar is not assigned then
6.              foreach v_o in D_otherVar do
7.                if not c.verify(a, v_o) then
8.                    Remove v_o from D_otherVar;
9.                    ElExpl(otherVar) += (i, a);
10.                   if D_otherVar is Empty return ElExpl (otherVar) ;//deadend
11.                   Q.add(otherVar);
12.               endif
13.             endforeach
14.         endif ;
15.     else if c is intermodulation then
16.             ngp = CheckIntermodulation(i, a, Q, c) ;
17.             if ngp is not NULL return ngp;
18.         else if c is sommation de perturbateurs then
19.             ngp = ChekSP(i, a, Q, c);
20.             if ngp is not NULL return ngp;
21.         endif;
22.     endif;
23.   endif;
24.   return NULL;
```

Algorithme 21 : Propagation de l'effet de l'ajout d'une nouvelle affectation à une configuration partielle sur les variables voisines

Pour une contrainte de type intermodulation qui porte sur 2 variables brouilleuses, l'Algorithme 22 propage l'effet de l'ajout de l'instanciation de *i* sur la seule variable libre de cette contrainte (ligne 2) en supprimant de son domaine toutes les valeurs inconsistantes (lignes 3 et 4). Si une modification a lieu, l'explication d'élimination de la variable libre *unassignedVar* est mise à jour (ligne 5) et cette variable est ajoutée à la liste Q (ligne 9). Si un *deadend* est détecté, un *nogood* est identifié (lignes 6 et 7).

```
Function checkIntermodulation (i, a, Q, c)
1.    if All vars of c are instanciated except one then
2.        foreach v_o in D_UnassignedVar do
3.            if not c.verify(v_o) then
4.                Remove v_o from D_UnassignedVar
5.                ElExpl(UnassignedVar) += (i, a);
6.                if D_unassignedVar is Empty then
7.                    return ElExpl (unassignedVar) ;//deadend
8.                endif;
9.                Q.add(UnassignedVar);
10.           endif;
11.       endforeach;
12.   endif ;
13.   return NULL;
```

Algorithme 22 : Filtrage des domaines par une contrainte d'intermodulation

La fonction *checkSP* traite tous les cas évoqués dans la section 3.4.2.4. La fonction telle qu'elle est donnée dans l'Algorithme 23 décrit un seul de ces cas, le même que celui traité par la fonction *checkIntermodulation*.

```
Function checkSP (i, a, Q, c)
1.    if not c.satisfy(i,a) then
2.        ng = AllAffectedCoupleOf(c);
3.        return ng;
4.    endif;
5.
6.    if All vars of c are instanciated except one then
7.        foreach v_o in D_UnassignedVar do
8.            if not c.verify(v_o) then
9.                Remove v_o from D_UnassignedVar
10.               ElExpl(UnassignedVar) += (i, a);
11.               if D_otherVar is Empty then
12.                   return ElExpl(otherVar);//deadend
13.               endif;
14.               Q.add(UnassignedVar);
15.           endif;
16.       endforeach;
17.   endif ;
18.   return NULL;
```

Algorithme 23 : Filtrage des domaines par une contrainte de sommation de perturbateurs

L'Algorithme 24 décrit la fonction *checkAC*. Cette fonction établit l'arc consistance pour toutes les contraintes binaires de la variable libre *varAC* (lignes 3 à 14). Elle fait appel aux fonctions *CheckIntermodulation* et *checkSP* pour traiter les contraintes d'intermodulation et les contraintes de sommation de perturbateurs (lignes 25 à 32).

```
Function checkAC (varAC, i, a, Q)
1.    ctrList = All constraints of varAC
2.    foreach c in ctrList do
3.      if c is binary constraint then
4.        otherVar = the 2nd variable of constraint c;
5.        if otherVar is not assigned then
6.            foreach v_o in D_otherVar
7.                if no support for v_o in D_varAC then
8.                  Remove v_o from D_otherVar;
9.                  ElExpl(otherVar) += (i, a);
10.                 if D_otherVar is Empty return ElExpl(otherVar);//deadend
11.                 Q.add(otherVar);
12.               endif
13.           endforeach
14.       endif
25.     else if c is intermodulation then
26.           ngp = CheckIntermodulation(i, a, Q, c) ;
27.           if ngp is not NULL return ngp;
```

```
28.          else if c is sommation de perturbateurs then
29.                ngp = ChekSP(i, a, Q, c);
30.                if ngp is not NULL return ngp;
31.          endif;
32.       endif;
33.    endif;
15.    endforeach
16.    return NULL ;
```

Algorithme 24 : Filtrage des domaines par une contrainte binaire ou n-aire

3.4.1.3. Algorithme de filtrage après réparation

Après la réparation d'une configuration partielle, il faut mettre à jour les domaines des variables. L'annulation d'une affectation dans la configuration partielle courante induit parfois (souvent) la réinsertion de certaines valeurs dans des domaines de certaines variables. Avec l'algorithme de filtrage utilisé, cette phase nécessite la restauration de tous les domaines initiaux des variables libres et le lancement d'un nouveau filtrage depuis le début.

L'algorithme de filtrage après réparation est donné dans Algorithme 25. L'algorithme initialise les domaines des variables libres à leurs états initiaux (lignes 2 à 4). Puis, il prend les variables affectées (ligne 6) et propage l'effet de leur instanciation sur le reste du graphe (ligne 7). La propagation de l'effet de chaque instanciation se fait sur les variables voisines (ligne 9) puis sur le reste du graphe (lignes 12 à 15). Dans le cas où un *deadend* est détecté, l'algorithme retourne le *nogood* responsable de l'échec.

```
Function AlgoFiltrageApresReparation(D, S, C, i, a, ng, Δ)
1.    //Mise à jour des domaines des variables libres
2.    foreach unassigned variable var do
3.          D_var=DomInitial_var
4.    endforeach
5.
6.    FutureVarsFC = Les variables instanciées dans S
7.    while FutureVarsFC is not Empty do
8.          varFC = FutureVarsFC.pop()
9.          val = valeur affectée à varFC
10.         ngp = checkFC(varFC, val, Q); //Propagation sur les variables voisines
11.         if ngp is not NULL return ngp ;
12.         //Propagation sur le reste du graphe
13.         while Q is not Empty do
14.               varAC = Q.pop();
15.               ngp = checkAC(varAC, varFC, val, Q) ;
16.               if ngp is not NULL return ngp ;
17.         endwhile;
18.   endwhile ;
```

Algorithme 25 : Filtrage des domaines après réparation d'une configuration partielle inconsistante

3.4.2. Traitement des contraintes n-aires

3.4.2.1. Contraintes d'intermodulation

La propagation des contraintes n-aires de type intermodulation portant sur 3 variables se fait de la façon suivante : une fois 2 variables instanciées sur cette contrainte, on propage l'effet de ces instanciations sur la troisième variable ; si le domaine de la troisième variable devient vide alors la contrainte ne pourra pas être satisfaite et un *deadend* est atteint. Le *nogood* est égal aux deux variables déjà affectées de la contrainte.

3.4.2.2. Contraintes de sommation de perturbateurs

Pour les contraintes n-aires de type sommation de perturbateurs qui portent sur plusieurs variables, deux phases sont nécessaires : renforcement des contraintes n-aires et propagation de ces contraintes pendant la recherche. Nous expliquons ces deux phases ci-après.

3.4.2.3. Renforcement des contraintes n-aires

Principe du renforcement :

Le renforcement d'une contrainte n-aire ne consiste pas en son remplacement [128] mais à l'écriture de contraintes plus simples, issues de la contrainte n-aire, qui viennent s'ajouter à la description du problème. Pour être en phase ces nouvelles contraintes ne doivent pas sur-contraindre le problème par rapport à la contrainte n-aire, mais simplement restreindre le champ des valeurs possibles pour les variables déclarées dans la contrainte n-aire. La contrainte n-aire est donc renforcée par l'ajout de sous-contraintes qu'elle contient et qui seront exploitées pour en faciliter la satisfaction.

Pour ce faire une phase de renforcement des contraintes de sommation de perturbateurs afin de calculer de nouvelles contraintes, binaires, plus simples, est lancée avant le début de la recherche d'une solution. Les nouvelles contraintes binaires seront traitées comme des contraintes binaires du problème et aideront à satisfaire l'ensemble des contraintes de type sommation de perturbateurs.

Le renforcement est réalisé comme suit. Soit la contrainte de type sommation de perturbateurs suivante :

$$\sum_i \beta_{i\rho} T_{i\rho}\left(\left|f_i - f_\rho\right|\right) \leq B_\rho$$

C'est une contrainte de sommation de perturbateurs électromagnétiques. f_i est la fréquence brouilleuse affectée à la variable i et f_ρ la fréquence porteuse affectée à la variable ρ. $\beta_{i\rho}$ est un coefficient qui lie les variables i et ρ. B_ρ est un seuil à ne pas dépasser par la contrainte n-aire. T est une fonction qui calcule le brouillage selon l'écart en fréquences, c'est une fonction monotone décroissante (Figure 18).

Comme $\beta_{i\rho}$ et $T_{i\rho}$ sont positifs non nuls, il faut que tous les perturbateurs i respectent l'équation :

$$\beta_{i\rho} T_{i\rho}\left(\left|f_i - f_\rho\right|\right) \leq B_\rho$$

Soit une contrainte binaire déduite : $\left|f_i - f_\rho\right| \geq T_{i\rho}^{-1}\left(\frac{B_\rho}{\beta_{i\rho}}\right)$

Comme la fonction $T_{i\rho}$ est paire et décroissante dans le domaine positif, elle est inversible et on aura une seule valeur d'écart possible à respecter.

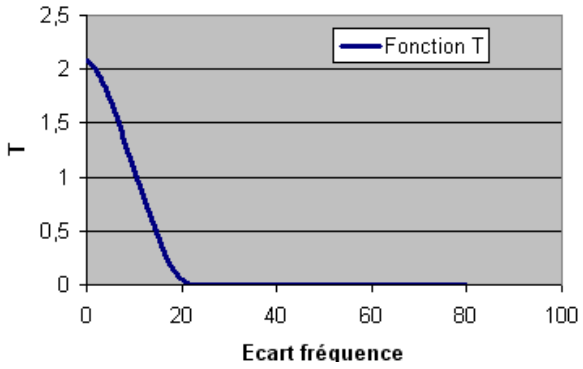


Figure 18 : Courbe représentative de la fonction *T*

Exemple de contrainte renforcée :

Soit une contrainte n-aire qui porte sur 6 variables, une variable p et 5 autres variables perturbatrices avec :

$\beta_{i\rho}$ = [47.58393 ; 37.79726 ; 39.57859 ; 49.82649 ; 31438.39] pour les 5 perturbateurs et B_ρ = 1.24

Si l'on calcule $\beta_{i\rho}T_{i\rho}(x)$ et que l'on compare par rapport aux seuils, on en déduit les contraintes binaires renforcées : {|f0-f1| >=21, |f0-f2| >=21, |f0-f3| >=21, |f0-f4| >=21, |f0-f5| >=80}.

Il faut noter que les contraintes binaires déduites à partir d'une contrainte n-aire ne sont pas des contraintes équivalentes. Les contraintes n-aires originales sont donc à conserver dans la définition du problème et forcément à gérer en termes de vérification par l'algorithme de recherche. Les contraintes binaires ajoutées seront utilisées comme accélérateur de recherche lors de la phase de propagation.

3.4.2.4. Propagation pendant la recherche

La contrainte de sommation de perturbateurs s'écrit sous la forme : $\sum_{i\in n}\beta_{i\rho}T_{i\rho}\left(\left|f_i - f_\rho\right|\right) \leq B_\rho$ où $T_{i\rho}$ sont des fonctions décroissantes positives. Supposons que la variable de réception v_ρ soit instanciée et que les variables perturbatrices $v_1, v_2, ..., v_k$ le soient aussi. Supposons que les variables perturbatrices $v_{k+1}, v_{k+2}, ..., v_n$ ne soient pas encore instanciées. On traite les cas de figure successifs suivants :

- Si $\sum_{i=1}^{k}\beta_{i\rho}T_{i\rho}\left(\left|f_i - f_\rho\right|\right) > \beta_\rho$ alors la contrainte ne peut pas être satisfaite, par conséquent le domaine des variables non instanciées se réduit à vide. Par exemple, si une contrainte porte sur 10 variables et qu'après l'instanciation de 3

variables (dont forcément v_ρ) on s'aperçoit que la contrainte est violée alors on peut conclure que l'affectation actuelle de ces 3 variables ne pourra pas nous conduire à une satisfaction totale de la contrainte. Un *nogood* est repéré, c'est l'ensemble d'affectation des 3 variables pour cette contrainte. Plus généralement le *nogood* est composé de v_ρ et des k variables déjà affectées.

- Soit $d_{i\rho}^{\max}$ l'estimation de l'écart maximal entre les fréquences de la variable de réception v_ρ déjà instanciée et de la variable perturbatrice non instanciée v_i avec $i>k$. Cette valeur est calculée à partir de l'écart maximal entre la plus petite ou la plus grande valeur du domaine filtré de v_i et la fréquence affectée à v_ρ. Si $\sum_{i=1}^{k} \beta_{i\rho} T_{i\rho} \left(\left| f_i - f_\rho \right| \right) + \sum_{i=k+1}^{n} \beta_{i\rho} T_{i\rho} \left(d_{i\rho}^{\max} \right) > B_\rho$ alors la contrainte ne peut pas être satisfaite, par conséquent le domaine des variables non instanciées se réduit à vide. De la même manière un *nogood* est repéré, c'est l'ensemble composé de v_ρ, des k variables affectées pour cette contrainte et de l'explication d'élimination de toutes les variables libres de la contrainte.
- Pour chaque variable non instanciée v_b, les fréquences f_b vérifiant la condition $\sum_{i=1}^{k} \beta_{i\rho} T_{i\rho} \left(\left| f_i - f_\rho \right| \right) + \sum_{\substack{i=k+1 \\ i \neq b}}^{n} \beta_{i\rho} T_{i\rho} \left(d_{i\rho}^{\max} \right) + \beta_{b\rho} T_{b\rho} \left(\left| f_b - f_\rho \right| \right) > B_\rho$ suivante sont à filtrer du domaine de la variable v_b.

Au final, une contrainte de type sommation de perturbateur est propagée s'il ne reste qu'une seule variable à affecter dans cette contrainte, l'effet de l'instanciation de toutes les autres variables sera propagé sur le domaine de la variable non encore instanciée. Si le domaine de cette variable devient vide alors on est dans un *deadend* et un *nogood* est repéré qui est l'ensemble des variables instanciées de la contrainte.

3.4.3. Paramètres de la méthode

Cette partie décrit les paramètres utilisés par la méthode *Tabu-NG*, pour résoudre les AFDP avec contraintes n-aires fournis par le CELAR.

3.4.3.1. Taille de la liste Tabou des nogoods Γ

Etant donné que les *nogoods* identifiés sont des *nogoods* probabilistes, sans certitude sur leurs effets quant à l'orientation de la recherche, il est donc inutile de demander à la méthode *Tabu-NG* de stocker tous les *nogoods* identifiés pendant la recherche d'une façon permanente. Nous avons fixé la taille de Γ à 15 000. Cette liste se comporte comme un FIFO (*First In, First Out*).

3.4.3.2. Durée Tabou utilisée pour Δ

Comme précédemment, la réparation d'une solution consiste à annuler des affectations déjà faites et qui font partie de la configuration partielle courante. Une

affectation est considérée Tabou pour une durée adaptative qui est égale au nombre de fois où cette affectation a été visitée par la méthode. Cette durée est donnée par :

$$\Delta(i,a) = iterationCourante + nbreMvt(i,a)$$

3.4.3.3. Heuristique de choix du voisinage

Suite aux résultats obtenus dans la section précédente, nous avons directement utilisé l'heuristique MRV (*Minimum Remaining Values*) pour le choix des variables à instancier.

3.4.4. Stratégies d'optimisation

3.4.4.1. Recherche de solution réalisable

Dans ce mode de fonctionnement, *Tabu-NG* recherche une solution réalisable, respectant toutes les contraintes, unaires, binaires et n-aires, quel que soit leur niveau de repli. La recherche d'une solution réalisable se fait en lançant *Tabu-NG* et en utilisant les algorithmes de filtrage décrits dans les paragraphes précédents. Le résultat est une solution réalisable qui satisfait toutes les contraintes, ou bien une configuration partielle consistante si le temps imparti est écoulé, ou bien une preuve de non réalisabilité du problème avec un arrêt du programme avant la fin du temps préfixé.

Afin de permettre à la méthode de diversifier sa recherche, nous avons adopté une stratégie de redémarrage de la recherche (principe du *restart* utilisé par d'autres méthodes). Si la méthode n'arrive pas à avancer au bout d'un nombre donné d'itérations (500 + 3**nombre de variables*), on recommence la recherche avec une configuration partielle vide, tout en supprimant les durées Tabou Δ ainsi que les poids des affectations calculées, mais en gardant tous les *nogoods* stockés. De cette façon *Tabu-NG* diversifie sa recherche en s'orientant vers des zones non encore visitées. On voit à nouveau l'intérêt de l'apprentissage de la méthode sur le problème à travers l'utilisation des *nogoods*.

Etant donné que les *nogoods* identifiés sont des *nogoods* partiels, *Tabu-NG* peut donc éliminer des branches utiles pendant son déroulement et par la suite ne pas pouvoir avancer dans la recherche d'une solution réalisable. De même, les éventuelles preuves de non réalisabilité du problème ne sont plus valables dans ce contexte. Il faut donc aussi pouvoir reconsidérer la liste des *nogoods* établis. Chaque fois qu'une preuve d'inconsistance est donnée par la méthode, on recommence la recherche avec une configuration partielle vide, tout en supprimant les durées Tabou *Δ*, les poids des affectations calculées et en vidant la liste Tabu des *nogoods*. En cas de stagnation cette stratégie permet à la méthode d'avancer sur un chemin qui est marqué comme *nogood* mais sans exactitude de ce marquage. Cela libère ainsi des espaces de recherche pour un nouveau départ de la méthode. Après 6 appels de cette stratégie, *Tabu-NG* arrête la recherche et considère que le problème est irréalisable.

L'inclusion de ces stratégies de redémarrage nous ont permis d'améliorer des résultats obtenus par *Tabu-NG* sur quelques instances fournies par le CELAR, nous donnerons par la suite des résultats concrets.

Nous voyons maintenant les différentes stratégies pour l'optimisation d'un critère en sus de la satisfaction des contraintes que nous venons d'énoncer.

3.4.4.2. MaxFreqG

Pour un niveau de relâchement unaire l et un niveau de relâchement non unaire k, ce mode vise à trouver une solution réalisable qui maximise la fréquence basse du spectre utilisée par les variables de la cible.

La stratégie d'optimisation *MaxFreqG* se base sur la notion de réalisabilité : on lance la méthode de base qui cherche une solution *(l,k)*-réalisable et une fois la réalisabilité atteinte, on cherche la fréquence la plus basse f_{min} utilisée dans cette solution. On ajoute une contrainte d'interdiction sur toutes les variables, x_i $\#[0,\ldots,f_{min}]$, on désaffecte toutes les variables utilisant cette fréquence dans la solution actuelle pour obtenir une configuration partielle S et on relance la méthode de base de recherche de réalisabilité avec comme point de départ la dernière configuration partielle S. Ce processus est répété plusieurs fois jusqu'à ce que la preuve de non réalisabilité soit atteinte ou que le temps maximum donné soit atteint.

3.4.4.3. MinFreqD

Pour un niveau de relâchement unaire l et un niveau de relâchement non unaire k, ce mode vise à trouver une solution réalisable qui minimise la fréquence haute du spectre utilisée par les variables de la cible.

La stratégie d'optimisation *MinFreqD* se base sur la notion de réalisabilité : on lance la méthode de base qui cherche une solution *(l,k)* réalisable et une fois la réalisabilité atteinte, on cherche la fréquence la plus haute f_{max} utilisée dans cette solution. On ajoute une contrainte d'interdiction sur toutes les variables, x_i $\#[f_{max},\ldots,infini]$, on désaffecte toutes les variables utilisant cette fréquence dans la solution actuelle pour obtenir une configuration partielle S_1 et on relance la méthode de base de recherche de réalisabilité avec comme point de départ la dernière configuration partielle S_1. Ce processus est répété plusieurs fois jusqu'à ce que la preuve de non réalisabilité soit atteinte ou que le temps maximum donné soit atteint.

3.4.4.4. MinSpec

Pour un niveau de relâchement unaire l et un niveau de relâchement non-unaire k, ce mode vise à trouver une solution *(l,k)*-réalisable qui minimise la largeur totale du spectre utilisée par les variables de la cible.

La présence de plusieurs types de contraintes binaires ainsi que des contraintes n-aires rend ce critère beaucoup plus difficile à optimiser que dans le cas AFD classique. Il ne suffit donc pas de faire du *MinFreqD* et du *MaxFreqG* pour optimiser ce critère.

La stratégie d'optimisation *MinSpec* repose sur la notion de réalisabilité et sur l'ajout des contraintes d'interdiction sur les variables. On définit l'algorithme suivant :

- Lancer *MaxFreqG* pour obtenir f_{min}, puis ajouter une contrainte d'interdiction aux variables x_i $\#[0,\ldots, f_{min}]$.

- Lancer *MinFreqD* pour obtenir f_{max}, puis ajouter une contrainte d'interdiction aux variables x_i $\#[f_{max}, \ldots, f_n]$.

Le schéma ci-dessous traduit la position des deux bornes identifiées sur le spectre de fréquences.



- Ensuite on déplace le domaine en faisant glisser les bornes de l'intervalle d'une fréquence vers le bas du spectre et on supprime une fréquence en haut du spectre. Le déplacement de l'intervalle vise si possible à ramener le problème à *MinFreqD* qui est plus simple ; la suppression réduit le *MinSpec*. Les opérations effectuées sont donc :
  - $f_{max} = f_{max} - 2$ (suppression de deux fréquences en haut)
  - $f_{min} = f_{min} - 1$ (ajout d'une fréquence en bas)
  - Lancer l'optimisation *MinFreqD* avec pour domaine de toutes les variables $[f_{min}, \ldots, f_{max}]$
  - Si f_{min} devient la borne inférieure du spectre disponible alors on se contente de faire du *MinFreqD*, sinon on répète entièrement le processus.

Si le domaine n'est pas continu, on contraint f_{min} et f_{max} à appartenir à un domaine d'une variable du problème. Donc par exemple, si $f_{max} = f_{max}$ *-2* conduit à f_{max} qui est une fréquence non autorisée alors on cherche directement la première fréquence autorisée en descendant progressivement dans le spectre. Ce processus est répété plusieurs fois jusqu'à ce que la preuve de non réalisabilité soit atteinte ou que le temps maximum donné soit atteint.

3.4.4.5. MinEcart

Pour un niveau de relâchement unaire *l* et un niveau de relâchement non unaire *k*, ce mode vise à trouver une solution *(l,k)*-réalisable qui minimise l'écart entre toute fréquence allouée aux deux variables de la cible et une ou deux fréquences choisies par l'allocateur. La solution sera regroupée autour d'une ou deux fréquences particulières.

Si l'objectif est de regrouper la solution autour d'une seule fréquence f_s, la stratégie d'optimisation de ce critère consiste à lancer la recherche de réalisabilité pour trouver une solution initiale, puis supprimer de la solution la fréquence la plus éloignée f_{el} pour obtenir une configuration partielle *S* et ajouter une contrainte d'interdiction sur toutes les variables :

- Si f_{el} est plus grande que f_s alors la contrainte à ajouter est de la forme : x_i $\#[f_{el}, \ldots, infini]$
- Si f_{el} est plus petite que f_s alors la contrainte à ajouter est de la forme : x_i $\#[0, \ldots, f_{el}]$

On relance ensuite la recherche de réalisabilité avec comme point de départ la configuration partielle obtenue *S*.

Si l'objectif est de regrouper la solution autour de deux fréquences, la même stratégie est appliquée à partir de la fréquence la plus éloignée des deux.

Dans les deux cas, le processus est répété plusieurs fois jusqu'à ce que la preuve de non réalisabilité soit atteinte ou que le temps maximum donné soit atteint.

3.4.4.6. MinFreq

Pour un niveau de relâchement unaire *l* et un niveau de relâchement non unaire *k*, ce mode vise à trouver une solution *(l,k)*-réalisable qui minimise le nombre de fréquences utilisées pour les variables de la cible.

Nous reprenons strictement la stratégie définie pour l'AFD en 3.3.4.4.

3.4.4.7. MinCard

Les trois cas suivants sont maintenant des objectifs de compromis avec relâchement de contraintes, c'est un peu plus délicat à traiter. Soit *l* un niveau de repli unaire donné. Sur l'ensemble de solutions *l*-unaires réalisables, ce mode vise à minimiser la somme pondérée des contraintes non unaires non satisfaites en gérant les niveaux de repli *k* sur les contraintes n-aires, binaires comprises. On doit optimiser dans l'ordre les 3 critères :

- Rechercher *k*, le plus petit niveau de relâchement tel qu'il existe une solution *(l,k)*-réalisable.
- Minimiser la somme pondérée des contraintes non unaires non satisfaites au niveau *k*-1.
- Minimiser la somme pondérées de toutes les contraintes non unaires non satisfaites aux niveaux strictement inférieurs à *k*-1.

Notre stratégie d'optimisation repose sur une hybridation entre la recherche de réalisabilité et la découverte de nouvelles solutions de compromis guidée par la gestion de *nogoods*. On réalise un mécanisme d'apprentissage sur les solutions trouvées successivement au niveau *k*-réalisable.

- On recherche d'abord le plus petit niveau de relâchement non-unaire *k* tel qu'il existe une solution réalisable. La stratégie se base sur un lancement itératif de la réalisabilité sur le problème en jouant sur le niveau de relâchement, on commence par le niveau 10 ; si on trouve une solution à ce niveau, on cherche pour le niveau 9 et ainsi de suite. On s'arrête lorsqu'on ne trouve pas une solution réalisable au niveau *k* courant (critère d'arrêt : nombre d'itération ou preuve d'inconsistance). Le niveau *k* est alors fixé pour la suite de la procédure.
- On cherche maintenant à minimiser la somme pondérée des contraintes violées en restant à ce niveau *k* de réalisabilité. A partir de la solution *S* de niveau *k*, on évalue les contraintes non unaires au niveau *k*-1 et aux niveaux strictement inférieurs. C'est notre première solution pour les deux critères à optimiser. Puis les contraintes non unaires non satisfaites au niveau *k*-1 sont transformées en un unique *nogood* qui est ajoutée à la liste des *nogoods* trouvée jusqu'alors ; cet ajout empêche l'algorithme de reproduire la même solution si on relance la réalisabilité. Une fois ce *nogood* archivé, la réalisabilité pour le niveau *k* est relancée à partir de la dernière solution trouvée réparée ; le *nogood* supplémentaire force l'algorithme à adopter une démarche de diversification dans l'espace de recherche, il s'agit là encore d'un mécanisme d'apprentissage pour la découverte de nouvelles solutions. Une nouvelle solution trouvée sera stockée si

elle améliore les sommes pondérées des contraintes violées du niveau *k*-1 et/ou des niveaux strictement inférieurs à *k*-1. Ce processus est répétée plusieurs fois jusqu'à atteindre la preuve de non réalisabilité ou le temps de recherche imparti.

Dans ce processus, on mesure l'importance de la recherche de la réalisabilité avec la gestion des *nogoods*. Une fois le niveau *k* fixé, l'algorithme revient implicitement sur cette étape pour tenter une réalisabilité au niveau *k*-1 en diversifiant la recherche. S'il la prouve on descend encore le niveau sinon on optimise le nombre de contraintes non satisfaites. Toute la procédure est basée sur une méthode constructive, pour la satisfaction comme pour l'optimisation.

Prenons un exemple pour illustrer le procédé algorithmique. Soit un problème donné pour lequel l'algorithme a trouvé une solution réalisable *S* au niveau *k*=5 lors de la première étape.

La deuxième étape est la minimisation des sommes pondérées au niveau *k*-1=4 en calculant les deux critères simultanément. Supposons que la solution de niveau 5 viole 2 contraintes binaires au niveau 4 et 8 pour les niveaux strictement inférieurs; un *nogood* est construit à partir de l'affectation de chaque variable impliquée dans les 2 contraintes. Par exemple si les contraintes impliquent les variable *i, j, p* et *q*, le *nogood* est *ng={(i, a)(j, b)(p, c)(q, d)}* avec *a, b, c* et *d* les ressources affectées aux variables *i, j, p* et *q* dans *S*. Une fois le *nogood* construit, il est ajouté à la liste des *nogood*s et on relance la réalisabilité au niveau 5 pour trouver une autre solution réalisable violant moins de 2 contraintes de niveau 4, ou moins de 8 contraintes de niveaux strictement inférieurs à 4 pour 2 contraintes violées de niveau 4, ou les deux.

Chaque fois qu'une nouvelle solution est trouvée, elle sera considérée comme meilleure si elle donne une somme pondérée des contraintes au niveau *k*-1 plus faible que la somme de la solution stockée. Si la somme pondérée de la nouvelle solution trouvée est égale à la somme pondérée de la solution stockée au niveau *k*-1, on évalue la somme pondérée pour les contraintes de tous les niveaux strictement inférieurs à k-1.

3.4.4.8. MinMob

Soit *k* un niveau de repli non unaire donné. Sur l'ensemble des solutions *k*-non-unaires réalisables, ce mode vise à minimiser le nombre de contraintes unaires non satisfaites en gérant les niveaux de repli *l* sur les contraintes unaires. On doit optimiser dans l'ordre les 3 critères :

- Rechercher *l,* le plus petit niveau de relâchement tel qu'il existe une solution (*l, k*) réalisable.
- Minimiser la somme pondérée des contraintes unaires non satisfaites au niveau *l*–1.
- Minimiser la somme pondérée des contraintes unaires non satisfaites aux niveaux strictement inférieurs à (*l-1*).

La stratégie d'optimisation repose sur des principes identiques à ceux mentionnés pour *MinCard* en deux étapes. L'algorithme recherche tout d'abord la réalisabilité pour le plus petit niveau de relâchement unaire *l*. Puis la satisfaction des contraintes unaires de niveau *l*-1 et de tous les niveaux strictement inférieurs à *l*-1 est évaluée. Ensuite, les contraintes unaires violées de niveau *l*-1 sont transformées en *nogood* et la réalisabilité pour le niveau *l* est relancée à partir de la dernière solution trouvée et réparée.

3.4.4.9. MinMax

Cet objectif est un peu particulier. Soit l un niveau de repli pour les contraintes unaires, soit k un niveau de repli pour les contraintes non unaires. Dans l'ensemble des solutions (l-10) réalisables, on recherchera à minimiser la non satisfaction uniforme des contraintes non unaires de niveaux supérieurs ou égal à k.

La stratégie d'optimisation pour ce critère repose sur une hybridation entre la recherche de réalisabilité, la gestion des *nogood*s pour la diversification et une restriction dans la procédure d'extension d'une configuration partielle. Elle se déroule en quatre étapes :

1. La recherche d'une solution réalisable au niveau (l-10). On évalue cette solution pour obtenir un score de départ, appelé *bestScore*, qui correspond au *MinMax* de la solution trouvée.
2. L'écriture du *nogood*. A partir de cette solution, on considère toutes les contraintes qui participent à l'évaluation du *MinMax*, c'est-à-dire toutes les contraintes violées non-unaires de niveaux supérieurs ou égal à k, et on les transforme en un *nogood* afin de ne pas retomber sur la même solution trouvée.
3. La création d'une nouvelle solution de départ. On désaffecte toutes les variables qui participent aux *nogoods* issus de la solution.
4. On relance la réalisabilité à partir de l'étape 1 en passant en contrainte dure le *bestScore* trouvé afin d'obliger la méthode à trouver une solution meilleure que la meilleure trouvée en terme de *MinMax*. Donc la méthode ne permettra pas de faire une affectation qui rend l'évaluation plus grande que le *bestScore* déjà trouvé.

La dernière solution trouvée correspond à la meilleure pour l'objectif fixé. Elle est retournée une fois que le temps de recherche imparti est atteint.

3.4.5. Résultats et analyse

Dans cette section, nous présentons les tests réalisés sur les 30 instances privées fournies par le CELAR. Pour certaines de ces instances nous analyserons l'évolution du nombre de *nogoods* stockés dans la liste Tabou des *nogoods* ainsi que leurs qualités. Nous donnons ensuite les résultats obtenus par *Tabu-NG* sur les instances avec une comparaison avec les optimums ou les meilleures solutions connues fournis par le CELAR à l'aide de divers outils. Les optimums connus fournis par le CELAR sont les meilleurs résultats obtenus sur plusieurs lancements de 3 heures chacun. *Tabu-NG* est évaluée sur la même machine afin que la comparaison reste significative.

3.4.5.1. Analyse de la liste Tabou des nogoods

La taille de la liste Tabou Γ est très importante pour le bon fonctionnement de *Tabu-NG*. Si Γ contient un très grand nombre de *nogoods*, *Tabu-NG* peut prendre un temps non négligeable pour gérer l'accès à cette liste et la recherche en sera ralentie.

L'idéal est de garder la totalité des *nogoods* identifiés pendant la recherche dans Γ, alors qu'en réalité cette possibilité reste difficile. Nous avons donc fixé la taille de Γ à 15 000. Cette liste se comporte comme une liste FIFO (*First In, First Out*).

Dans cette partie, nous allons montrer l'évolution de la taille de Γ en fonction du temps, ou du nombre d'itérations, pour deux petits problèmes et deux grands problèmes. Nous verrons que le nombre de *nogoods* stockés ne dépend pas forcément de la taille du problème mais plutôt de la nature des contraintes du problème.

La Figure 19 donne l'évolution de la taille de la liste Tabou des *nogoods* en fonction du nombre d'itérations pour l'instance SCEN02 avec l'objectif *MaxFreqG*. Le graphique donne l'évolution jusqu'à l'itération 33 000.

SCEN02 est un petit problème de 41 variables, 555 contraintes et 196 ressources. A noter que la solution trouvée par *Tabu-NG* avec *MaxFreqG* comme objectif pour SCEN02 est l'optimum connu sur ce problème. Durant la recherche d'une solution avec *MaxFreqG,* la recherche de réalisabilité a été lancée 7 fois. Sur le graphique, Γ n'a pas été vidée une seule fois ce qui indique que la méthode n'a jamais été bloquée (méthode n'est pas redémarré 3 fois avec une remise à zéro des poids). Nous remarquons que la notion de dominance entre les *nogoods* fonctionne très bien ; chaque fois qu'un *nogood* est identifié, il supprime de la liste les *nogoods* qu'il domine. Ce phénomène est visible via les oscillations de la courbe de la Figure 19. Pour ce problème, *Tabu-NG* n'a aucune difficulté dans la gestion de Γ. La taille de la liste Tabou des *nogoods* varie de 100 à 700 éléments.

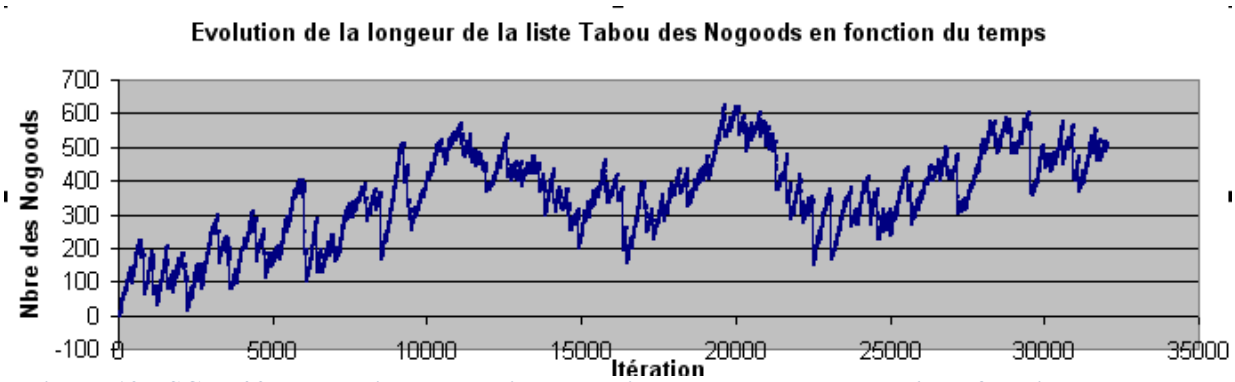


Figure 19 : SCEN02 - Evolution de la taille de la liste Tabou Γ des *nogoods* en fonction du temps

Afin d'évaluer la qualité des *nogoods* stockés dans Γ, nous avons développé une routine qui vérifie la validité de chacun des *nogoods* de la liste. Autrement dit, on vérifie si le *nogood* stocké amène à un *deadend* ou non. Pour cela, on prend chaque *nogood*, on le considère comme une configuration partielle et on applique un algorithme de filtrage. Si un domaine d'une variable devient vide, alors c'est un *nogood* exact, sinon c'est un *nogood* partiel.

Le Tableau 16 montre le nombre et le pourcentage des *nogoods* exacts et partiels pour SCEN02 avec *MaxFreqG*. Pour chaque type de *nogoods*, nous donnons aussi la moyenne de la longueur des *nogoods*. Le test de qualité des *nogoods* est fait après chaque amélioration de l'objectif, notée *Pas* dans le tableau.

Nous remarquons que :

- Le pourcentage des *nogoods* exacts est toujours plus grand que celui des partiels.
- Les *nogood*s stockés sont de bonne qualité. La moyenne de la taille des *nogood*s est petite, sachant que le problème compte 50 variables. Plus les *nogoods* sont courts plus les coupes sont hautes dans l'arbre de recherche, ce qui aide *Tabu-NG* à réduire l'espace de recherche d'une manière très efficace.

- Le nombre de *nogoods* partiels est non négligeable mais la méthode a réussi à trouver une solution, l'optimum connu, sans redémarrage ni suppression des *nogoods* dans Γ. Nous concluons que la méthode diversifie bien sa recherche, elle est capable de se rediriger vers une bonne branche afin de trouver une solution pour le problème.

SCEN02	***nogood*s exacts**		***nogood*s partiels**	
Pas	**Nombre (%)**	**Longueur moyenne**	**Nombre (%)**	**Longueur moyenne**
1	26 (52%)	6.53846	24 (48%)	7.16667
2	78 (64%)	6.01282	44 (36%)	6.68182
3	79 (83%)	4.79747	16 (17%)	5.1875
4	58 (85.3%)	3.7069	10 (14.7%)	4.3
5	72 (57.6%)	4.68056	53 (42.4%)	6.4717
6	63 (76.8%)	3.96825	19 (23.2%)	5.26316
7	73 (66.4%)	4.15068	37 (33.6%)	5.81081

Tableau 16 : Qualité des *nogoods* stockés pour SCEN02

La Figure 20 montre l'évolution de la taille de Γ en fonction du temps pour le problème SCEN05 avec l'objectif *MaxFreqG* ; la taille reste maîtrisée. SCEN05 est un petit problème de 50 variables avec deux fois plus de contraintes et ressources que SCEN02, 1261 contraintes et 514 ressources. SCEN05 est un problème moyennement difficile à résoudre en s'appuyant sur les outils du CELAR. Afin de maximiser la fréquence basse utilisée dans la solution, la stratégie *MaxFreqG* lance 60 fois la réalisabilité. A noter que la solution trouvée par *Tabu-NG* avec *MaxFreqG* comme objectif pour SCEN05 est l'optimum connu sur ce problème.

Le processus d'amélioration de la solution courante par *Tabu-NG* a été retracé ci-dessous pour l'objectif *MaxFreqG* :

- 46 améliorations à partir de la solution consistante précédente. Ce qui montre que les *nogoods* identifiés pendant la recherche guident d'une manière efficace la recherche vers de nouvelles solutions réalisables.
- 4 améliorations après redémarrage de la recherche à partir d'une solution vide. Ce qui montre que la phase de diversification est très importante pour *Tabu-NG*, elle permet d'explorer des nouvelles branches.
- 6 améliorations après la première suppression des *nogoods* stockés dans Γ, 1 amélioration après la deuxième et 3 améliorations après la dernière. Trouver des solutions après la suppression des *nogoods* stockés semble montrer que des *nogoods* partiels peuvent empêcher d'atteindre certaines solutions. Le fait de trouver 1 solution après la $2^{ème}$ suppression de Γ et 3 solutions après la $3^{ème}$ suppression de Γ indique que la recherche pourrait butter parfois sur des *nogoods* partiels qui sont un obstacle sur la voie de nouvelles solutions réalisables. Il faudrait donc chercher à en limiter le nombre.

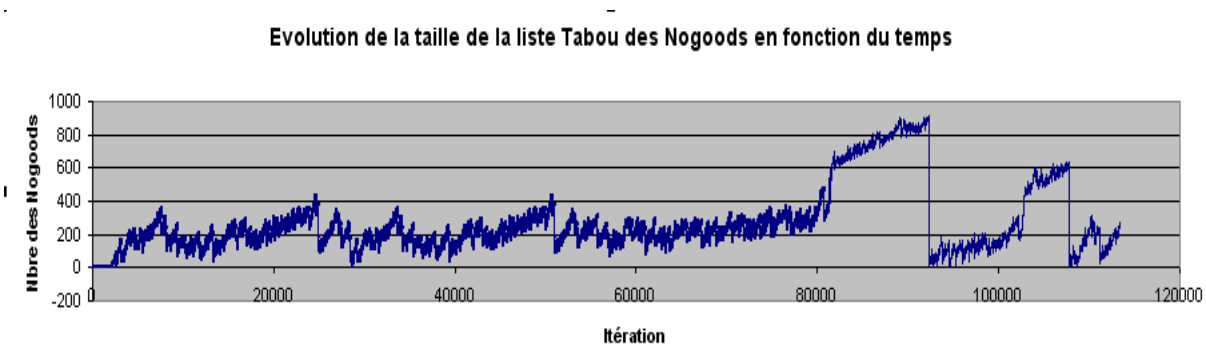


Figure 20 : SCEN05 - Evolution de la taille de la liste Tabou Γ des *nogoods* en fonction du temps

Nous avons analysé pour SCEN05, la qualité des *nogoods* stockés dans la liste Γ. Nous étions étonnés par les résultats car *Tabu-NG* n'est jamais tombé sur un *nogood* inexact, tous les *nogoods* identifiés pendant la recherche sont exacts ce qui est en contradiction avec l'observation précédente! Nous avons relancé la méthode sur SCEN05 en gardant le principe du restart (redémarrer d'une solution vide avec liste Tabou Δ et poids des *nogoods* à zéro) mais sans vider la liste des *nogoods*. *Tabu-NG* a encore trouvé l'optimum connu pour SCEN05. Nous pouvons conclure, que c'est la diversification liée au redémarrage de la recherche qui a permis à *Tabu-NG* de trouver le meilleur optimum connu et non pas la suppression des *nogoods* dans Γ.

Nous avons aussi analysé la qualité des *nogoods* pour SCEN25. SCEN25 est un grand problème de 364 variables, 10 032 contraintes avec des contraintes *n-aires* et un domaine de taille 841. Pour SCEN25, tous les *nogoods* identifiés pendant la recherche sont exacts. La moyenne de la longueur des *nogoods* stockés est de 4,73824. *Tabu-NG* a pu donc bénéficier d'une manière très efficace des *nogoods* stockés, en éliminant des zones importantes dans l'espace de recherche. A noter que ce scénario est très difficile à résoudre, aucun outil du CELAR n'a pu résoudre ce problème, alors que *Tabu-NG* a réussi à trouver une très bonne solution.

La Figure 21 montre l'évolution de la taille de la liste Tabou des *nogoods* Γ pour le problème SCEN12. La taille est parfaitement maîtrisée, ne dépassant pas 400 éléments. SCEN12 est un grand problème de 702 variables, 39 123 contraintes et de domaine de taille 72 ; c'est un problème difficile à résoudre par *Tabu-NG* ainsi que par les outils du CELAR. Afin de maximiser la fréquence basse utilisée dans la solution, *Tabu-NG* a lancé la réalisabilité 8 fois. Sur ces lancements, *Tabu-NG* a réussi à améliorer la solution :

- 4 fois sans redémarrage et sans suppression des *nogoods* stockés.
- 3 fois après le 1er redémarrage de la recherche, ce qui montre une nouvelle fois l'intérêt de la diversification.
- 0 fois après la première et la seule suppression de la liste Tabou des *nogoods.*

Sur un problème de grande taille, *Tabu-NG* avec les *nogoods* partiels prouve une nouvelle fois qu'elle est capable de trouver une solution pour le problème donné. A noter que la solution trouvée par *Tabu-NG* avec *MaxFreqG* comme objectif pour SCEN12 est très proche de l'optimum connu (une seule fréquence de différence).

En analysant la liste Tabou des *nogoods* du SCEN12, on s'est aperçu qu'un seul *nogood* inexact est identifié pendant la recherche, tous les autres *nogoods* sont exacts. La longueur du *nogood* inexact est de 2, ce *nogood* pourra être pénalisant pour *Tabu-NG* car il est très sévère (très court). La taille moyenne des *nogoods* exacts stockés est de 9.66, ce qui aide *Tabu-NG* à

réduire l'espace de recherche d'une manière très efficace vu le nombre de variables du problème (702 variables).

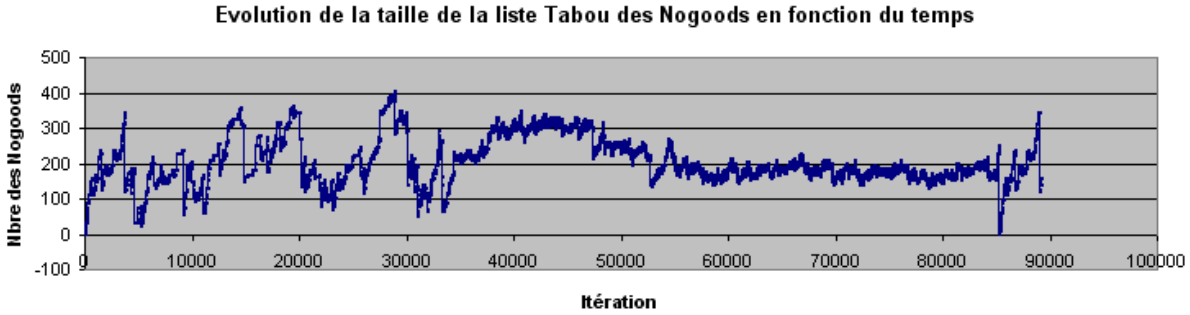


Figure 21 : SCEN12 - Evolution de la liste Tabou *Γ* des *nogoods* en fonction du temps

La Figure 22 montre l'évolution de la taille de la liste Tabou des *nogoods* en fonction du temps pour SCEN30 qui est un problème de très grande taille avec 3 000 variables, 182 739 contraintes et un domaine de taille 91. C'est un problème très difficile à résoudre, le problème était qualifié comme irréalisable par le CELAR car aucun outil n'était capable de fournir une solution.

Tabu-NG a pu fournir une solution pour ce problème en moins d'une heure d'exécution sans redémarrage et sans suppression de *nogoods* stockés. La taille de la liste Tabou des *nogoods* est relativement constante et faible au cours du temps. La taille du problème n'est pas un facteur direct de croissance de la liste Tabou des *nogoods*.

En analysant la liste Tabou des *nogoods* du SCEN30, nous avons trouvé que tous les *nogoods* stockés sont exacts et de taille moyenne 10,5354, sauf un *nogood* inexact de taille 2. Ce *nogood* inexact de taille 2 est un *nogood* très sévère, il élimine une zone très importante de l'espace de recherche mais cela n'a pas empêché *Tabu-NG* de trouver une solution réalisable à ce problème réputé très difficile. Donc un *nogood* inexact n'est pas forcément source d'empêchement pour trouver une solution réalisable. On ne peut pas conclure sur ce point.

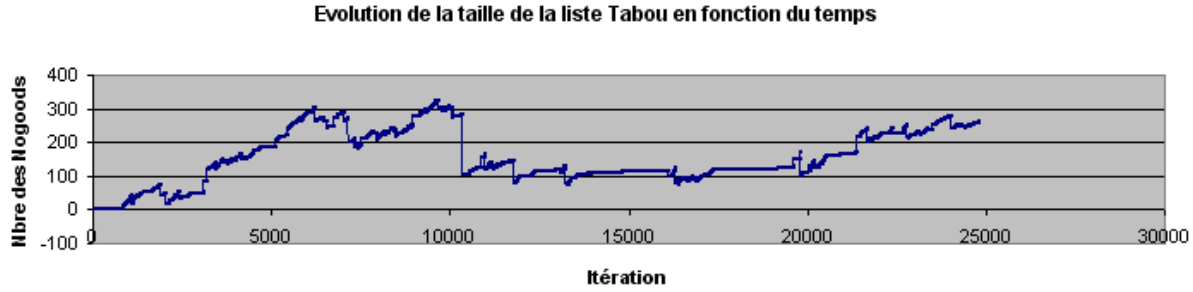


Figure 22 : SCEN30 - Evolution de la liste Tabou *Γ* des *nogoods* en fonction du temps

Cette section a permis de montrer que le nombre de *nogoods* stockés dans la liste *Γ* augmente légèrement au fur et à mesure de la recherche d'une solution pour les problèmes d'affectation de fréquences, puis se stabilise très vite. Pour toutes les instances traitées, et quel que soit l'objectif, la taille de la liste *Γ* reste plus ou moins constante. Nous n'avons eu aucun problème de gestion de la liste *Γ* sur l'ensemble des problèmes traités y compris sur les plus grands problèmes. La notion de dominance entre *nogoods* semble efficace pour gérer la taille de la liste ; elle est aussi très efficace en termes de temps de recherche des *nogoods* puisque l'algorithme perd moins de temps à parcourir la liste réduite.

Les *nogoods* identifiés pendant la recherche, et pour toutes les instances, sont de très bonne qualité en général. La nature physique des contraintes a beaucoup contribué à la qualité de ces *nogoods*. Même quand les *nogoods* ne sont pas exacts, lorsqu'ils portent sur une partie des couples *(variables, valeur)* d'un *nogood* exact, *Tabu-NG* parvient à atteindre l'optimalité ce qui montre l'efficacité de notre approche d'identification et de gestion des *nogoods*. La taille moyenne des *nogoods* identifiés est petite par rapport au nombre de variables des problèmes traités. Ainsi, *Tabu-NG* a pu bénéficier d'une manière très efficace des *nogoods* identifiés pour réduire l'espace de recherche et par la suite se concentrer sur les zones prometteuses afin de chercher une solution.

A noter enfin au cours de ces tests visant à mieux cerner le comportement de la méthode que les phases de diversification ont été très importantes pour améliorer le fonctionnement de *Tabu-NG*. La procédure de restart qui permet à l'algorithme de redémarrer d'une solution initiale vide avec une remise à zéro de la liste Tabou sur le choix des variables et une remise à zéro des poids des affectations calculées apporte un vrai plus. Le redémarrage de *Tabu-NG* bénéficie simplement de l'apprentissage via la liste Tabou des *nogoods* et entame une nouvelle recherche uniquement avec ce capital. L'algorithme reste déterministe vu qu'il n'y a aucun choix aléatoire.

3.4.5.2. Résultats sur les instances fournies

Dans cette section, nous présentons les résultats finaux de *Tabu-NG* sur les 30 instances privées. Ces résultats sont comparés avec les optimums ou les meilleurs scores connus fournis par le CELAR. Tous les tests ont été effectués une seule fois sur la même machine selon une exigence contractuelle (PC avec un processeur de 2.4 GHZ et 1 Go de RAM). Pour chaque objectif, les problèmes traités ne sont pas forcément les mêmes, ce qui rend la chose plus difficile ; les contraintes peuvent être plus ou moins dures selon le niveau de relâchement demandé pour chaque contrainte.

Les outils du CELAR ne traitent pas les contraintes harmoniques et les contraintes d'intermodulation. Les optimums fournis par le CELAR pour les instances contenant ces 2 types de contraintes ont été obtenus sur les instances relâchées de ces 2 types de contraintes. La comparaison entre *Tabu-NG* et l'optimum connu pour ces cas est donc à titre indicatif puisque *Tabu-NG* prend en compte tous les types des contraintes. L'ajout des contraintes à un problème donné peut parfois faciliter la résolution de ce problème. Ces contraintes peuvent contribuer d'une manière très efficace à la réduction de l'espace de recherche, surtout si ces contraintes sont dures en termes de satisfaction. Mais elles peuvent aussi durcir le problème si elles éliminent un grand nombre de solutions.

Pour chaque objectif, nous dressons le résultat de *Tabu-NG* obtenu sur chaque instance, l'optimum ou la meilleure solution connu, ainsi qu'une remarque si l'instance contient des contraintes de type harmonique ou intermodulation. La valeur *false* dans le tableau de résultats signifie que la méthode a pu prouver la non-réalisabilité du problème. La valeur *RIEN* signifie que la méthode n'a pas pu trouver une solution qui satisfait toutes les contraintes du problème dans le temps imparti autrement dit il reste un certain nombre de contraintes violées.

Le temps d'exécution alloué à la résolution de chaque instance est donné dans le Tableau 17.

SCENARIO	Temps d'exécution	SCENARIO	Temps d'exécution
SCEN01	6h	SCEN16	1h
SCEN02	6h	SCEN17	1h20
SCEN03	6h	SCEN18	2h
SCEN04	6h	SCEN19	1h
SCEN05	6h	SCEN20	1h
SCEN06	6h	SCEN21	2h
SCEN07	6h	SCEN22	2h
SCEN08	6h	SCEN23	2h
SCEN09	6h	SCEN24	2h
SCEN10	6h	SCEN25	1h
SCEN11	2h	SCEN26	2h
SCEN12	2h	SCEN27	1h
SCEN13	1h	SCEN28	2h
SCEN14	1h	SCEN29	2h
SCEN15	2h	SCEN30	2h

Tableau 17 : Temps d'exécution alloué à chaque instance

3.4.5.3. MinFreq

Le Tableau 18 dresse les résultats obtenus sur l'ensemble des instances avec le critère *MinFreq* à optimiser. *Tabu-NG* a pu améliorer l'optimum connu sur 9 instances données sur 30. Elle n'a pas pu atteindre l'optimum connu sur 3 instances (SCEN14, 17 et 28).

Points remarquables :

- Amélioration de l'optimum connu du SCEN05 d'une façon très importante de 38 à 16 fréquences.
- Solutions réalisables trouvées pour 8 instances sur lesquelles les outils du CELAR n'ont pas trouvé des solutions de compromis dans le temps imparti.
- N'a pas atteint les optimums connus sur 3 instances.

Problème	*Tabu-NG*	Optimum connu	Remarque
SCEN 01	8	8	
SCEN 02	10	10	
SCEN 03	false	false	
SCEN 04	8	8	
SCEN 05	16	38	
SCEN 06	false	false	
SCEN 07	6	6	
SCEN 08	9	9	
SCEN 09	false	false	
SCEN 10	15	15	
SCEN 11	17	false	
SCEN 12	61	RIEN	
SCEN 13	false	false	
SCEN 14	16	15	
SCEN 15	false	false	

SCEN 16	8	RIEN	Présence des contraintes d'intermodulation et harmoniques
SCEN 17	35	31	
SCEN 18	false	false	
SCEN 19	false	false	
SCEN 20	17	RIEN	//
SCEN 21	false	false	
SCEN 22	37	RIEN	
SCEN 23	false	false	
SCEN 24	41	RIEN	
SCEN 25	18	RIEN	//
SCEN 26	false	false	
SCEN 27	40	RIEN	//
SCEN 28	26	19	
SCEN 29	false	false	
SCEN 30	91	RIEN	

Tableau 18 : Comparaison des résultats obtenus pour *MinFreq*

Le critère *MinFreq* est très difficile à optimiser d'un point de vue algorithmique. Les résultats obtenus sont encourageants, ils prouvent que *Tabu-NG* est une méthode efficace pour la recherche de la réalisabilité. Les résultats obtenus montrent aussi que le traitement des contraintes n-aires est aussi efficace ; par exemple, pour les problèmes SCEN16, 20 et 25 *Tabu-NG* arrive à trouver une solution alors qu'on ne connaissait pas de solution réalisable pour ces problèmes.

3.4.5.4. MinSpec

Le Tableau 19 montre les résultats obtenus sur l'objectif *MinSpec*.

Points remarquables :

- Amélioration de l'optimum connu pour 5 instances (SCEN05, 09, 22, 24 et 27), dont 3 instances (SCEN09, 22, 24) où il n'y avait pas de solution connue qui satisfaisait toutes les contraintes.
- N'a pas atteint l'optimum connu pour 3 instances (SCEN12, 17 et 28).
- Les instances avec des contraintes harmoniques et intermodulation ne sont pas comparables, nous ne pouvons donc rien conclure sur ces instances puisque les optimums fournis sont pour des instances moins contraignantes que celles traitées par *Tabu-NG*.

D'après les résultats obtenus sur SCEN12, 17 et 28, nous pouvons dire que ce sont des instances difficiles à résoudre par *Tabu-NG*. Les résultats obtenus pour les critères *MinSpec* et *MinFreq* dépendent beaucoup des stratégies d'optimisation utilisées. Nous allons voir dans les sections suivantes, que lorsque la stratégie est exacte, *Tabu-NG* réussit à atteindre l'optimalité pour ces 3 problèmes.

Problème	*Tabu-NG*	Optimum connu	Remarque
SCEN 01	156	156	
SCEN 02	185	185	
SCEN 03	false	false	
SCEN 04	3288500	3288500	
SCEN 05	5080	5100	
SCEN 06	false	false	
SCEN 07	152	152	
SCEN 08	140	140	
SCEN 09	410	RIEN	
SCEN 10	164	164	
SCEN 11	false	false	
SCEN 12	264	260	
SCEN 13	false	false	
SCEN 14	4960	4960	
SCEN 15	false	false	
SCEN 16	9125	7200	Présence des contraintes d'intermodulation et harmoniques
SCEN 17	264	256	
SCEN 18	false	false	
SCEN 19	false	false	
SCEN 20	11100	7750	//
SCEN 21	false	false	
SCEN 22	260	RIEN	
SCEN 23	false	false	
SCEN 24	332	RIEN	
SCEN 25	11250	9600	//
SCEN 26	false	false	
SCEN 27	38950	31300	//
SCEN 28	300	296	
SCEN 29	false	false	
SCEN 30	false	false	

Tableau 19 : Comparaison des résultats obtenus pour *MinSpec*

3.4.5.5. MinFreqD

Le Tableau 20 compare les résultats obtenus sur l'objectif *MinFreqD*.

Points remarquables :

- Amélioration de l'optimum connu sur 6 instances. *Tabu-NG* a pu améliorer l'optimum notamment sur un petit problème (SCEN02).
- Résolution des problèmes avec des contraintes n-aires (SCEN16, 20, 25 et 27) d'où l'efficacité du traitement des contraintes n-aires.
- Résultat non comparable sur SCEN20 étant donné que c'est une instance avec des contraintes harmoniques et d'intermodulation, l'optimum fourni par le CELAR n'est pas comparable avec les résultats obtenus par *Tabu-NG* puisque le problème à résoudre n'est pas le même.

Problème	*Tabu-NG*	Optimum connu	Remarque
SCEN 01	4156	4156	
SCEN 02	1385	1392	
SCEN 03	false	false	
SCEN 04	7722000	7722000	
SCEN 05	49775	49775	
SCEN 06	false	false	
SCEN 07	3160	3160	
SCEN 08	3160	3160	
SCEN 09	false	false	
SCEN 10	3168	3168	
SCEN 11	false	false	
SCEN 12	1960	RIEN	
SCEN 13	false	false	
SCEN 14	49825	49825	
SCEN 15	false	false	
SCEN 16	47150	47300	Présence des contraintes d'intermodulation et harmoniques
SCEN 17	false	false	
SCEN 18	false	false	
SCEN 19	false	false	
SCEN 20	40200	37900	//
SCEN 21	false	false	
SCEN 22	2032	RIEN	
SCEN 23	false	false	
SCEN 24	1620	RIEN	
SCEN 25	39600	39600	//
SCEN 26	false	false	
SCEN 27	70000	RIEN	//
SCEN 28	1596	1596	
SCEN 29	false	false	
SCEN 30	false	false	

Tableau 20 : Comparaison des résultats obtenus pour *MinFreqD*

3.4.5.6. MaxFreqG

Le Tableau 21 compare les résultats obtenus sur le critère *MaxFreqG*.

Points remarquables :

- Amélioration de l'optimum connu sur 9 instances. Malgré la petite taille du SCEN02, *Tabu-NG* a réussi à améliorer son optimum connu. SCEN05 est un problème difficile à résoudre pour *MaxFreqG* car son domaine est très grand (514 fréquences) mais *Tabu-NG* a pu améliorer son optimum connu. SCEN30 était qualifié comme non réalisable, *Tabu-NG* a réussi à trouver une solution et par la suite à prouver sa réalisabilité.
- N'a pas atteint l'optimum sur SCEN12 (une seule fréquence de différence) ; une exécution de 24 heures sur cette instance donne une solution du niveau de l'optimum connu.

Problème	*Tabu-NG*	Optimum connu	Remarque
SCEN 01	4028	4028	
SCEN 02	1210	1206	
SCEN 03	False	false	
SCEN 04	false	false	
SCEN 05	47905	46925	
SCEN 06	false	false	
SCEN 07	3020	3020	
SCEN 08	3020	3020	
SCEN 09	false	false	
SCEN 10	3020	3020	
SCEN 11	false	false	
SCEN 12	1708	1712	
SCEN 13	false	false	
SCEN 14	44865	44865	
SCEN 15	false	false	
SCEN 16	46075	RIEN	Présence des contraintes d'intermodulation et harmoniques
SCEN 17	1300	1300	
SCEN 18	false	false	
SCEN 19	false	false	
SCEN 20	39075	RIEN	//
SCEN 21	false	false	
SCEN 22	1800	RIEN	
SCEN 23	false	false	
SCEN 24	1324	RIEN	
SCEN 25	39900	RIEN	//
SCEN 26	false	false	
SCEN 27	30050	RIEN	//
SCEN 28	RIEN	RIEN	
SCEN 29	false	false	
SCEN 30	2280	RIEN	

Tableau 21 : Comparaison des résultats obtenus pour *MaxFreqG*

3.4.5.7. MinEcart

Le Tableau 22 compare les résultats obtenus sur le critère *MinEcart*.

Points remarquables :

- Amélioration de l'optimum connu sur 6 instances, notamment le SCEN02 et le SCEN30.
- Solution pour 3 problèmes sans solution auparavant (SCEN22, 24 et 30)
- 2 problèmes avec contraintes n-aires ne sont pas comparables (SCEN20 et 25).

Problème	*Tabu-NG*	Optimum connu	Remarque
SCEN 01	36	36	
SCEN 02	80	90	
SCEN 03	false	false	
SCEN 05	815	815	
SCEN 06	false	false	
SCEN 09	false	false	
SCEN 11	false	false	
SCEN 12	false	RIEN	
SCEN 13	false	false	
SCEN 14	2825	2825	
SCEN 15	false	false	
SCEN 16	45750	45800	Présence des contraintes d'intermodulation et harmoniques
SCEN 17	100	100	
SCEN 18	false	false	
SCEN 19	false	false	
SCEN 20	39600	36500	//
SCEN 21	false	false	
SCEN 22	190	RIEN	
SCEN 23	false	false	
SCEN 24	404	RIEN	
SCEN 25	38200	38100	//
SCEN 26	false	false	
SCEN27	68600	21	//
SCEN 28	388	388	
SCEN 29	false	false	
SCEN 30	192	RIEN	

Tableau 22 : Comparaison des résultats obtenus pour *MinEcart*

3.4.5.8. MinCard

MinCard est un problème difficile pour une méthode constructive de part ses deux critères à optimiser après la réalisabilité. Le Tableau 21 présente en colonne les 3 critères : niveau *k*-réalisable, somme des contraintes pondérées violées niveau *k*-1 et somme des contraintes pondérées violées de tous les niveaux plus petits que *k*-1. Les méthodes qui manipulent des configurations complètes sont très efficaces pour minimiser ou maximiser un objectif, ce qui explique en partie la difficulté de notre méthode à atteindre l'optimalité connue sur l'ensemble des instances.

Points remarquables :

- Amélioration de l'optimum connu sur 6 instances (SCEN9, 16, 20, 22, 24 et 25).
- Solution pour 3 problèmes sans solution auparavant (SCEN9, 22 et 24)
- N'a pas atteint l'optimalité sur 5 instances. Pour SCEN02, 07, 08 et 17, *Tabu-NG* a atteint le niveau *k* de l'optimum connu, mais elle n'a pas minimisé au même niveau que l'optimum connu la somme pondérée des contraintes non-unaires violées au niveau *k*-1. Pour SCEN12, le résultat est mauvais puisqu'il y a 5 niveaux d'écart sur la réalisabilité.

- Le SCEN27 n'est pas comparable du fait des contraintes n-aires.

Problème	***Tabu-NG***			**Optimum connu**	**Remarque**
	K	**CTR-1**	**CTR_TOUS**	**K, CTR-1**	
SCEN 01	2	6	12	2, 4	
SCEN 02	4	11	65	4, 9	
SCEN 03	false			false	
SCEN 04	9	1	11	9, 1	
SCEN 05	false			false	
SCEN 06	false			false	
SCEN 07	6	7	133	6, 2	
SCEN 08	4	6	73	4, 3	
SCEN 09	3	7	21	RIEN	
SCEN 10	3	3	14	3, 3	
SCEN 11	false			false	
SCEN 12	6	61	1713	1, 153	
SCEN 13	false			false	
SCEN 14	0	0	0	0, 0	
SCEN 15	false			false	
SCEN 16	0	0	0	7, 3	Présence des contraintes d'intermodulation et harmoniques
SCEN 17	8	99	984	8, 12	
SCEN 18	false			false	
SCEN 19	false			false	
SCEN 20	1	312	312	2, 897	//
SCEN 21	false			false	
SCEN 22	9	350	10327	RIEN	
SCEN 23	false			false	
SCEN 24	7	218	5283	RIEN	
SCEN 25	0	0	0	7, 1	//
SCEN 26	false			false	
SCEN 27	7	4450	31930	4, 168	//
SCEN 28	false			false	
SCEN 29	false			false	
SCEN 30	false			false	

Tableau 23 : Comparaison des résultats obtenus pour *MinCard*

3.4.5.9. MinMob

MinMob est un problème difficile pour une méthode constructive de part ses deux critères à optimiser après la réalisabilité. Le Tableau 24 compare les résultats obtenus sur l'ensemble des instances contenant des contraintes unaires.

Points remarquables :

- Résolution de SCEN09.
- N'a pas atteint l'optimalité sur SCEN14 à 1 contrainte violée près.

Problème	*Tabu-NG*			Optimum connu
	K	CTR-1	CTR_TOUS	K, CTR-1
SCEN 05	6	1	11	K=6 CTR-1=1
SCEN 09	3	7	21	RIEN
SCEN 14	6	2	32	K=6 CTR-1=1

Tableau 24 : Comparaison des résultats obtenus pour *MinMob*

3.4.5.10. MinMax

MinMax est un problème difficile pour les méthodes constructives. Le Tableau 23 compare les résultats obtenus sur l'ensemble des instances.

Problème	*Tabu-NG*	Optimum connu	Remarque
SCEN 01	6	6	
SCEN 02	2,198	9	
SCEN 03	false	false	
SCEN 04	3459	3459	
SCEN 05	false	false	
SCEN 06	false	false	
SCEN 07	1	1	
SCEN 08	4	4	
SCEN 09	false	false	
SCEN 10	4	4	
SCEN 11	false	false	
SCEN 12	131	10	
SCEN 13	false	false	
SCEN 14	0	0	
SCEN 15	false	false	
SCEN 16	6,173	RIEN	Présence des contraintes d'intermodulation et harmoniques
SCEN 17	182	56	
SCEN 18	243	false	
SCEN 19	false	false	
SCEN 20	8,728	RIEN	//
SCEN 21	21	false	
SCEN 22	156	false	
SCEN 23	false	false	
SCEN 24	187	5	
SCEN 25	78,971	RIEN	//
SCEN 26	false	false	
SCEN 27	9,188	RIEN	//
SCEN 28	false	false	
SCEN 29	false	false	
SCEN 30	210	RIEN	

Tableau 25 : Comparaison des résultats obtenus pour *MinMax*

Points remarquables :

- Amélioration de l'optimum connu sur 7 instances (SCEN2, 16, 20, 22, 25, 27 et 30).
- Solution pour 5 problèmes sans solution auparavant (SCEN16, 20, 25, 27 et 30).

- N'a pas atteint l'optimalité sur 3 instances (SCEN12, 17 et 24). Ces scénarios sont de grandes tailles et *Tabu-NG* a besoin de plus du temps afin de pouvoir minimiser ce critère. Pour des lancements de 24 heures, *Tabu-NG* a pu améliorer les scores sur ces scénarios, par contre on restait quand même très loin des optimums connus (110 pour SCEN12, 159 pour SCEN17 et 173 pour SCEN24).

3.4.6. Discussions

Pour l'étude des instances privées fournis par le CELAR dans le cadre du projet, le cahier de charges prévoyait une phase d'évaluation de performances du module d'allocation de fréquences livré à la fin du projet. L'évaluation des performances du module pour chacun des neufs modes d'optimisation s'est effectuée sur les trente scénarios fournis par le CELAR. Ce dernier a fixé 3 exigences :

EXG-1. Dix scénarios comportant moins de 50 variables sur 80% desquels le titulaire devra fournir la meilleure solution connue en moins de six heures.

EXG-2. Dix scénarios pour lesquels une solution optimale est connue, sur 80% desquels le titulaire devra fournir une solution d'évaluation égale à la solution optimale en un temps fixé au niveau de chaque scénario.

EXG-3. Dix scénarios inspirés de cas réels sur lesquels le titulaire devra apporter une solution à moins de 10% de la meilleure solution connue en un temps fixé au niveau de chaque scénario.

Le Tableau 26 montre l'évaluation du module fourni selon les exigences définis par le CELAR. La valeur 100% signifie que le module fourni a réussi à respecter, et éventuellement à dépasser, les résultats attendus (problème de satisfaction de contraintes !). La méthode *Tabu-NG* a montré donc de très bonnes performances sur l'ensemble des instances et des objectifs quelles que soient la taille des problèmes et la nature des contraintes.

	Réalisabilité	***MinFreq***	***MaxFreqG***	***MinFreqD***
EXG-1	100%	90%	100%	100%
EXG-2	100%	80%	100%	90%
EXG-3	100%	90%	100%	100%

	MinSpec	***MinEcart***	***MinMob***	***MinCard***	***MinMax***
EXG-1	90%	100%	100%	100%	100%
EXG-2	80%	100%	80%	90%	80%
EXG-3	90%	100%	100%	100%	90%

Tableau 26 : Synthèse des résultats obtenus sur les instances privées fournies par le CELAR

3.5. Synthèse

Ce chapitre a présenté l'application de la méthode *Tabu-NG* sur des problèmes d'affectation de fréquences pour des réseaux de radiocommunications militaires. Il est composé de quatre parties dont la présentation de la formalisation du problème et des instances, des modèles théoriques et applicatifs de référence, l'application de la méthode sur des problèmes d'affectation de fréquences classiques puis l'application de la méthode sur les

problèmes d'affectation de fréquences avec polarisation et contraintes n-aires. L'affectation de fréquences a constitué une plateforme réelle d'expérimentation pour *Tabu-NG*. Nous tenons à souligner qu'une partie de ce travail a été réalisée dans le cadre d'un contrat de recherche avec le CELAR.

Dans la première partie, le problème d'affectation de fréquences est expliqué tel qu'il nous été proposé par le CELAR. Il s'agit d'un problème spécifique dont l'objectif est d'allouer à chaque trajet une fréquence et une polarisation en respectant un ensemble de contraintes unaires, binaires et n-aires et en ayant un ensemble de critères à optimiser. Les critères à optimiser se classifient en deux familles, la première est l'optimisation de l'utilisation du spectre fréquentiel, et la deuxième est la minimisation de la somme pondérée des contraintes violées.

Nous avons étudié dans la littérature les problèmes qui se rapprochent le plus de ce nouveau problème. Les modèles de T-coloration ont pour objectif d'allouer une valeur à des variables en respectant une contrainte d'écart entre les valeurs allouées. Ces types de problèmes modélisent différentes applications d'affectation de fréquences liés à des différents systèmes de radiocommunications. Cependant, qu'en problème a trois grandes particularités. D'une part, les interférences ne se traduisent pas en contraintes d'écartement de fréquences comme en T-Coloration, à cause de la présence de nouveau type de contraintes binaires et n-aires. D'autre part, il faut aussi allouer une polarisation à chaque variable. Enfin, les critères à optimiser sont très variés, de la recherche d'une solution satisfaisant l'ensemble des contraintes du problème et optimisant l'emploi du spectre, à la recherche d'une solution de compromis qui minimise la somme pondérée des contraintes violées. Le modèle le plus proche est la T-coloration d'hypergraphe, cependant très peu d'études existent sur ce modèle et les résultats sont récents. Nous avons donc choisi finalement de représenter le problème sous la forme d'un *CSP*, qui est un modèle abstrait qui peut représenter n'importe quel problème d'affectation de ressources. Cette modélisation nous a permis d'utiliser des techniques et des principes déjà développées et testées dans la littérature sur ce modèle.

La troisième partie de ce chapitre contient une partie de notre contribution sur ce problème. Elle explique d'abord comment la méthode *Tabu-NG* a été appliquée sur des problèmes d'affectation de fréquences classiques avec uniquement des contraintes binaires. Le schéma général de la méthode est resté identique à celui proposé dans le premier chapitre, nous y avons cependant ajouté des techniques de filtrage et d'identification de *nogoods* spécifiques au problème posé, ainsi que des heuristiques comme cela se fait pour toute méthode générale. Les heuristiques ont été utilisées dans les phases d'extension et de réparation d'une configuration partielle courante. Les *nogoods* identifiés sont exacts mais ne sont pas minimaux. Une notion de dominance a été introduite afin d'éviter le stockage des *nogoods* inutiles et la croissance rapide de l'occupation de la mémoire. Des stratégies d'optimisation de spectre ont été proposées et testées aussi dans cette partie. Les résultats sont au niveau des meilleurs connus dans la littérature sur des problèmes benchmarks.

La quatrième partie de ce chapitre contient la deuxième partie de notre contribution sur ce problème. Elle explique comment la méthode *Tabu-NG* a été appliquée sur des problèmes d'affectation de fréquences avec polarisation et contraintes n-aires. Le schéma général est resté identique à celui proposé dans le 1er chapitre, cependant les techniques de filtrage ont été adaptées pour supporter tous les types de contraintes traités (unaires, binaires et n-aires). Nous avons aussi introduit la notion de *nogood* partiel et la façon dont on identifie et traite ce type de *nogood*. Nous avons aussi présenté des mécanismes de diversification que nous avons ajoutés à *Tabu-NG* tels que le redémarrage de la recherche et la suppression de la liste des *nogoods* archivés. Dans la section de résultats, nous avons montré que la méthode *Tabu-NG*

avec les *nogoods* partiels donne de très bons résultats et que ces *nogoods* partiels guident efficacement la méthode vers les zones prometteuses de l'espace de recherche. Nous avons vu aussi que l'identification des *nogoods* partiels tel que nous l'avons proposé, n'est pas complexe et que la qualité des *nogoods* identifiés est acceptable. Pour les problèmes que nous avons traités, nous avons vu que la taille de la liste des *nogoods* était relativement stable et petite durant la recherche.

Le chapitre se conclut par un résumé sur les résultats de *Tabu-NG* sur les instances fournies par le CELAR. L'évaluation de *Tabu-NG* a confirmé le bon comportement et l'efficacité de cette méthode pour ce problème.

4. *Tabu-NG* et coloration de graphe

Ce chapitre présente les études menées pour l'application de la méthode *Tabu-NG* sur les problèmes de *k*-coloration de graphe. Notre objectif a été d'étudier l'applicabilité de la méthode et ses performances sur un problème connu fournissant une bonne base comparative avec d'autres travaux.

Tout d'abord, les principes de la *k*-coloration seront abordés. Ensuite nous présentons la similitude entre le modèle CSP et la *k*-coloration de graphe. Nous décrivons par la suite l'ensemble des instances utilisées.

La deuxième partie de ce chapitre décrit les différentes études menées sur les problèmes de coloration de graphe. Nous analysons dans un premier temps les points de différence entre la coloration de graphe et l'affectation de fréquences, et la nécessité de l'adaptation de quelques implémentations pour la coloration de graphe. Nous donnons l'algorithme de filtrage choisi ainsi que les structures de données utilisées pour ce problème. Nous discutons aussi de l'utilité du stockage des *nogoods* pendant la résolution de ce type de problèmes. Une analyse et une comparaison des performances sont données pour justifier les choix que nous avons adoptés.

La troisième partie traite de l'application de la méthode *Tabu-NG* à la coloration. Nous avons travaillé sur les heuristiques d'extensions de la configuration partielle, de réparation de la configuration partielle lorsqu'elle est inconsistante et sur la durée Tabou. Nous avons effectué plusieurs tests et analyses.

La dernière partie de ce chapitre présente les travaux faits sur le réglage de *Tabu-NG* et notamment le choix des heuristiques d'extension et de réparation des configurations partielles ainsi que la nécessité de diversifier la recherche à un stade donné pendant le déroulement de la méthode. Les résultats obtenus sur l'ensemble des instances choisies sont donnés.

A la fin de ce chapitre, nous présentons une comparaison des résultats de la méthode *Tabu-NG* avec ceux d'autres méthodes connues de la littérature.

Sommaire

4. *Tabu-NG* et coloration de graphe 142

4.1. Coloration de graphe 143
4.2. Implémentations pour la coloration de graphe.....151
4.3. Méthode *Tabu-NG* pour la coloration de graphe 160
4.4. Réglages des actions d'intensification et de diversification 166
4.5. Comparaison des résultats obtenus avec la littérature 172
4.6. Synthèse 174

4.1. Coloration de graphe

4.1.1. Contexte

En recherche opérationnelle, le problème de coloriage de graphe fait partie des problèmes de référence. Il s'agit de déterminer un nombre de couleurs suffisant pour colorier les sommets d'un graphe donné de telle façon que deux nœuds adjacents (liés par une contrainte) n'aient pas la même couleur. Une des applications phares dans le domaine de coloriage de graphe est le problème de coloriage des cartes géographiques qui remonte au milieu du XIX^{e} siècle. Il s'agit de colorier des zones représentées sur une carte géographique de telle manière que deux zones voisines ne soient pas coloriées de la même couleur. Le passage de la représentation cartographique à une représentation en graphe se fait en remplaçant les zones par des nœuds et en reliant par des arêtes les zones connexes. Depuis les travaux de Francis Guthrie en 1852 sur la coloration des régions d'Angleterre il est prouvé que l'utilisation de 4 couleurs suffit à la coloration de n'importe quelle carte découpée en régions. Ce théorème, connu aussi sous l'appellation du théorème des quatre couleurs, s'étend à tout graphe de type planaire [144], i.e. un graphe qui peut être représenté par des arêtes droites qui ne se croisent pas.

La coloration de graphes est aussi à la base de la modélisation de beaucoup de problèmes réels notamment issus de domaines technologiques tels que les problèmes d'affectations de ressources (affectation de fréquences dans les réseaux de communications, affectation de registres dans les compilateurs…) [145], d'ordonnancement [155], de conception d'emplois du temps [146], d'optimisation de chaines logistiques [156], etc. La coloration de graphes a aussi été utilisée pour modéliser et résoudre des problèmes connus en mathématiques et statistiques [147][148][149]. C'est grâce à la puissance et à la facilité des principes de modélisation par la coloration de graphes que ce domaine est toujours aussi vivant et dynamique.

Afin de s'adapter aux spécificités de ces applications, différentes variantes du problème de coloration de graphe ont vu le jour [150]. Nous nous focaliserons dans notre étude sur la problématique de *k*-coloration de graphe.

Avant d'aborder la résolution de ce type de problèmes avec la méthode *Tabu-NG* nous rappelons quelques définitions. Dans un premier temps, nous définissons des notions propres à la théorie des graphes suivies de quelques notions sur la coloration et la définition formelle de ce problème. Pour finir, les différents jeux de tests utilisés dans ce chapitre seront présentés.

4.1.2. Définition formelle du problème et quelques notions

Définition 31 : Graphe non orienté

Un problème de coloriage de graphe est défini par un graphe non orienté $G = (V, E)$ où V représente l'ensemble des nœuds et $E \subset V \times V$ l'ensemble des arêtes du graphe. Les graphes que nous traitons sont des graphes non orientés. Nous noterons $N = |V|$ le nombre de nœuds.

Définition 32 : Degré d'un nœud

Le degré d_x d'un nœud $x \in V$ désigne le nombre de voisins directs du nœud x. Autrement dit le nombre de nœuds directement reliés par une arête à x.

Un nœud x dont le degré $d_x=0$ est dit isolé.

- Le nœud de degré minimal (respectivement maximal) est noté d_{min} (respectivement d_{max}).
- Le degré moyen d'un graphe est donné par l'équation :

$$d(G) = \frac{1}{|V|}\sum_{v \in V} d(v) \qquad \textbf{Equation 1}$$

Définition 33 : Clique

Une clique est un sous-ensemble de nœuds, $V' \subset V$, deux-à-deux adjacents.

- On note *p-clique* une clique avec p nœuds.
- La plus grande clique (clique composée du plus grand nombre de nœuds) est appelée *clique-maximale*.

Définition 34 : Graphe complet

Un graphe complet est une *p-clique* avec $p = |V| = N$, elle contient $\left(\frac{N(N-1)}{2}\right)$ arêtes. Le degré d'un nœud quelconque dans un graphe complet est égal à N-1.

Définition 35 : Voisinage d'un nœud

Le voisinage d'un nœud x dans un graphe, noté $v(x)$ désigne l'ensemble des nœuds liés à x par une arête. Dans un graphe complet, le voisinage d'un nœud x est $V \backslash \{x\}$.

Définition 36 : Densité du graphe

La densité d'un graphe représente la probabilité qu'il y ait une arête entre deux nœuds distincts quelconques du graphe. La densité Δ est calculée de la façon suivante :

$$\Delta = \frac{|E|}{\frac{N(N-1)}{2}} \qquad \textbf{Equation 2}$$

Notons que la densité d'un graphe complet est égale à 1.

Définition 37 : Coloration

Soit Γ l'ensemble des couleurs disponibles pour la coloration d'un graphe. Une coloration est une relation $\varphi : V \mapsto \Gamma$ qui affecte une couleur $\varphi(x)$ unique à chaque nœud *x* du graphe.

Définition 38 : Coloration partielle

Une coloration partielle est une relation $\varphi : V' \mapsto \Gamma$, avec $V' \subset V$, qui affecte une couleur unique à un sous-ensemble de nœuds du graphe.

Définition 39 : Conflit

L'affectation de la même couleur à deux nœuds voisins dans un graphe donné est appelé conflit.

Définition 40 : Coloration valide

Une coloration φ est dite valide si et seulement si elle n'engendre aucun conflit. Autrement dit, deux nœuds voisins reliés par une arête sont coloriés par deux couleurs différentes.

$$\forall (x, y) \in E, \varphi(x) \neq \varphi(y) \qquad \textbf{Equation 3}$$

Un graphe est dit *k-coloriable*, s'il existe une coloration valide utilisant *k* couleurs différentes.

Définition 41 : Coloration partielle valide ou consistante

Une coloration partielle φ est dite valide ou consistante si et seulement si l'ensemble des nœuds coloriés n'engendre aucun conflit.

$$\forall (x, y) \in E / x, y \in V', \varphi(x) \neq \varphi(y) \qquad \textbf{Equation 4}$$

Définition 42 : Problème de *k*-coloration de graphe

Le problème de *k*-coloration de graphe consiste à trouver une coloration valide du graphe *G* utilisant un nombre de couleur égale à $k = |\Gamma|$

Définition 43 : Nombre chromatique

Le nombre chromatique χ d'un graphe *G* est le nombre de couleurs minimum permettant une coloration complète et valide du graphe *G*.

4.1.3. CSP et coloration de graphe

Le problème de coloration de graphe se modélise naturellement en un problème de satisfaction de contraintes *CSP*. Nous associons des variables à des domaines finis aux nœuds du graphe et des contraintes de différence à chaque arête. Le *CSP* équivalent sera : $X=\{x_1, x_2, ..., x_n\}$ est un ensemble de n variables.

D est la fonction qui associe à chaque variable x_i son domaine $D(x_i)=\Gamma$, i.e. l'ensemble des couleurs que peut prendre x_i.

$C = \{c_1, c_2, ..., c_m\}$ est un ensemble de m contraintes. Chaque contrainte c_i est une contrainte de différence qui porte sur deux variables voisines.

$$\forall c_i \in C, \forall (x,y) \in c_i, \varphi(x) \neq \varphi(y) \qquad \textbf{Equation 5}$$

Exemple : Soit le graphe *G(V, E)* figurant dans la Figure 23 avec 5 nœuds (*A, B, C, D* et *E*), 6 arcs *{(A, B) (A, C) (A, D) (C, D) (B, D) (D, E)}* et 3 couleurs possibles pour colorier le graphe $\Gamma = \{Rouge, Vert, Bleu\}$

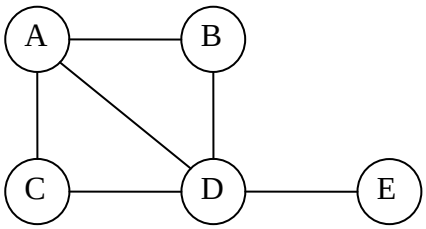


Figure 23 : Un graphe *G(V, E)*

Le CSP équivalent s'écrit de la façon suivante :

- Ensemble des variables : $X = \{A, B, C, D, E\}$
- Ensemble des domaines : $D(A)=D(B)=D(C)=D(D)=D(E)=$ *{Rouge, Vert, Bleu}*. Les couleurs sont souvent référencées par un simple numéro comme par exemple {1, 2, 3}, 1 pour le rouge, 2 pour le vert et 3 pour le bleu.
- Ensemble des contraintes : $C = \{c_1, c_2, ..., c_6\}$ avec $c_1 : \varphi(A) \neq \varphi(B)$, $c_2 : \varphi(A) \neq \varphi(C)$, $c_3 : \varphi(A) \neq \varphi(D)$, $c_4 : \varphi(C) \neq \varphi(D)$, $c_5 : \varphi(B) \neq \varphi(D)$, $c_6 : \varphi(D) \neq \varphi(E)$.

La Figure 24 représente une coloration partielle valide du graphe. Cette configuration est extensible vers une coloration valide (solution réalisable) comme illustré en Figure 25.

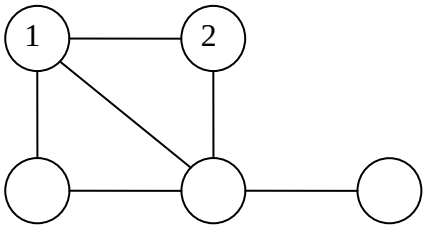


Figure 24 : Coloration partielle consistante du graphe *G(V, E)*

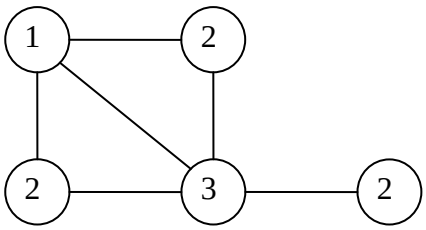


Figure 25 : Coloration valide du graphe *G(V, E)*

4.1.4. Benchmarks utilisés

Au cours de cette étude, nous avons utilisé différents types de graphe pour traiter les problèmes de *k*-coloration. Cette partie présente les principales caractéristiques des instances DIMACS (*Discrete Mathematics and Theoretical Computer Science*) et d'autres instances utilisées durant cette thèse pour évaluer la méthode *Tabu-NG*.

4.1.4.1. Les graphes DIMACS

Les instances DIMACS réalisées en 1992-1993 sont très utilisées dans la littérature. Elles regroupent un ensemble très important de graphes de différentes natures, structures, tailles et difficultés. Nous décrivons, dans cette partie, les graphes que nous avons utilisés.

Graphes Aléatoires : Créés par David Johnson en 1991 [153], les graphes aléatoires sont nommés par *DSJC n.p* où *n* est le nombre de nœuds et la valeur *p* représente le dixième de la probabilité que deux nœuds soient liés par une arête. Par exemple, l'instance *DSJC1000.9* correspond à un graphe composé de 1000 nœuds et une probabilité d'existence d'une arête entre deux nœuds égale à 90%. Le nombre chromatique de ces graphes n'est pas connu.

Graphes Leighton : Générées par Leighton en 1979 [154], les instances Leighton sont des graphes aléatoires générés en fixant le nombre chromatique χ et en introduisant des cliques de taille 2 à χ dans le graphe. Tous les graphes Leighton utilisent le même nombre de nœuds. Les graphes sont nommés de la façon suivante : $Le450_\chi l$, avec 450 le nombre de nœuds, χ le nombre chromatique et une lettre $l \in \{a,b,c,d\}$utilisée pour distinguer les différents graphes de même nombre chromatique.

Graphes Plats : Les graphes plats sont des graphes quasi-aléatoires générés par Culberson [152]. Ils sont notés de la façon suivante : $flatn_\chi_0$ avec *n* le nombre de nœuds et χ le nombre chromatique.

Graphes REG : Les graphes REG [159] sont des instances qui correspondent à des problèmes d'allocation de registres sur code réel. Le compilateur souhaite affecter les variables à des registres. Deux variables peuvent être affectées au même registre si elles ne sont pas appelées en même temps dans une partie du code. Les instances de type *mulsol*, *zeroin*, *fpsol2*, et *inithx* ont été créées à l'occasion de la conférence DIMACS en 1994.

Graphes School : Les graphes *school* [159] correspondent à un problème d'emploi du temps scolaire, avec ou sans gestion des salles de classe.

Graphes Myciel : Ces graphes ne contiennent pas de clique de tailles supérieure à 2 mais leur nombre chromatique augmente avec la taille du graphe. Ainsi un graphe de type *Mycielk* aura un nombre chromatique égal à $k + 1$.

Graphes Full-Insertions: Ce sont des graphes obtenus en appliquant une transformation sur un graphe initial de n nœuds pour obtenir une succession de graphes [160]. Le mécanisme se base sur l'insertion de $n(K+1)+K+2$ sommets dans chaque transformation. Pour un graphe *K-FullInsertion-I*, avec K et I deux paramètres qui déterminent la transformation, il n'existe pas de preuve concernant leurs nombres chromatiques.

Graphe de carré latin : Un seul graphe de carré latin (*latin_square_10*) [159] est utilisé. Le nombre chromatique de ce graphe n'est pas connu.

4.1.4.2. Autres graphes

Les graphes *bookgraphs* [161] sont des graphes de livres qui se basent sur les réseaux sociaux. Par exemple pour un livre de littérature, un personnage sera représenté par un sommet, deux personnages seront liés par un arc s'ils se rencontrent dans le livre… Cinq graphes de ce type ont été générés : Anna Karénine (anna), David Copperfield (david), L'Iliade (homer), Huckleberry Finn (huck) et Les Misérables (jean).

Les graphes de jeux *game graphs* [161] représentent des matchs joués entre les différentes équipes lors d'une saison de football américaine. Une équipe est représentée par un sommet. Deux sommets sont liés par un arc, si les équipes correspondantes ont joué l'une contre l'autre pendant la saison.

Les graphes de type *Mile* [161] sont similaires à des graphes géométriques. Chaque sommet représente un point géographique. Deux sommets sont liés si les points correspondants sont suffisamment proches l'un de l'autre.

4.1.5. Méthodes de résolution

La littérature sur les problèmes de coloration de graphe est très riche, plusieurs milliers de publications portent sur le traitement de ces problèmes. La coloration de graphe est l'un des problèmes les plus étudiés en optimisation combinatoire. Le lecteur intéressé peut se référer à la bibliographie faite par Marco Chiarandini (plus de 200 références) qui est mise à jour régulièrement [158], ou bien à l'article de Galinnier et Hertz [157].

Plusieurs méthodes ont été proposées pour résoudre les problèmes de coloriage de graphe. Les méthodes exactes de résolution ont montré leurs limitations ; [163] et [164] ont prouvé qu'on peut résoudre des graphes aléatoires de 90 nœuds et des graphes géométriques de 250 nœuds dans un temps raisonnable. Le meilleur algorithme exact selon les résultats sur les instances DIMACS est celui de Caramia et Dell'Olmo [165] qui a pu résoudre quelques graphes de 500 nœuds en 15 minutes, c'est un algorithme de *Branch&Bound*. Les algorithmes exacts restent inefficaces sur des problèmes de coloriage de graphe de grandes tailles.

Dans cette section, nous présentons les meilleurs algorithmes provenant de 3 classes de méthodes, les méthodes de recherche locale, les méthodes à base de programmation

par contraintes et les méthodes hybrides (hybridation de méthodes incomplètes). La méthode *Tabu-NG* sera évaluée par rapport aux résultats provenant des approches présentées dans cette section. A noté que chaque approche présente des résultats soit pour un algorithme unique, soit pour plusieurs algorithmes associés en combinant les performances de plusieurs algorithmes ou, simplement les performances de plusieurs variantes du même algorithme en changeant par exemple des paramètres.

4.1.5.1. Méthodes de recherche locale

Nous présentons dans cette section quatre méthodes de recherche locale qui ont montré une grande efficacité dans la résolution des problèmes de coloriage de graphe et particulièrement les instances DIMACS.

ILS 2002 : La recherche locale réitérée ILS (*Iterated Local Search*), proposée dans [167], est une méthode combinant une Recherche Tabou et une procédure de perturbation. Une Recherche Tabou est exécutée jusqu'à ce que la meilleure solution trouvée reste inchangée durant un nombre fixé d'itérations. Une perturbation est alors appliquée à cette solution et la Recherche Tabou est à nouveau exécutée sur la solution perturbée.

GLS 2005 : La recherche locale guidée GLS (*Guided Local Search*) est basée sur la modification de la fonction de coût pour échapper aux optimums locaux. Une application de cette méthode est proposée par Chiarandini [166].

ALS 2008 : La recherche locale adaptative ALS (*Adaptative Local Search*), proposée dans [104][105], est une méthode de recherche locale qui alterne des phases d'intensification et de diversification en se basant sur une procédure de détection de boucles. Une boucle correspond à l'apparition répétitive d'une même valeur d'une variable. Pour cela, ALS fonctionne en mode intensification où l'amélioration de la solution courante est privilégiée. Lorsqu'une boucle est détectée, ALS lance une phase de diversification permettant d'élargir le voisinage explorée de la solution courante.

RCTS 2009 : La Recherche Tabou renforcée de coloration RCTS (*Reinforced Coloring Tabu Search*), proposée dans [168], est une amélioration des algorithmes Tabou classiques en utilisant deux nouvelles fonctions d'évaluation et une liste Tabou réactive. Les nouvelles fonctions d'évaluation proposées enrichissent la fonction d'évaluation conventionnelle f en prenant en considération des informations supplémentaires relatives à la structure du graphe (degrés de conflit des sommets) ainsi que des informations dynamiques acquises au cours de la recherche (la fréquence de changement de couleur). De plus, la gestion réactive de la liste Tabou permet à RCTS d'éviter le bouclage ; ainsi, elle peut effectivement profiter d'un temps de calcul plus long.

4.1.5.2. Méthodes à base de programmation par contraintes

A notre connaissance, la programmation par contraintes n'a pas montré une grande efficacité sur les problèmes DIMACS que nous traitons dans cette thèse. En exemple, la méthode incomplète noté IDB (*Incomplete Dynamic Backtracking*) [172] basée sur l'algorithme DBT (*Dynamic Backtracking*) a obtenu des scores très loin des meilleurs scores connus. Nous avons choisi de présenter dans cette section une méthode notée

FCNS [19] qui se base sur la PPC en la combinant avec la recherche locale. FCNS est une hybridation intéressante, elle est bien meilleure qu'IDB sur les instances DIMACS.

FCNS 2002 : FCNS (*Forward Checking with Consistent Partial Colorations)* est une méthode de résolution proposée dans [162]. Elle est basée sur une hybridation du type de celle de *Tabu-NG* en combinant des mécanismes issus de la programmation par contraintes et de recherche locale. FCNS manipule des configurations partielles. L'idée est d'affecter une couleur valide à une variable libre afin d'étendre une configuration partielle. Après chaque instanciation, la propagation de contraintes est appliquée sur les variables voisines de la dernière variable instanciée. Lorsque la variable sélectionnée ne peut pas être coloriée tout en gardant la consistance (cas de *deadend*), FCNS sélectionne parmi la liste des variables coloriées, et en utilisant une heuristique prédéfinie, une variable à désinstanciée. De cette manière la recherche est non systématique et la complétude de la méthode n'est pas assurée. FCNS utilise un paramètre *B* qui varie pendant la recherche comme la température en recuit simulé. Chaque fois qu'un *deadend* est atteint, *B* variables seront libérées. Plusieurs heuristiques ont été testées après la découverte d'un *deadend*. Dans la partie dédiée à la comparaison des résultats, nous dressons les meilleurs scores obtenus par FCNS quelle que soit l'heuristique utilisée.

4.1.5.3. Méthodes hybrides

Dans cette section, nous présentons quelques méthodes hybrides qui ont montrées de très bonne performance sur la résolution des problèmes de coloration de graphe et en particulier sur les instances DIMACS. On entend par hybridation dans cette section, le fait de faire collaborer des approches incomplètes entre elles, telles que Recherche Tabou et algorithme génétique.

DCNS 1996 : *Distributed Coloration Neighborhood Search* [169], une large collection de techniques de résolution, y compris des méthodes distribuées, des algorithmes Metropolis, et un codage à base de colorations légales partielles. Les résultats reportés de DCNS dans cette thèse pour la comparaison sont les meilleurs obtenus par toutes les méthodes incluses dans DCNS.

AMACol 2008 : est une méthode hybride combinant une méthode à base de population et une Recherche Tabou. Cette méthode, appelée AMACOL (*Adaptive Memory Algorithm for k-COLoring*), est proposée par Galinier et al. [170]. Elle a permis pour la plupart des instances d'obtenir le plus petit nombre de couleurs connu *k* ; elle est classée parmi les meilleures méthodes de coloration actuelles. Cette méthode est basée sur l'utilisation d'une mémoire centrale de solutions composées de classes de couleur stables maximales. Dans un premier temps, un ensemble de solutions aléatoires est généré puis la méthode Tabucol [74] est appliquée pendant un nombre fixé d'itérations. Sur ces solutions, on applique un opérateur permettant de générer des classes de couleur stables maximales. Une classe de couleur rassemble tous les nœuds de même couleur au sein d'une solution. On appelle classe de couleur stable un ensemble de nœuds non-reliés deux à deux de même couleur. L'opérateur a donc pour objectif de créer des solutions utilisant des classes de couleur stables de taille maximale, c'est à dire regroupant le plus grand nombre de nœuds.

MACol 2010 : *Memetic Algorithm for Graph Coloring* (ou MemCol) [171] est un algorithme mimétique (méthode évolutionniste incorporant une recherche locale) très récent avec un opérateur de croisement adaptatif multi-parent et un contrôle efficace de

balance diversification/intensification. Cet algorithme a donné de très bons résultats et a amélioré certains scores connus.

4.2. Implémentations pour la coloration de graphe

Dans cette section, nous présentons les adaptations des implémentations faites initialement pour l'affectation de fréquences au contexte de la coloration de graphes. Nous justifions nos choix au regard de lenteurs excessives de l'algorithme en termes de nombre d'itérations effectué indépendamment des performances en termes de résultats, bien que les deux éléments soient bien entendu intrinsèquement liés. Cependant il ne s'agit pas d'évolution des principes de la méthode *Tabu-NG* mais de détails d'implémentations de certaines parties de la procédure qui permettent de la rendre plus rapide.

4.2.1. Coloration de graphe versus affectation de fréquences discrètes

En dehors des différences liées à l'affectation de deux ressources simultanées pour le FAP, la coloration de graphe se distingue par le fait d'utiliser exclusivement des contraintes binaires de différence et un domaine identique pour toutes les variables ; par ailleurs il y a aussi quelques points particuliers qui ont une incidence forte sur la recherche.

1. Les symétries par permutation. L'espace de recherche d'un problème de coloriage de graphe est composé de plusieurs zones symétriques obtenues par permutation des couleurs. Par exemple, pour colorier une clique de 5 nœuds, on a besoin de 5 couleurs quel que soit l'ordre d'attribution de ces couleurs aux nœuds du problème. Il existe donc un grand nombre de solutions équivalentes et par la même de configurations invalides équivalentes.
2. La nature de l'arc-consistance. Pour une contrainte de différence donnée c_{ij}, l'arc-consistance est assurée naturellement dès lors que les domaines de deux variables i et j contiennent plus d'une valeur. Il convient de concevoir un algorithme de filtrage adapté à ce cas de figure pour rendre la propagation efficace. Il est par exemple inutile de parcourir le domaine d'une variable pour vérifier la consistance d'un arc si la taille de son domaine est supérieure à 1.
3. La propagation des affectations. Dans un graphe de contraintes où toutes les contraintes sont des contraintes de différence, l'effet de l'instanciation d'une variable x ne se propage souvent que sur les variables voisines de x. D'où la nécessité d'utiliser un algorithme de filtrage de type *Forward Checking* afin d'accélérer la propagation de contraintes contrairement à l'affectation de fréquences ou on utilise un algorithme beaucoup plus lourd et complexe pour assurer l'arc-consistance du graphe (AC-3).

Dans la partie suivante, nous détaillons les composantes de *Tabu-NG* dont nous avons adaptées l'implémentation pour une recherche plus rapide dans le contexte du coloriage de graphe.

4.2.2. Adaptation de l'algorithme de filtrage et de propagation de contraintes

Dans cette section, nous présentons la variante de la procédure de propagation associée au problème de coloration que nous avons introduite dans *Tabu-NG*. Comme pour toute méthode générale, l'introduction d'heuristique spécifique permet de bénéficier de la nature de ce nouveau problème afin d'accélérer la recherche.

Dans un premier temps, nous avons appliqué *Tabu-NG* telle qu'implémentée pour les problèmes d'affectation de fréquences sur le coloriage de graphe. Ceci était nécessaire pour identifier les composantes de la méthode adaptées à la problématique de coloration et celles qui au contraire nécessitaient une conception différente. Nous avons donc formaté quelques instances de coloriage de graphe au format de l'affectation de fréquences. Les contraintes de différences, de type $coul(x) \neq coul(y)$ avec x et y deux nœuds voisins du graphe à colorier, sont transformées en contraintes d'écart absolu présentées dans le chapitre précédent du type $|coul(x) - coul(y)| > 0$. La variante de la méthode *Tabu-NG* avec les *nogoods* partiels a été appliquée au problème de coloration.

Les premiers tests effectués ont révélé une grande lenteur de la méthode dans le calcul ; la durée d'un cycle extension/propagation/réparation était très long par rapport au FAP. La première cause de cette lenteur a été identifiée comme étant la procédure de propagation de contraintes. La procédure de base utilisée pour l'affectation de fréquences possède une complexité élevée. Cette complexité reste raisonnable dans le cas de l'affectation de fréquences car le résultat de la propagation des contraintes est assez perceptible en termes de valeurs éliminées du fait en particulier des contraintes de type T-coloration. Cette complexité est cependant non négligeable pour la coloration car l'effet d'une extension de la solution est faible sur le reste des variables ; on calcule beaucoup plus que l'on ne propage. Une valeur v dans le domaine d'une variable x ne sera éliminée que si au moins un domaine d'une des variables voisines de x se réduit à un singleton $\{v\}$.

Nous avons donc construit un nouvel algorithme de propagation plus efficace. Le nouvel algorithme de filtrage (Algorithme 26) qui propage l'effet de l'affectation d'une valeur v à une variable x, rend en sortie directement l'ensemble des variables bloquées c'est-à-dire toutes les variables dont le domaine est devenu vide après la propagation de la dernière affectation. Il est beaucoup plus simple mais beaucoup plus adapté au problème. La fonction *PropagerAffectation* élimine dans un premier temps les valeurs v de tous les domaines des variables non instanciées voisines de x. Si un blocage (*deadend*) survient, la fonction ajoute directement la variable bloquée à la liste des variables bloquées. Si un domaine d'une variable y voisine de x devient singleton, alors on affecte la valeur w restante dans le domaine de y et on lance récursivement la procédure *PropagerAffectation* pour l'affectation de la valeur w à la variable y.

L'algorithme *PropagerAffectation* identifie un *nogood* de la même façon que pour l'affectation de fréquences. La nouvelle instanciation (x, v) se propage sur les variables voisines de x en supprimant la valeur v dans le domaine d'une variable opposée y, on ajoute ensuite cette instanciation à l'ensemble *ElExpl* de la variable y. Ceci signifie que l'instanciation (x, v) a participé à l'élimination d'au moins une valeur dans le domaine de y. Dans le cas d'un *deadend*, le *nogood* associé sera constitué de l'ensemble *ElExpl* de la variable bloquée.

```
BlockedVars : une liste globale qui contiendra les variables bloquées

Function PropagerAffectation (x, v)
1.  Bool isDeadEnd = false ;
2.  foreach uninstanciated variable y in v(x) do
3.      if Remove v from Dom(y) == true then ElExpl(y) += (x, v);
4.          if |Dom(y)| == 0 then
5.              isDeadEnd = true;
6.              BlockedVars += y;
7.          endif
8.      endforeach;
9.      if isDeadEnd == true return true; //au moins une variable est bloquée
10.     foreach uninstanciated variable y in v(x) do
11.         if |Dom(y)| == 1 then
12.             w = la seule valeur restante dans Dom(y)
13.             if PropagerAffectation(y, w) == true return true ;
14.         endif
15.     endForeach
16. return false ;//aucune variable n'est bloquée
17. end function
```

Algorithme 26 : Propagation d'une instanciation dans un problème de coloriage de graphe

Nous comparons maintenant les performances de la méthode initiale de propagation de contraintes telle que développée pour l'affectation de fréquences avec la procédure adaptée au problème de coloriage de graphe. Le reste de *Tabu-NG* est inchangé. Pour cela, nous avons considéré les problèmes de *k*-coloration de graphe Leighton. Sur chaque instance de problème nous avons lancé les deux versions de l'algorithme, version FAP et version CG1.

Tableau 27 indique sur chaque ligne le nom du problème, le nombre de couleurs *k* utilisées pour colorier le graphe, le temps de calcul fixé en secondes et le nombre d'itérations effectuées par les deux versions de *Tabu-NG* au cours de ce temps. De part la nature déterministe de *Tabu-NG* nous savons que les deux versions donneront à la fin le même résultat en termes d'établissement de consistance du graphe, de suppression de valeurs inconsistante... Nous voulons uniquement mesurer l'impact de la complexité des algorithmes de propagation sur le problème de coloration sur un temps donné, ici 5 secondes. A noter qu'aucun problème n'est résolu en 5 secondes par les deux méthodes. Le tableau montre que la version CG1 est toujours plus rapide, jusqu'à 10 fois plus sur ces tests ; donc pour le même temps de recherche le nombre de configurations visitées est beaucoup plus grand, ce qui a bien sur un impact sur la performance de l'algorithme pour la coloration.

Nom	*k*	T(s)	Iter. *Tabu-NG* : FAP	Iter. *Tabu-NG* : CG1
le450_15c	15	5	1073	3821
le450_15d	15	5	1043	4293
le450_25c	25	5	1009	4372
le450_25d	25	5	1015	4533
le450_5a	4	5	2222	13913
le450_5b	4	5	2067	13651
le450_5c	4	5	1319	10059
le450_5d	4	5	1254	12347

Tableau 27 : Gain de temps sur la propagation de contraintes

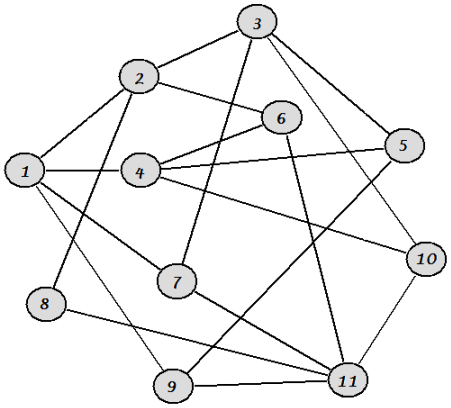
4.2.3. Adaptation des structures de données

Même avec le gain de temps sur la propagation de contraintes, la méthode reste anormalement lente sur la durée d'un cycle extension/propagation/réparation par rapport au FAP. Les structures de données conçues pour un traitement rapide des réparations sur une configuration partielle de FAP ne sont pas adaptées à la CG. En FAP, après la réparation il était nécessaire de restaurer tous les domaines initiaux et de propager à nouveau les instanciations depuis le début du fait de la nature complexe des contraintes et de l'effet de la propagation par *look-ahead*. Dans cette partie nous expliquons les structures de données développées spécifiquement pour la coloration et les mécanismes associés pour réparer une configuration partielle et remettre les domaines en état plus rapidement.

4.2.3.1. Représentation des contraintes

L'ensemble des contraintes d'un problème est présenté sous forme d'un tableau de listes. Chaque ligne *i* du tableau pointe sur la liste de toutes les variables voisines de la variable *i*.

Soit le graphe *G(V, E)* qui représente le problème de coloriage de graphe nommé *Myciel3* et son tableau de contraintes donné en Figure 26. La ligne 1 du tableau des contraintes signifie que la variable 1 a comme voisins les variables 2, 4, 7 et 9. Cette représentation a pour avantage une grande simplicité d'implémentation avec un besoin de mémoire de stockage raisonnable et un accès très rapide aux variables voisines d'une variable donnée.



1	2	4	7	9	
2	1	3	6	8	
3	2	5	7	10	
4	1	5	6	10	
5	3	4	8	9	
6	2	4	11		
7	1	3	11		
8	2	5	11		
9	1	5	11		
10	3	4	11		
11	6	7	8	9	10

Figure 26 : Le graphe du problème *Myciel3* avec son tableau de contraintes

4.2.3.2. Représentation des solutions

La solution est présentée par un vecteur de taille fixe égale au nombre de variables du problème. Une configuration partielle vide correspond à un vecteur dont toutes les cases sont à 0. Les valeurs des couleurs étant numérotées à partir de 1, la valeur 0 est utilisée pour signifier que la variable correspondante n'est pas encore instanciée.

Nous illustrons la structure de données utilisée avec le problème *Myciel3* composé de 11 variables. Le contenu de la configuration partielle initiale vide est donné par la Figure 27, dans cette configuration, aucune variable n'est instanciée.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0	0	0	0	0	0	0	0	0	0	0

Figure 27 : Structure de la solution pour le problème *Myciel3*

L'avantage de cette représentation est la possibilité de traiter facilement des configurations partielles et complètes. La complexité spatiale de stockage est de l'ordre de *O(n)*, *n* étant le nombre de variables du problème.

4.2.3.3. Représentation des domaines

Dans un problème de coloration de graphe, toutes les variables possèdent le même domaine correspondant aux couleurs disponibles, on peut donc simplifier leur représentation par rapport au FAP où les domaines sont non seulement distincts par variables mais aussi composés d'union d'ensembles disjoints ce qui demande une représentation complexe. En coloration les couleurs disponibles sont représentées par des entiers ; si le nombre de couleurs disponibles pour colorier un graphe donné est *k* alors les couleurs seront représentées par les chiffres de 1 à *k*.

Nous avons représenté le domaine des variables par une matrice à 2 dimensions qui informera l'algorithme sur les couleurs disponibles et indisponibles par nœud. Prenons le problème *Myciel3* comme exemple ; si on dispose de 4 couleurs pour colorier ce graphe, alors les domaines sont codés par une matrice 11×4 dont un élément *(i,j)* représente un couple (variable, couleur). La structure du domaine pour le problème *Myciel3* est donnée en Figure 28.

	1	**2**	**3**	**4**
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0

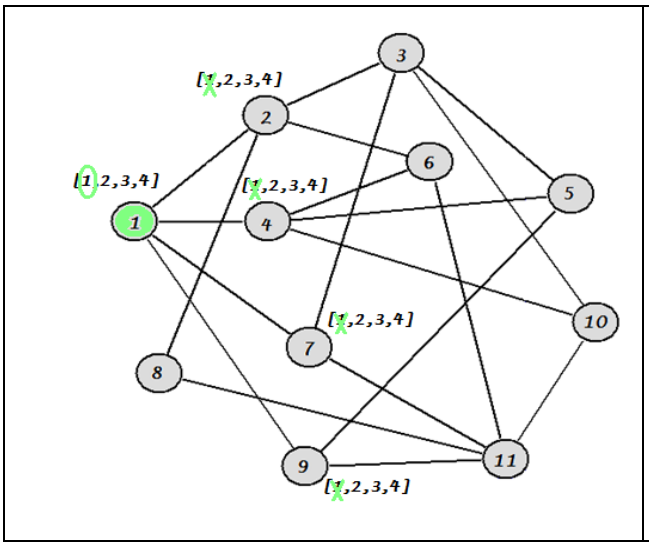
Figure 28 : Représentation des domaines pour le problème *Myciel3*

Un élément de la matrice indique la disponibilité de la couleur *j* pour le nœud *i*. La matrice est initialisée à 0 ce qui signifie que toute les valeurs sont libres pour toutes les

variables. Si l'élément *(i,j)* est supérieur à 0 alors la couleur *j* n'est pas disponible pour le nœud *i* i.e. elle est affectée à un nœud voisin de *i*. La valeur stockée dans *(i,j)* contient le nombre de voisins de *i* ayant *j* comme couleur, la matrice est donc mise à jour pour chaque nouvelle affectation ou désaffectation.

Cette structure de donnée remplit donc deux fonctions essentielles pour l'algorithme : marquer l'ajout d'une nouvelle instanciation à la configuration partielle courante lors de la propagation et marquer le retrait d'une instanciation lors de la propagation d'une réparation d'une configuration partielle. Nous traitons un exemple ci-après.

Supposons que l'algorithme affecte la valeur 1 à la variable 1, alors les nœuds voisins 2, 4, 7 et 9 (voir Figure 29) ne pourront plus prendre la valeur 1. Cette élimination est représentée par une incrémentation de 1 dans les cases correspondantes aux couples (*nœuds voisins*, 1).



	1	2	3	4
1		0	0	0
2	1	0	0	0
3	0	0	0	0
4	1	0	0	0
5	0	0	0	0
6	0	0	0	0
7	1	0	0	0
8	0	0	0	0
9	1	0	0	0
10	0	0	0	0
11	0	0	0	0

Figure 29 : Propagation de contraintes pour le problème *Myciel3*

Lors d'une réparation, la propagation inverse des contraintes est la procédure par laquelle l'algorithme restitue l'état des valeurs utilisables par chaque variable suite à une désaffectation. La détection d'un *deadend* (une variable a son domaine vide) correspond à une ligne de la structure contenant uniquement des valeurs supérieures à 0. L'algorithme doit alors libérer une des variables impliquée dans le *deadend* en la désaffectant. Ces variables sont connues à partir du *nogood* responsable du blocage de la variable dont le domaine est devenu vide. Suite à la libération d'une variable *x* de sa valeur *v*, il suffira alors de décrémenter de 1 dans la structure des domaines, les cases (*y*, *v*) où *y* est un voisin de la variable désaffectée *x*. Dans l'exemple de la figure 7, si la variable 1 dont la valeur est 1 est désaffectée, les cases (*nœuds voisins*, 1) sont décrémentées.

Pour les problèmes de coloriage de graphe, cette structure est simple et efficace pour les deux sens de propagation. Cette simplicité vient de la nature des contraintes du problème et de la nature de la propagation de contraintes via *Forward Checking*. Une structure telle que celle-ci n'était pas adaptée au FAP du fait de la complexité des contraintes.

4.2.3.4. Représentation des deadends

Pour accélérer le mécanisme de détection de *deadend*, un tableau de *n* cases (avec *n* nombre de variables) est associé au tableau des domaines. Ce tableau associe une case à

chaque variable contenant le nombre de valeurs libres pour la variable correspondante. Toutes les cases sont initialisées au nombre de couleurs disponibles *k* au début du coloriage.

Ce tableau est mis à jour à chaque itération de la recherche. La détection d'un *deadend* est immédiate avec cette structure car il suffit alors qu'une variable non instanciée n'ait plus aucune valeur libre, soit que l'élément du tableau indique 0.

4.2.3.5. Représentation des nogoods

Tabu-NG est une méthode qui utilise la notion de *nogood*. Chaque fois qu'un *deadend* est détecté pour une variable bloquée, un *nogood* est identifié puis mémorisé comme étant à l'origine de ce blocage. L'identification d'un *nogood* en FAP et en coloration ne se fait pas de la même façon. Nous présentons dans cette partie une structure de donnée simple adaptée au problème qui consiste à stocker pour chaque couleur la liste des nœuds voisins d'une variable bloquée qui utilisent cette couleur.

Prenons l'exemple de la Figure 30 avec *k*=3 couleurs, l'affectation de la couleur bleue (valeur 3) à la variable 10 a causé un blocage (*deadend*), la variable 11 n'a plus de valeur libre.

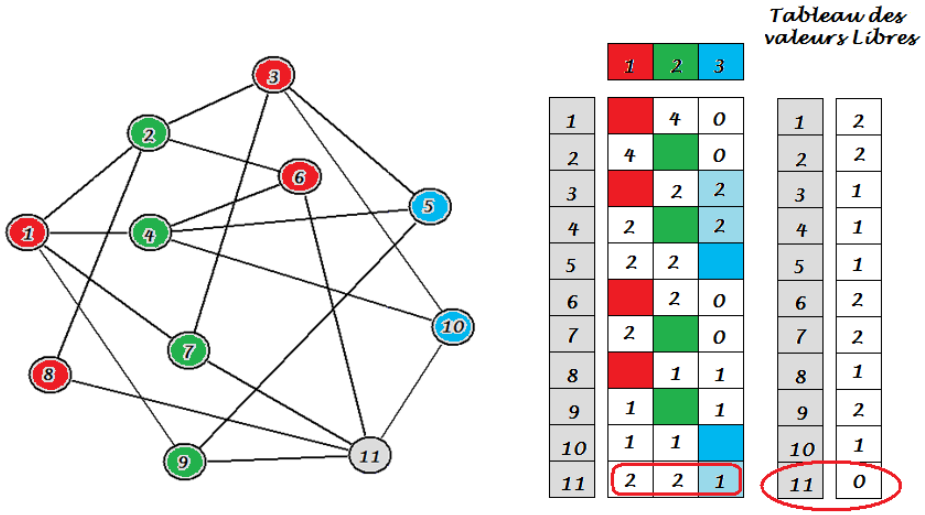


Figure 30 : Cas d'un deadend pour le problème *Myciel3*

L'identification du *nogood* sur ce blocage passe par la création d'un vecteur de dimension *k* qui stocke pour chaque couleur les indices des nœuds voisins de la variable bloquée ayant éliminé cette couleur. Un *nogood* correspond dès lors à l'association d'une explication d'élimination pour chaque valeur possible de couleur. En Figure 9, la structure de gauche indique les raisons de l'élimination de chaque valeur possible de la variable bloquée. Pour cette exemple, toute association du type {V6=1 ou V8=1} et {V7=2 ou V9=2} et {V10=3} est un *nogood*. Par exemple l'association de l'affectation de la couleur rouge à la variable 6, de la couleur verte à la variable 9 et de la couleur bleue à la variable 10 constitue un *nogood*.

Plus généralement, en cas de blocage sur une variable *x* donnée, toute association du type $\{y_{x1}=1\}$, $\{y_{x2}=2\}$, …, $\{y_{xk}=k\}$, avec y_{xi} un voisin de la variable *x* et *k* le nombre de couleurs disponibles, est un *nogood*. Cette méthode présente deux avantages, on peut trouver facilement un *nogood* sans avoir à enregistrer la cause d'élimination de chaque valeur, ainsi on réduit la complexité spatiale de la recherche des *nogoods*, et on peut trouver plusieurs

nogoods suite à un seul blocage. Cette structure et cette méthode ne sont pas applicables pour le FAP du fait de la complexité des contraintes.

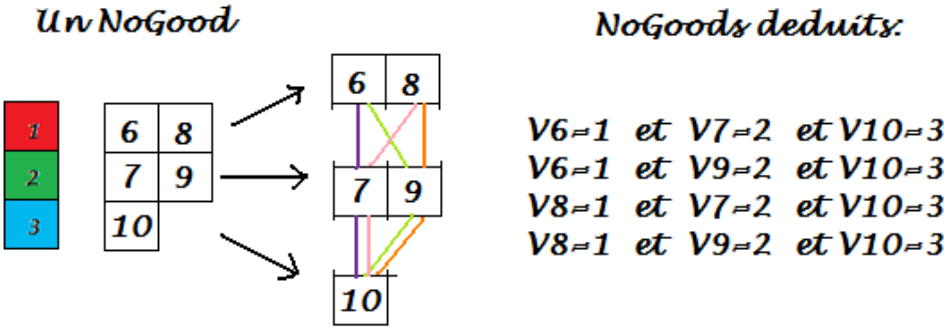


Figure 31 : *Nogoods* déduits sur *Myciel3*

4.2.3.6. *Efficacité des structures proposées*

Dans cette section, nous donnons dans le Tableau 28, quelques résultats qui illustrent l'efficacité des nouvelles structures proposées. La colonne *Tabu-NG:FAP* présente le nombre d'itérations fait en 5 secondes par la méthode implémentée pour le FAP. La colonne *+Filtrage* rappelle le nombre d'itérations fait en 5 secondes en changeant l'algorithme de propagation (remplacement par un algorithme de filtrage de type *Forward Checking* cf. section 4.2.2). La colonne *+Structures* présente les résultats obtenus en utilisant les structures de données adaptées à la coloration (cf. section 4.2.3). Le gain de ces adaptations permet sur cet intervalle de temps restreint de visiter 20 à 50 fois plus de configurations.

Nom	***k***	**T(s)**	**Iter *Tabu-NG* :FAP**	**+Filtrage**	**+Structures**
le450_15c	15	5	1073	3821	72543
le450_15d	15	5	1043	4293	67283
le450_25c	25	5	1009	4372	57770
le450_25d	25	5	1015	4533	57686
le450_5a	4	5	2222	13913	46585
le450_5b	4	5	2067	13651	46067
le450_5c	4	5	1319	10059	46902
le450_5d	4	5	1254	12347	43277

Tableau 28 : Adaptation de *Tabu-NG* et comparaison de performances

A ce stade, les paramètres originaux de la méthode *Tabu-NG* ont été conservés : heuristiques d'extension des configurations partielles, choix de la variable à désaffecter et durée Tabou. Notre attention s'est portée sur la lenteur de l'algorithme du fait de structures et propagations issues du FAP et non adaptées à la coloration. Ce travail met très clairement en évidence l'importance de l'implémentation informatique qui doit être adaptée à chaque problème quelle que soit la méthode de résolution utilisée.

4.2.4. Stockage des *nogoods*

Les structures utilisées pour propager et retro-propager les décisions ayant été adaptées, nous avons remarqué que le contrôle de l'appartenance de la configuration partielle courante à la liste des *nogoods* stockés est désormais la procédure qui impacte le plus la rapidité de la méthode. Le Tableau 29 donne quelques grandeurs utiles pour montrer le temps de calcul imparti à la vérification des *nogoods*. Sans vérification de l'appartenance de la configuration partielle courante aux *nogoods*, le nombre d'itérations effectué en 5 secondes est 3 à 5 fois plus important. A noter qu'aucun des problèmes n'est résolu en 5 secondes.

Nom	***k***	**T(s)**	**Iter *Tabu-NG* avec *nogood***	**Iter *Tabu-NG* sans *nogood***
le450_15c	15	5	72543	196547
le450_15d	15	5	67283	181848
le450_25c	25	5	57770	214568
le450_25d	25	5	57686	216493
le450_5a	4	5	46585	208671
le450_5b	4	5	46067	224291
le450_5c	4	5	46902	211455
le450_5d	4	5	43277	211857

Tableau 29 : Nombre d'itérations de *Tabu-NG* en 5 secondes avec et sans *nogood*

Nous avons complété ce test par une vérification du gain en qualité des solutions produites avec l'utilisation des *nogoods* afin de justifier la lenteur apportée. La méthode *Tabu-NG* étant déterministe la mesure de performances peut se faire sans erreur. Le Tableau 30 affiche le nombre d'itérations (ou de configurations) nécessaire pour obtenir la solution de 3 problèmes DIMACS avec les deux approches avec et sans stockage de *nogoods*. Les résultats montrent que l'impact de l'élimination du test d'appartenance des *nogoods* est réel mais faible sur le nombre de configurations visitées pour trouver une solution optimale au problème. Si le critère de comparaison d'algorithme est le nombre de configurations visitées alors le stockage des *nogoods* est justifié ; si le critère est le temps nécessaire pour trouver une solution alors on peut penser que, toute proportion gardée, l'écart entre le nombre d'itérations gagné avec le stockage pour trouver l'optimum et la lenteur supplémentaire engendré, ne justifie pas forcément leur utilisation. Le fait est que pour les problèmes difficiles, demandant éventuellement de visiter beaucoup de configurations, et pour les problèmes irréalisables, il vaut mieux privilégier la rapidité des calculs pour visiter plus de configurations dans le temps imparti.

Nom	***k***	**Solution optimale trouvée**	**Iter *Tabu-NG* avec *nogood***	**Iter *Tabu-NG* sans *nogood***
le450_5c	6	Oui	2103	2114
le450_5d	6	Oui	3732	3752
inithx.i.2	31	Oui	676	676

Tableau 30 : Tests de performances avec et sans *nogood*

Nous avons analysé plus en détail l'apport des *nogoods* tels qu'utilisés pour le FAP dans le cadre de la coloration car ils contribuent beaucoup moins vite à trouver une solution en coloration. Le but principal de la propagation de contraintes est la prévention du blocage le plus tôt possible et par la suite d'éviter d'aller loin dans une branche non prometteuse de l'arbre de recherche. Le but principal des *nogoods* est de bénéficier des informations déjà trouvées pendant la recherche. Si l'ajout d'une certaine instanciation à la configuration partielle courante est reconnu comme cause de blocage (*deadend*) immédiat ou pas, il est important d'éviter à nouveau le choix de cette instanciation. La gestion des *nogoods* gagne en efficacité au fur et à mesure de la recherche permettant à la méthode de détecter les situations de blocage de plus en plus tôt et de fournir une protection contre le recours excessif aux procédures de propagation de contraintes.

Dans le cadre de la coloration, les phénomènes de symétrie ne sont pas gérés par le stockage actuel des *nogoods*, ces derniers sont donc moins efficaces pour détecter les coupes et réduisent moins efficacement l'espace de recherche que pour le FAP. Par ailleurs la nature des contraintes fait que l'on propage moins *loin* les affectations courantes rendant les coupes très basses dans l'arbre ou pourrait-on dire très locales par rapport à la dernière décision prise. Les *nogoods* actuels gèrent donc quelques cas de *trashing* pour la coloration mais très peu comparativement au FAP. Nous avons par conséquent tranché en faveur de l'élimination de la procédure de test d'appartenance aux *nogoods* pour gagner du temps en calcul. Un travail est à faire sur l'adaptation des *nogoods* stockés pour le cas de la coloration afin de placer des coupes plus générales qu'actuellement et donc plus efficace pour trouver les bonnes branches de l'arbre.

Dans l'immédiat nous avons décidé de supprimer la procédure de stockage des *nogoods*. Ainsi lors d'un *deadend*, le *nogood* responsable est calculé directement puis guide la procédure de désaffectation de variables pour rendre la configuration consistante ; aucune information n'est conservée par la suite. On garde donc une procédure d'explication locale mais l'apprentissage des poids sur l'ensemble des *nogoods* n'est pas conservé. Ainsi toutes les affectations indiquées dans le *nogood* seront directement annulées. Par exemple, en référence à la Figure 31, 4 *nogoods* seront identifiés et tous désaffectés, soient 5 variables libérées. A ce stade, toutes les variables qui étaient bloquées suite à la dernière instanciation se débloquent directement du fait de l'annulation de la dernière instanciation qui se trouve forcément dans le *nogood* et qui va être annulée.

4.3. Méthode *Tabu-NG* pour la coloration de graphe

4.3.1. Schéma général

Nous donnons dans cette section, le pseudo code de notre méthode après les adaptations. Le schéma général de la méthode est donné dans la Figure 32, l'architecture reste la même. La méthode commence par une configuration partielle vide qu'elle modifie itérativement. A chaque itération, elle étend cette configuration en choisissant une nouvelle instanciation (variable, valeur). Après la phase d'extension, la méthode lance la procédure de propagation de contraintes. Après cette étape, deux cas peuvent se présentés :

1. Blocage : la méthode répare la configuration partielle en annulant une ou plusieurs instanciations déjà faites et revient à l'étape d'extension.
2. Pas de blocage : si la configuration est complète alors une solution valide est trouvée ; sinon on poursuit l'extension de la configuration partielle.

Le changement touche la forme plutôt que le fond de la méthode. Elle reste constructive et déterministe, elle se base toujours sur la liste de *nogoods* pour réparer la solution courante même si la notion de stockage des *nogoods* est supprimée. Suite à une réparation, la rétro-propagation annule directement le *deadend* (cf. section 4.2.4), par rapport à l'organigramme du chapitre 1, Figure 14, il n'y a donc plus la boucle sur le test de la persistance d'un *deadend*.

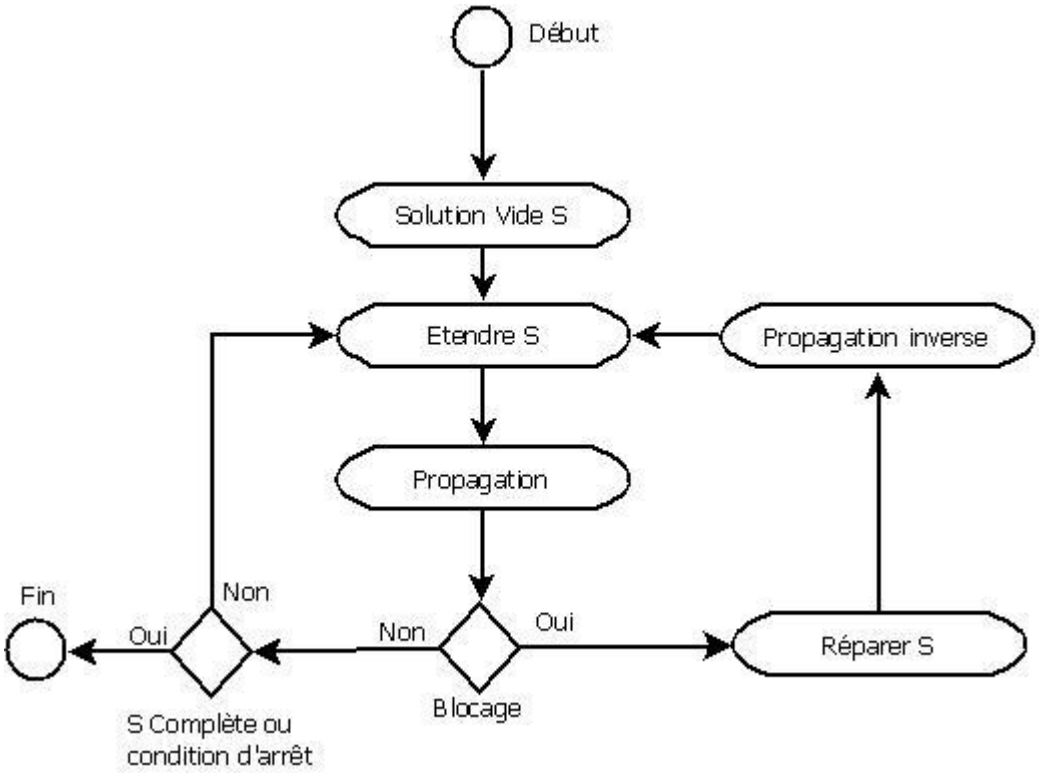


Figure 32 : Schéma général de *Tabu-NG* pour la coloration de graphe

Le pseudo code de la méthode *Tabu-NG* pour les problèmes de coloration de graphe est donné dans Algorithme 27.

```
Procedure Tabu-NG ()
1.      S = vide
2.      do
3.              (x, v) = EtendreConfiguration(S);
4.              isDeadEnd = PropagerAffectation(x, v) ;
5.              while isDeadEnd == true do
6.                      S = ReparerConfiguration(S)
7.                      isDeadEnd = PropagerInverse()
8.              endwhile
9.      while(S est partielle)
```

Algorithme 27 : Pseudo-code général de *Tabu-NG* pour la coloration de graphe

Dans les sections suivantes, nous donnons les paramètres de la méthode. Dans un premier temps, nous avons conservées les mêmes paramètres testés sur les problèmes d'affectation de fréquences et les avons testés sur les problèmes de coloriage de graphe. Suite aux résultats obtenus, nous avons commencé à réfléchir à adapter ces paramètres en fonction du problème traité.

4.3.2. Heuristique d'extension d'une configuration partielle consistante

Dans cette section, nous rappelons l'heuristique responsable de l'extension modifiée légèrement suite à l'élimination du stockage des *nogoods*. L'extension d'une configuration partielle *S* repose sur le choix d'un couple (*variable, valeur*) à ajouter. La variable choisie doit obéir à deux règles :

1. MRV pour *Minimum Remaining Values* : nous sélectionnons les variables ayant le plus petit domaine restant parmi les variables libres.
2. Si aucune d'elle n'a de valeur libre non taboue, on sélectionne une nouvelle liste pour le MRV suivant.
3. Sinon (cas où plusieurs variables sont à égalité sur MRV et possèdent des valeurs libres), une variable est choisie aléatoirement pour être affectée.
4. Si toutes les variables libres n'ont que des valeurs taboues, alors on sélectionne une variable aléatoirement.

Le choix de la valeur est simple, il s'agit de la première valeur du domaine, parcouru de la plus petite à la plus grande valeur, qui n'est pas taboue.

Le pseudo code de la fonction responsable de l'extension d'une solution est donné dans Algorithme 28.

```
Function EtendreConfiguration(S)
1.    Vars = getVarsMRVNotTabu() ;
2.    if Vars is Not Empty then
3.          varChoisie = Vars.Random()
4.          valChoisie = getFreeValueNotTabu(varChoisie)
5.    else
6.          varChoisie = AllFreeVariables.Random()
7.          valChoisie = getFreeValue(varChoisie)
8.    Endif
9.    S = S ⋃ (varchoisie, valChoisie)
10.   Return (varChoisie,valChoisie)
```

Algorithme 28 : Etendre une configuration partielle consistante

La fonction *EtendreConfiguration* étend la configuration partielle *S* courante en ajoutant le couple (variable, valeur) choisie à *S*. La propagation de l'effet de la dernière instanciation ajoutée, qui est décrite dans la section 4.2.2, est lancée après l'extension afin d'éliminer les valeurs inconsistantes dans les domaines des variables libres.

4.3.3. Heuristique de réparation d'une configuration partielle inconsistante

Après l'extension d'une configuration partielle, la propagation de l'effet de l'ajout d'une nouvelle instanciation peut conduire à un *deadend*. Dans le cas d'un *backtracking* normal, on annulerait la dernière affectation faite. Dans notre algorithme, la première variable bloquée, notée *x*, parmi la liste des variables bloquées (variables dont le domaine est devenu vide) est utilisée pour rendre la configuration consistante. On recherche la cause de l'élimination de toutes ses valeurs, soit un n*ogood.* Ce *nogood* (voir section 4.2.3.5 pour son identification) est composé de plusieurs ensembles d'affectations qui bloquent l'instanciation de la variable *x* ;

par exemple 4 ensembles donnant lieu à 5 affectations annulées dans l'exemple de la Figure 31. Afin de réparer la configuration partielle courante, nous annulons toutes les affectations qui appartiennent au *nogood* repéré.

En Figure 33, nous présentons l'état d'exécution de la méthode *Tabu-NG* sur un problème de coloration de graphe fictif composé de 11 variables avec k=3. La méthode a déjà affecté les 5 premières variables. Après l'affectation de la 5ème variable, un blocage a eu lieu. Les variables 6, 7 et 8 sont bloquées. Afin de réparer la configuration partielle courante, on prend la première variable bloquée (6) et on calcule le *nogood* source du blocage, puis on annule toutes les affectations qui appartiennent à ce *nogood*. Supposons que le *nogood* source du blocage de la variable 6 soit le suivant : {(1,1) ou (2,1)} et {(3,2)} et {(5,3)}. Dans ce cas les variables à désaffecter sont 1, 2, 3 et 5.

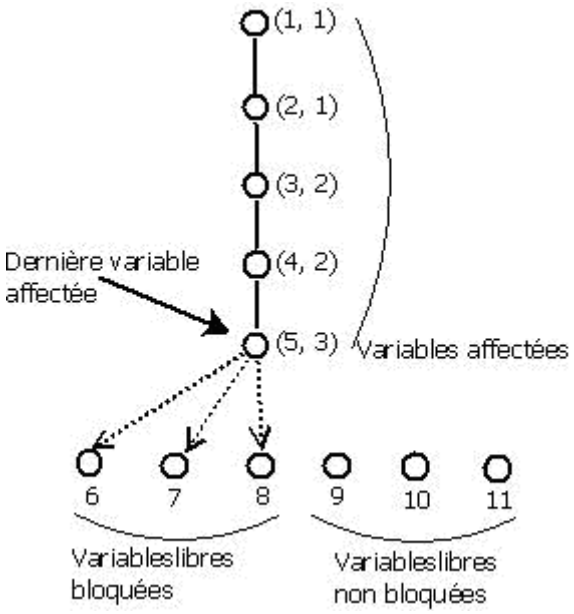


Figure 33 : Blocage des variables

4.3.4. Durée Tabou

Suite à une réparation d'une configuration partielle, les affectations (*variable*, *valeur*) annulées sont considérées comme taboues pour un nombre d'itérations donné afin de favoriser une diversification et d'éviter des cycles courts dans la recherche. Dans un premier temps, nous avons utilisé la durée du FAP (voir Equation 5) qui est calculée selon le nombre de fois où la même décision a été prise par la recherche.

$$\Delta(x, v) = iterCourante + nbreMvt(x, v) \qquad \textbf{Equation 5}$$

Suite à quelques tests montrant de mauvais scores par rapport aux résultats de la littérature, nous avons utilisé la durée Tabou de la méthode *TabuCol* (Equation 6) qui a prouvé son efficacité dans [FF96] [GH99]. La valeur de la durée Tabou s'adapte à l'avancement de la construction de la solution ; plus la configuration partielle est grande plus la durée Tabou est courte, il y a donc plus de diversité dans le choix des couples (*variable*, *valeur*) au début de la recherche, ce qui est plus ou moins le contraire de la durée Tabou issue du FAP... *NbreVarLibre* représente le nombre de variables libres de la configuration partielle courante et la fonction *UNIFORM(0,9)* retourne aléatoirement un chiffre entre 0 et 9 inclus.

$$\Delta(x, v) = iterCourante + 0.6*NbreVarLibre + UNIFORM(0,9)$$ **Equation 6**

Cette durée a donné des résultats corrects que nous présenterons en fin de partie, nous l'avons donc conservé sans chercher d'autres mécanismes. A noter que la méthode perd son aspect déterministe du fait de l'introduction de cette durée.

4.3.5. Tests et analyse

Nous présentons dans cette section les résultats obtenus par *Tabu-NG* dans sa version adaptée à la coloration. Dans cette section, la méthode utilise toutes les adaptations évoquées précédemment : nouvelle propagation, nouvelles structures de données, nouvelle durée Tabou et nouvelles procédures d'extension et de réparation. La méthode est rendue stochastique du fait de la durée Tabou qui inclut un paramètre aléatoire et du fait du choix aléatoire de la variable à affecter lors de l'extension si toutes les variables libres ont des valeurs taboues.

Le Tableau 31 montre les résultats obtenus par la méthode sur des problèmes réputés faciles. Les tests ont consisté en 10 exécutions sur chaque instance. χ est le nombre chromatique, le taux de succès *Suc* est une note sur 10 représentant le nombre d'exécutions ayant abouti à une solution complète, *k* est le nombre de couleurs utilisé, *Iter* est le nombre moyen d'itérations nécessaire à l'obtention de cette solution et *Sec* est le temps moyen d'exécution de la méthode en seconde. La solution optimale est trouvée systématiquement et sans délai ce qui est un point de départ rassurant.

		Tabu-NG			
Nom	χ	**Suc**	***k***	**Iter**	**Sec**
fpsol2.i.1	6	10	65	96	0
fpsol2.i.2	30	10	30	451	0
fpsol2.i.3	30	10	30	425	0
inithx.i.1	54	10	54	864	0
inithx.i.2	31	10	31	645	0
inithx.i.3	31	10	31	621	0
mulsol.i.1	49	10	49	197	0
mulsol.i.2	31	10	31	188	0
mulsol.i.3	31	10	31	184	0
mulsol.i.4	31	10	31	185	0
mulsol.i.5	31	10	31	186	0
school1	14	10	14	385	0
school1_nsh	-	10	14	423	0
zeroin.i.1	49	10	49	211	0
zeroin.i.2	30	10	30	211	0
zeroin.i.3	30	10	30	206	0
anna	11	10	11	138	0
david	11	10	11	87	0
homer	13	10	13	561	0
huck	11	10	11	74	0
jean	10	10	10	80	0
games120	9	10	9	120	0

		Tabu-NG			
Nom	χ	**Suc**	***k***	**Iter**	**Sec**
miles1000	42	10	42	128	0
miles1500	73	10	73	128	0
miles250	8	10	8	128	0
miles500	20	10	20	128	0
miles750	31	10	31	128	0
myciel3	4	10	4	11	0
myciel4	5	10	5	23	0
myciel5	6	10	6	47	0
myc el6	7	1	7	95	0
myciel7	8	10	8	191	0
1-FullIns_3	4	10	4	30	0
1-FullIns_4	5	10	5	93	0
1-FullIns_5	6	10	6	282	0
2-FullIns_3	5	10	5	52	0
2-FullIns_4	6	10	6	212	0
2-FullIns_5	7	10	7	852	0
3-FullIns_3	-	10	6	80	0
3-FullIns_4	7	10	7	405	0
4-FullIns_3	7	10	7	114	0
4-FullIns_4	8	10	8	690	0
5-FullIns_3	8	10	8	154	0

Tableau 31 : Résultats obtenus sur des instances faciles

Le Tableau 32 montre les résultats obtenus sur des problèmes reconnus difficiles par la communauté. La colonne *Best* donne le meilleur résultat connu dans la littérature pour colorier le problème correspondant. Pour ces tests le temps d'exécution est fixé à 30 minutes. Les cellules *blanches* représentent des résultats au même niveau que les meilleurs connus ; les cellules *gris clair* signifient que *Tabu-NG* est très proche du meilleur connu avec au plus 2 couleurs ; les cellules *gris foncé* signifient que *Tabu-NG* est loin du meilleur connu. Lorsque 2 scores sont présentés pour *Tabu-NG* il s'agit de voir le taux de succès si on descend *k* d'une valeur lorsque le meilleur connu n'est pas atteint.

			Tabu-NG		
Nom	**χ**	**Best**	**Suc**	***k***	**Itér**
DSJC125.1	5	5	10	5	285150
DSJC125.5	-	17	10	17	301612
DSJC125.9	-	44	10	44	68079
DSJC250.1	-	8	10		694266
DSJC250.5	-	28	10	30	694266
			0	29	
DSJC250.9	-	72	10	72	4400283
DSJC500.1	-	12	10	13	79810
			0	**12**	
DSJC500.5	-	48	10	52	513580
			0	51	
DSJC500.9	-	126	10	133	376306
			3	132	2485695
DSJR500.1	-	12	10	12	1103
DSJR500.1c	-	85	10	87	8146
			0	**86**	
DSJR500.5	-	125	10	126	500
			0	**125**	
DSJC1000.1	-	20	10	21	1859202
			0	20	
DSJC1000.5	-	84	10	93	1361376
			0	92	
DSJC1000.9	-	223	10	257	350049
			2	256	4586235

			Tabu-NG		
Nom	**χ**	**Best**	**Suc**	***k***	**Itér**
flat300_20_0	20	20	10	39	359405
			6	38	1487562
flat300_26_0	26	26	26	39	1998795
			0	38	
flat300_28_0	28	31	32	40	167018
			4	39	1548957
flat1000_50_0	50	50	10	112	589451
			3	111	1784563
flat1000_60_0	60	60	10	114	476932
			2	113	1326549
flat1000_76_0	76	83	10	115	758864
			3	114	2565871
le450_5a	5	5	10	5	397189
le450_5b	5	5	10	5	101563
le450_5c	5	5	10	5	71284
le450_5d	5	5	10	5	488
le450_15a	15	15	10	16	1618
			4	15	65418
le450_15b	15	15	10	16	450
			5	15	36426
le450_15c	15	15	10	22	33360
			1	21	678563
le450_15d	15	15	10	22	15625
			3	21	486572
le450_25a	25	25	10	25	450
le450_25b	25	25	10	25	450
le450_25c	25	25	10	27	24092
			0	26	
le450_25d	25	25	10	27	4464
			0	26	

Tableau 32 : Résultats obtenus sur des instances difficiles

Sur l'ensemble des résultats la méthode est relativement éloignée des meilleurs résultats connus. Elle résout 14 problèmes sur 33 de façon optimale, 3 problèmes sur 33 avec une couleur de plus. Pour les problèmes *flats*, la méthode proposée est très mauvaise.

La méthode est relativement agressive dans ces décisions pour l'extension et pour la réparation d'une configuration. L'extension utilise la règle qui privilégie la variable de plus petit domaine restant, cela se traduit par une intensification systématique autour d'une

variable ou de quelques-unes pour étendre la configuration courante. La réparation annule toujours toutes les affectations impliquées dans un *nogood* et les rend taboues, cela se traduit par une diversification systématique sur plusieurs variables à la fois. Ces deux règles qui fonctionnaient bien pour le FAP sont peut-être trop agressives pour la coloration ; notons que pour le FAP la réparation n'annulait pas systématiquement plusieurs variables, donc la règle était moins agressive. Pour tenter d'améliorer les résultats de la méthode nous avons introduit des décisions moins systématiques en tentant d'équilibrer l'intensification et la diversification de la recherche dans les deux phases, elles sont présentées dans la partie suivante. Nous donnons par ce biais un comportement radicalement stochastique à la méthode en introduisant des heuristiques de décision avec choix aléatoires en extension et réparation.

4.4. Réglages des actions d'intensification et de diversification

4.4.1. Diversification de l'extension d'une configuration partielle consistante

Afin de rendre la méthode moins déterministe dans le choix de la prochaine variable à affecter, nous remplaçons l'heuristique MRV lors du choix des variables par le choix aléatoire d'une variable parmi l'ensemble des variables non affectées et possédant au moins une valeur libre non taboue dans leurs domaines. Nous apportons ainsi plus de diversification sur l'extension de la configuration courante.

Le choix de la valeur est le même qu'avant, il s'agit de la première valeur du domaine, parcouru de la plus petite à la plus grande valeur, qui n'est pas taboue.

4.4.2. Intensification de la réparation d'une configuration partielle inconsistante

Dans le *backtracking* utilisé suite à un *deadend*, la version actuelle de la réparation d'une configuration choisit aléatoirement une variable parmi celles dont le domaine est vide (voir section 4.3.3), puis le *nogood* associé à la variable est identifié (voir section 4.2.3.5) et l'ensemble des affectations du *nogood* sont annulées et mises taboues. Cette méthode de réparation est très agressive car elle réalise systématiquement une désaffectation de plusieurs variables. Par ailleurs ces annulations deviennent toutes taboues pour les décisions à venir, de ce fait cette heuristique affaiblit l'action d'intensification de la méthode en faveur de la diversification systématique. En effet, l'action conjuguée d'annulation de plusieurs affectations et de déclarations taboues conduit la méthode à construire une solution relativement différente de la solution courante après chaque blocage.

Nous proposons de limiter l'impact de la réparation sur la configuration courante en réduisant le nombre de variables modifié. Dans la nouvelle procédure de réparation, on libère une seule couleur à partir du *nogood*. Le choix de la couleur à libérer se fera en minimisant le nombre d'instanciations à annuler. Il suffit pour cela de choisir la couleur la moins utilisée par les variables voisines (voir 4.2.3.3).

Prenons un exemple avec une configuration partielle dont la dernière extension a conduit à un *deadend* sur 3 variables. La nouvelle procédure de réparation choisit alors une de ces 3 variables bloquées au hasard, notée *x*, et la couleur *v* à libérer qui fait le moins d'annulation d'instanciations possible. Il suffit pour cela de choisir la couleur la moins utilisées par les variables voisines (voir 4.2.3.3). Puis les variables voisines de *x* qui utilisent *v* sont désaffectées et *v* est affectée à *x* puisqu'elle est libre (et c'est l'unique valeur libre) pour cette variable si cela ne génère pas un *deadend*, sinon *x* reste libre. Notons que cette action ne

signifie pas que l'algorithme n'est plus en état de *deadend*, les deux autres variables bloquées peuvent l'être encore du fait qu'une seule couleur du *nogood* a été désaffectée. On réévalue donc l'état de *deadend* et la procédure est répétée jusqu'à élimination de toutes les causes de blocage. Toutes les affectations annulées sont alors considérées taboues pour une certaine durée (le calcul de la durée est le même qu'avant).

Le pseudo-code de la nouvelle procédure de réparation est donné dans Algorithme 29. Suite à un blocage, on récupère toutes les variables bloquées *VB*, et pour chaque variable *x* appartenant à *VB* on choisit une *couleur* puis on annule toutes les affectations voisines de *x* qui utilisent *couleur*. La fonction *getVarsBloque* retourne l'ensemble des variables bloquées et la fonction *MinVoisinA_Annuler* retourne pour une variable bloquée *x* la couleur qui est affectée au plus petit nombre de voisin de *x*.

```
Function ReparerConfiguration(S)
1.      VB = S.getVarsBloque() ; //variables bloquées
2.      while VB.NotEmpty() do
3.              x = VB.Pop() ; //prendre une variable
4.              couleur = MinVoisinA-Annuler(x)
5.              foreach variable y in ν(x) do
6.                      if φ(y)==couleur then
7.                              AnnulerAffectation(y)
8.                              Mettre (y, couleur) taboue
9.                      endif
10.             endforeach
11.             S = S ⋃ (x,couleur)
12.             if PropagerAffectation(x, couleur) == true then
13.                     PropagerInverse(x, couleur)
14.                     S = S\(x, couleur)
15.             endif
16.     endwhile
End function
```

Algorithme 29 : Réparation d'une configuration partielle inconsistante

4.4.3. Tests et analyse

Le Tableau 33 dresse une comparaison entre les résultats obtenus précédemment (*Tabu-NG1*) et la version de *Tabu-NG* (*Tabu-NG2*) avec les nouvelles procédures d'extension et de réparation de configurations partielles. Les améliorations apportées sont importantes avec 21 problèmes sur 30 résolus au niveau des meilleurs connus et notamment sur les problèmes *flat* où 5 problèmes sur 6 sont résolus au niveau des meilleurs scores connus. Il y a 7 problèmes résolus avec une ou deux couleurs de plus que la meilleure solution connue et 3 problèmes restent durs à résoudre par la méthode (DSJC1000.5, DSJC1000.9 et flat1000_76_0) qui sont considérés parmi les problèmes les plus difficiles de DIMACS. A noter une unique régression par rapport à la version précédente de *Tabu-NG* pour le problème DSJC1000.1.

			Tabu-NG1			***Tabu-NG2***		
Nom	**χ**	**Best**	**Suc**	***k***	**Iter**	**Suc**	***k***	**Iter**
DSJC125.1	5	5	10	5	285150	10	5	99264
DSJC125.5	-	17	10	17	301612	10	17	4412275
DSJC125.9	-	44	10	44	68079	10	44	91929
DSJC250.1	-	8	10	8	694266	10	8	26708468
DSJC250.5	-	28	10	30	694266	7	28	255506820
DSJC250.9	-	72	10	72	4400283	10	72	165319263
DSJC500.1	-	12	10	13	79810	10	13	129010
DSJC500.5	-	48	10	52	513580	10	50	20245854
DSJC500.9	-	126	10	133	376306	10	129	16909615
			3	132	2485695	6	128	183985474
DSJC1000.1	-	20	10	21	1859202	10	22	230283
			0	20		7	21	5721775
DSJC1000.5	-	84	10	93	1361376	10	91	14946805
			0	92		2	90	26512365
DSJC1000.9	-	223	10	257	350049	10	232	56601705
			2	256	4586235	0	231	
flat300_20_0	20	20	6	38	1487562	10	20	58143
flat300_26_0	26	26	26	39	1998795	10	26	1013961
flat300_28_0	28	28	4	39	1548957	8	29	45758156
flat1000_50_0	50	50	3	111	1784563	10	50	4944318
flat1000_60_0	60	60	2	113	1326549	10	60	22553666
flat1000_76_0	76	83	3	114	2565871	7	89	223456411
le450_5a	5	5	10	5	397189	10	5	149952
le450_5b	5	5	10	5	101563	10	5	281825
le450_5c	5	5	10	5	71284	10	5	8608
le450_5d	5	5	10	5	488	10	5	3661
le450_15a	15	15	10	16	1618	10	16	97767
			4	15	65418	5	15	58389
le450_15b	15	15	10	16	450	10	16	268758
			5	15	36426	5	15	32690
le450_15c	15	15	1	21	678563	10	15	377148
le450_15d	15	15	3	21	486572	10	15	791150
le450_25a	25	25	10	25	450	10	25	16461
le450_25b	25	25	10	25	450	10	25	2812
le450_25c	25	25	10	27	24092	10	27	2694457
le450_25d	25	25	10	27	4464	10	27	641580

Tableau 33 : Résultats obtenus après le calibrage de *Tabu-NG*

A ce stade, nous avons décidé de conserver ces nouvelles procédures d'extension et de réparation des solutions partielles et de réaliser une analyse de comportement sur *Tabu-NG* pour proposer des améliorations éventuelles.

Nous avons analysé l'occurrence du choix des variables et des couleurs, autrement dit le nombre de fois où chaque variable ou chaque couleur est utilisée. La Figure 34 montre le nombre de mouvement par couleur et par variable pour 4 instances résolus à l'optimalité par la méthode. Les variables sont ordonnées par ordre d'apparition dans le fichier d'entrée de l'instance.

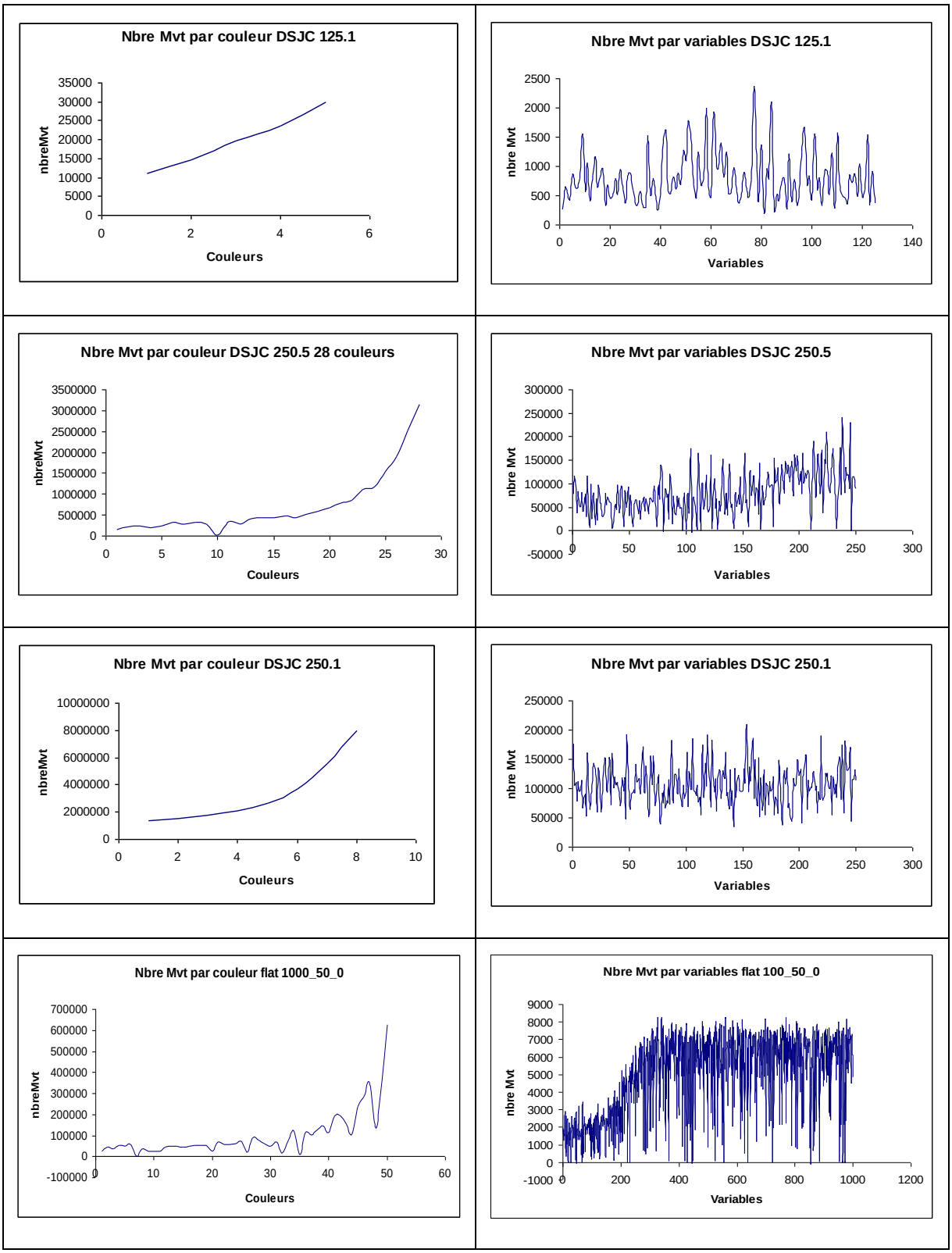


Figure 34 : Nombre de mouvements par couleur et par variable pour des problèmes résolus à l'optimalité

La Figure 35 montre le nombre de mouvements par couleur et par variable pour deux instances non résolues.

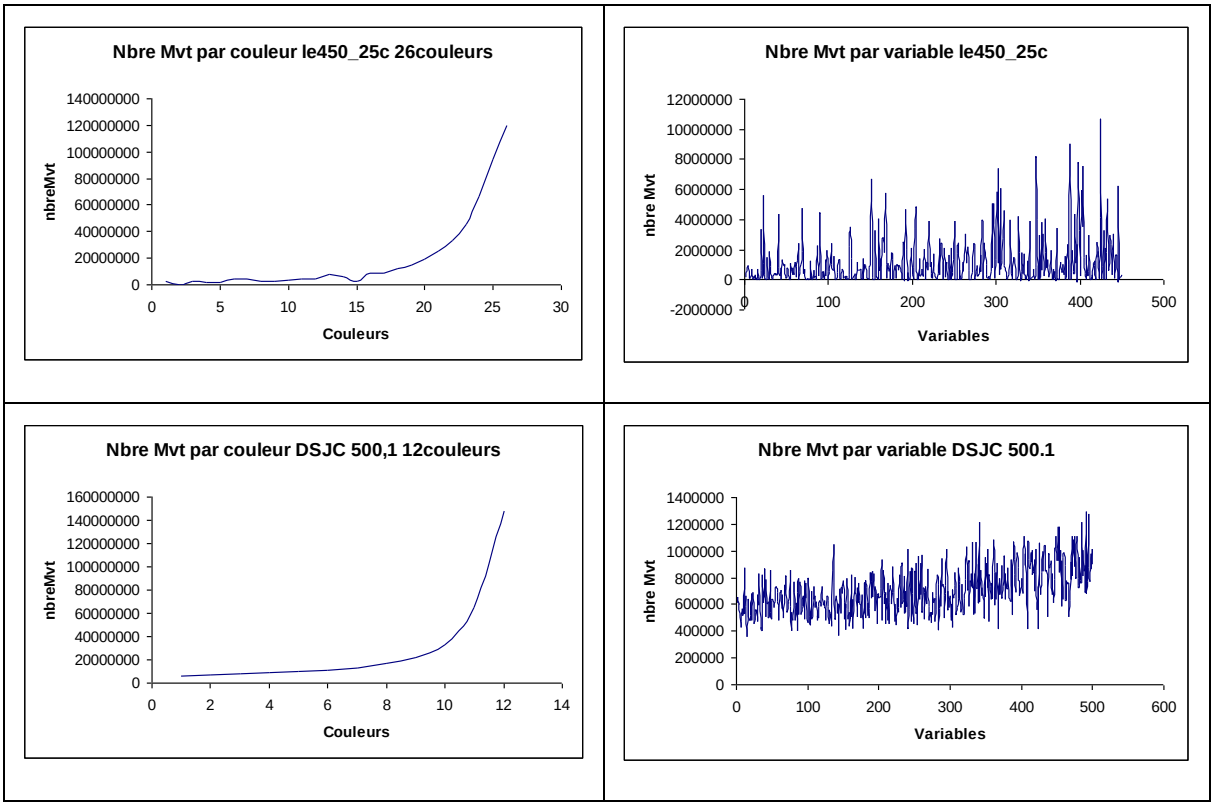


Figure 35 : Nombre de mouvements par couleur et par variable pour des problèmes non résolus

On observe que les courbes ne sont pas très homogènes surtout dans le choix des couleurs, il y a un manque de diversité sur le choix des couleurs. Par ailleurs, on a constaté en traçant les couples (*variable, valeur*) que lorsque l'algorithme ne progresse plus, la méthode boucle sur l'affectation des mêmes couleurs aux mêmes variables. Lorsqu'il ne reste plus que quelques variables non affectées, la méthode boucle sur les mêmes choix. Pour illustrer cette situation prenons l'exemple suivant :

1. Il reste 4 variables à affecter toutes en état de *deadend* i.e. domaines vides.
2. La méthode de réparation agit pour libérer une couleur du domaine de chaque variable bloquée.
3. Supposons que cette action de réparation nécessite de désaffecter 5 autres variables ramenant leurs domaines respectifs à de simples singletons. Ces valeurs sont taboues pendant un temps donné mais étant la seule valeur valide, le statut Tabou sera levé pour pouvoir faire une affectation.
4. Les extensions futures de la configuration partielle courante conduiront à choisir les mêmes couleurs pour les variables qui viennent d'être libérées puisque leurs domaines sont des singletons.

Par conséquent, la méthode effectuera des cycles instanciations/annulations des mêmes couleurs pour les mêmes variables. Le problème *latin_square_10* est montré en exemple en Figure 36, il est non résolu avec 105 couleurs après $500*10^6$ itérations. L'instance contient 900 variables, *Tabu-NG* affecte rapidement 895 variables et boucle le reste du temps sur les mêmes 5 variables restantes. A ce stade on voit que les extensions possibles de la solution

sont limitées par les heuristiques à un groupe de configurations possibles, l'algorithme ne peut plus du tout diversifier ; pour étendre cet ensemble de configurations il faut à un moment donné décider de relâcher les règles de décision.

Pour certains problèmes les stratégies d'intensification/diversification proposées sont efficaces (*flat* par exemple), pour d'autres ce n'est pas le cas, ce qui montre qu'une diversification doit toujours être possible pour l'algorithme.

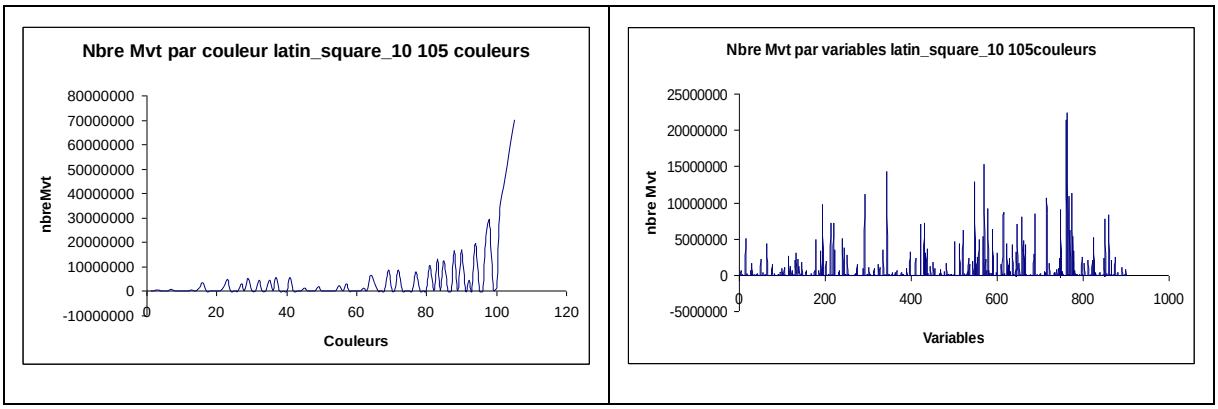


Figure 36 : Exemple détaillé pour *latin_square_10* non résolu avec 105 couleurs

4.4.4. Diversification du choix des couleurs

La phase d'extension d'une configuration partielle consiste à déterminer le couple (*variable*, *valeur*) permettant d'étendre la configuration partielle courante. La version actuelle de la procédure d'extension tente de choisir un couple valide dont la valeur affectée à la variable est consistante et non taboue ; si toutes les valeurs valides de la variable choisie sont taboues, alors un choix aléatoire est effectué parmi les couples (variable, valeur valide) tabous. Cette procédure fait boucler l'algorithme sur les couples (variable, valeur) lorsque valeur devient un singleton tabou ou pas tabou.

Pour permettre à la méthode de changer de configurations nous avons opté pour l'élargissement du choix d'extension vers les couleurs non consistantes de la variable. Nous introduisons la règle suivante : quand l'ensemble des valeurs valides d'une variable non instanciée sont taboues, une valeur non valide du domaine de la variable est choisie aléatoirement. L'extension de la solution conduit alors à un ensemble de conflits nouveaux corrigés par une succession de désaffectations pouvant conduire l'algorithme vers de nouvelles configurations. Le processus se poursuit ensuite normalement, les affectations ainsi annulées sont alors ajoutées à la liste des couples taboues, etc.

Le Tableau 34 montre les améliorations obtenues par l'utilisation de la nouvelle procédure d'extension. Parmi ces améliorations 3 scénarios, flat300_28_0, DSJC500.1 et DSJC1000.9 sont améliorés sur k et il n'y a aucune régression sur k par rapport à la version précédente. Concernant le taux de succès pour une même valeur de k, il y a 4 améliorations et 4 régressions. La méthode est donc globalement meilleure.

			Tabu-NG + calibrage			***Tabu-NG + diversification***			
Nom	**χ**	**Best**	**Suc**	**k**	**Iter**	**Suc**	**k**	**Iter**	**Sec**
DSJC125.1	5	5	10	5	99264	10	5	34903	0.1
DSJC125.5	-	17	10	17	4412275	10	17	1791777	5
DSJC125.9	-	44	10	44	91929	10	44	251930	7
DSJC250.1	-	8	10	8	26708468	10	8	50528290	738
DSJC250.5	-	28	7	28	255506820	3	28	340780939	7211
DSJC250.9	-	72	10	72	165319263	10	72	80390808	2512
DSJC500.1	-	12	10	13	129010	3	12	76415	2
DSJC500.5	-	48	10	50	20245854	10	50	124432552	6120
DSJC500.9	-	126	10	129	16909615	10	129	26542975	1620
			6	128	183985474	4	128	319294241	16390
DSJC1000.1	-	20	10	22	230283	10	22	83385	3
			7	21	5721775	4	21	116730928	27060
DSJC1000.5	-	84	10	91	14946805	10	91	4702367	450
			2	90	26512365	10	90	322343681	47400
DSJC1000.9	-	223	10	232	56601705	9	232	295366145	30725
			0	231		2	231	198812593	37200
flat300_20_0	20	20	10	20	58143	10	20	40541	2
flat300_26_0	26	26	10	26	1013961	10	26	431505	17
flat300_28_0	28	31	8	29	45758156	4	28	256139089	8760
flat1000_50_0	50	50	10	50	4944318	10	50	6978348	1661
flat1000_60_0	60	60	10	60	22553666	10	60	47564159	10762
flat1000_76_0	76	83	7	89	223456411	8	89	260191068	25842
le450_5a	5	5	10	5	149952	10	5	20401	1
le450_5b	5	5	10	5	281825	10	5	161815	7
le450_5c	5	5	10	5	8608	10	5	2372	0.2
le450_5d	5	5	10	5	3661	10	5	5632	1
le450_15a	15	15	10	16	97767	10	16	57808	1
			5	15	58389	7	15	68984	1
le450_15b	15	15	10	16	268758	10	16	51826	1
			5	15	32690	6	15	45264	1
le450_15c	15	15	10	15	377148	10	15	299660	3
le450_15d	15	15	10	15	791150	10	15	2136025	7
le450_25a	25	25	10	25	16461	10	25	43433	0.5
le450_25b	25	25	10	25	2812	10	25	1284	0.05
le450_25c	25	25	10	27	2694457	10	27	2049237	7
le450_25d	25	25	10	27	641580	10	27	2482660	8

Tableau 34 : Résultats obtenus après la diversification implémentée dans *Tabu-NG*

4.5. Comparaison des résultats obtenus avec la littérature

Dans cette section, nous présentons les derniers résultats en cours de *Tabu-NG* pour toutes les instances difficiles de coloration DIMACS que nous avons traitées dans cette thèse [166].

A noter que tous les tests sont faits avec le même paramétrage (durée Tabou notamment). Le Tableau 35 compare *Tabu-NG* avec les 8 méthodes présentées en section 4.1.5. Certaines de ces méthodes sont dédiées à la coloration (AMACOL, MACOL et RCTS) et certaines sont basées sur l'utilisation de classes de couleurs (AMACOL). MACOL 2010 présente actuellement les meilleurs scores connus dans la littérature sur toutes les instances.

				Recherche locale				**Algorithmes Hybrides**			**PPC**
Nom	χ	**Best**	***Tabu-NG***	**RCTS 2009**	**ALS 2008**	**ILS 2002**	**GLS 2005**	**DCNS 1996**	**AMACol 2008**	**MACOL 2010**	**FCNS 2002**
DSJC125.1	5	5	5	5	5	5	5	-	5	5	
DSJC125.5	-	17	17	17	17	17	18	-	17	17	18
DSJC125.9	-	44	44	44	44	44	44	-	44	44	
DSJC250.1	-	8	8	8	8	8	8	-	8	8	
DSJC250.5	-	28	28	28	28	28	29	-	28	28	32
DSJC250.9	-	72	72	72	72	72	72	-	72	72	
DSJC500.1	-	12	12	12	13	13	13	-	12	12	
DSJC500.5	-	48	50	48	49	50	52	49	48	48	54
DSJC500.9	-	126	128	126	127	127	129	-	126	126	
DSJC1000.1	-	20	21	21	21	21	21	-	20	20	
DSJC1000.5	-	84	90	87	89	90	93	89	84	83	97
DSJC1000.9	-	223	231	224	226	227	233	226	224	223	
flat300_20_0	20	20	20	20	20	20	20	-	20	20	20
flat300_26_0	26	26	26	26	26	26	33	-	26	26	35
flat300_28_0	28	28	28	30	31	31	33	31	31	29	35
flat1000_50_0	50	50	50	50	88	88	50	-	50	50	95
flat1000_60_0	60	60	60	60	87	89	90	-	60	60	97
flat1000_76_0	76	82	88	87	88	89	92	86	84	82	98
le450_15a	15	15	15	15	15	15	15	-	15	15	15
le450_15b	15	15	15	15	15	15	15	-	15	15	15
le450_15c	15	15	15	16	15	15	15	15	15	15	21
le450_15d	15	15	15	16	15	15	15	15	15	15	21
le450_25a	25	25	25	25	25			-	25	25	
le450_25b	25	25	25	25	25			-	25	25	
le450_25c	25	25	27	25	26	26	26	26	26	25	
le450_25d	25	25	27	25	26	26	26	26	26	25	
le450_5a	5	5	5	5	5	5	5	-	5	-	
le450_5b	5	5	5	5	5	5	5	-	5	-	
le450_5c	5	5	5	5	5			-	5	-	
le450_5d	5	5	5	5	5	5	5	-	5	-	
Latin_sqr_10	98	98	104	100	106			98	104	99	106
TNG Mieux			-	3	5	5	11	1	1	1	12
TNG Pareil			-	19	19	18	14	2	21	17	3
TNG Pire			-	8	6	4	2	7	8	8	0
Total			-	30	30	27	27	10	30	26	15

Tableau 35 : Comparaison des résultats de *Tabu-NG* avec la littérature pour les instances DIMACS

Tabu-NG est moins efficace que les trois méthodes hybrides pour 8 instances contre 1 et se classe troisième sur 5 pour les recherches locales. A noter qu'il n'y a pas d'instances avec

de trop grand écart par rapport au meilleur score connu (8 couleurs de plus au maximum). *Tabu-NG* qui appartient à la même classe de méthodes que FCNS (hybridation de PPC et recherche locale) dépasse cette dernière sur toutes les instances. Notons enfin que *Tabu-NG* améliore *flat300_28_0* par rapport à MACOL en étant la seule parmi les méthodes présentées à résoudre cette instance à l'optimalité.

En colonne le Tableau 35 présente les informations suivantes :

- Colonne 2, le nombre chromatique lorsque celui-ci est connu.
- Colonne 3, le meilleur résultat connu dans la littérature (plus petit *k*).
- Colonne 4, le meilleur score de *Tabu-NG* sur 10 exécutions avec 1000 minutes maximum.
- 4 colonnes pour des algorithmes de recherche locale.
- 3 colonnes pour des algorithmes hybrides (évolutionnaires et Recherche Tabou).
- La dernière colonne pour une approche PPC (la meilleure parmi celles référencées en *k*-coloration).

En ligne le Tableau 35 présente les informations suivantes par rapport à chaque méthode mentionnée :

- Ligne *Mieux*, le nombre d'instance où *Tabu-NG* a un plus petit *k*.
- Ligne *Pareil*, le nombre d'instance où *Tabu-NG* a le même *k*.
- Ligne *Pire*, le nombre d'instance où *Tabu-NG* a un plus grand *k*.
- Ligne *Total*, le nombre total d'instance traité par cette méthode.

4.6. Synthèse

L'application d'une méthode de recherche sur un problème très référencé et largement étudié dans la littérature constitue une bonne plateforme de mise au point. Le problème de *k*-coloration de graphe est un des problèmes les plus étudiés dans la littérature avec toutes sortes de méthodes. Dans ce chapitre, nous avons présenté les travaux sur les instances DIMACS qui sont utilisées par la plupart des articles publiés sur la *k*-coloration, elles ont la réputation d'être difficiles à résoudre. Ces jeux sont toujours d'actualité avec une publication des meilleurs scores connus en 2010. Le travail est présenté volontairement de manière progressive pour associer les solutions proposées aux analyses successives que nous avons faites.

La première partie de ce chapitre présente le problème de *k*-coloration de graphe et la similitude entre ce problème et le CSP. Elle présente aussi l'ensemble des instances utilisées qui regroupe des instances faciles et des instances difficiles. Nous donnons aussi dans cette partie une vue globale sur les méthodes de résolution et nous présentons les meilleures réparties en trois classes, les méthodes de recherche locale, les méthodes à base de programmation par contraintes et les méthodes hybrides.

La deuxième partie de ce chapitre se concentre sur les évolutions de certaines implémentations de la méthode *Tabu-NG* pour traiter la coloration. Les différences avec le FAP sont assez importantes sur la nature des contraintes et des domaines. Nous avons donc proposé de nouvelles structures de données propres aux problèmes de coloration (contrainte,

domaine, solution, *nogood*, *deadend*), et nous avons montré l'importance de ces structures en termes de rapidité d'exécution (nombre de configurations visitées pour un temps donné). Ces structures nous ont permis de maintenir plus facilement la consistance dans un graphe de contraintes suite à l'extension d'une configuration partielle consistante ou suite à la réparation d'une configuration partielle inconsistante.

Un deuxième travail a été fait sur l'utilité du stockage des *nogoods* dans une liste Tabou par rapport au temps de calcul engendré pour maintenir et parcourir cette liste. Les *nogoods* sont utilisés pour prendre les décisions sur les réparations lors de conflit. La liste établit en plus un apprentissage sur les coupes de l'espace de recherche. Ne traitant pas les problèmes de symétrie de la coloration, les coupes s'avèrent trop locales pour être efficaces. Nous avons alors estimé que le stockage des *nogoods* pour la coloration n'était pas forcément nécessaire et que les *nogoods* pouvaient être calculés en ligne. Nous avons donc éliminé la liste Tabou perdant ainsi la notion d'apprentissage au bénéfice d'un accroissement du nombre de configurations visité.

La troisième partie porte sur l'application de la méthode *Tabu-NG* à la coloration. Le schéma général de *Tabu-NG* est expliqué puis nous présentons les adaptations des procédures d'extension et de réparation et l'adaptation de la durée Tabou. L'extension est faite selon les mêmes principes que pour le FAP avec une intensification autour des variables de plus petit domaine restant ; la diversification utilise le *nogood* calculé pour désaffecter toutes les variables explicatives du *deadend* ; enfin la durée Tabou utilisée est une modification de celle de *TabuCol* basée sur la taille de la configuration partielle courante. Toutes les instances faciles de DIMACS sont résolues à l'optimalité mais les instances difficiles sont hors de portée. Notre analyse a montré que les heuristiques d'extension et de réparation sont trop agressives dans leur démarche, l'une trop intensive et l'autre trop diversifiée.

Dans la quatrième partie de ce chapitre, nous corrigeons les effets de ces heuristiques et proposons de nouveaux résultats. L'extension est plus diversifiée en permettant un choix de variables parmi toutes celles non affectées et non taboues au lieu de *Minimum Remaining Values*. Suite à un *deadend,* la réparation est plus intensive en désaffectant les variables explicatives de la même couleur au lieu de désaffecter toutes les variables explicatives quelle que soit leur couleur. Enfin le choix des couleurs pour l'extension a été diversifié. Le choix de couleur limité aux valeurs valides (taboues ou non) conduisait l'algorithme à reconduire systématiquement les mêmes décisions pour des domaines singletons et empêchait l'exploration de configurations partielles nouvelles. Nous avons donc introduit la possibilité de choisir une valeur inconsistante sous certaines conditions.

Nous avons présenté et comparé avec la littérature les derniers résultats de *Tabu-NG* sur les instances difficiles DIMACS en fin de 4ème partie. *Tabu-NG* est très compétitive par rapport à des méthodes dédiées à ce problème. Elle se classe derrière les 3 meilleures méthodes hybrides et 3ème sur les 5 recherches locales référencées. L'écart maximum sur toutes les instances est seulement de 8 couleurs supplémentaires. A noter que *Tabu-NG* améliore une instance difficile en étant la seule parmi les méthodes présentées à résoudre cette instance à l'optimalité (*flat300_28_0*).

Au vu des premiers résultats de *Tabu-NG* sur ce problème, nous sommes convaincus que la méthode présente encore du potentiel. Nous sommes conscient qu'un travail supplémentaire d'analyse comportementale est nécessaire afin d'améliorer les performances. Il est par exemple important de reconsidérer le stockage des *nogoods* en fixant peut être la taille du nombre de *nogoods* stockés comme compromis entre rapidité et apprentissage. Les paramètres de la méthode méritent aussi d'être étudiés d'une manière approfondie ; la méthode est sensible aux heuristiques d'extension et de réparation, mais elle est aussi sensible

à la durée Tabou. Pour l'instant *Tabu-NG* a été appliquée sur toutes les instances avec les mêmes paramètres, nous devons faire des tests avec des paramètres différents sur les problèmes les plus difficiles.

5. Conclusion et perspectives

Le travail présenté dans cette thèse s'inscrit dans un contexte opérationnel de résolution et d'optimisation de problèmes combinatoires en traitant des problèmes de satisfaction de contraintes CSP. Pour faciliter la résolution des CSP, les chercheurs orientent de plus en plus leurs travaux vers des hybridations entres des méthodes exactes et des méthodes approchées en particulier entre programmation par contraintes et recherche locale. Les résultats obtenus par ce type d'approche depuis quelques années nous ont encouragés à poursuivre cette démarche afin de proposer des solveurs plus efficaces qui tirent leurs performances des avantages respectifs des méthodes combinées. A noter que parmi les méthodes de recherche locale, les champs d'action pour l'hybridation sont très vastes et nous nous sommes limités aux méthodes réputées peu gourmandes en temps de calcul ou en ressources machines, et donc nous avons exclus délibérément les méthodes à base de population du travail que nous avons réalisé.

Dans cette thèse, nous avons proposé et étudié une nouvelle méthode de résolution hybride pour les problèmes de satisfaction de contraintes. Nous avons nommé cette méthode *Tabu-NG* pour *Tabu with Nogood-Recording*. Le nom est un peu réducteur car il s'agit d'une hybridation d'algorithme de filtrage, de propagation de contraintes, de Recherche Tabou et de gestion de *nogoods*, cependant nous pensons qu'elle porte principalement son innovation par la combinaison de Tabu Search et de *nogoods*. La méthode a été appliquée sur deux types de problèmes. Le premier est l'affectation des fréquences dans les réseaux de radiocommunications militaires, en particulier les problèmes proposés de 1993 (instances fournies dans le cadre du projet européen CALMA) jusqu'à 2010 (instances fournies dans le cadre d'un projet de recherche avec le CELAR). Le deuxième est le problème académique de *k*-coloration de graphe. Dans les deux problèmes nous avons traité des contraintes unaires et binaires, cependant l'affectation de fréquences nous a aussi permis d'évaluer les performances de cette méthode pour des contraintes n-aires et pour l'optimisation de fonction avec satisfaction de contraintes. Il est clair que ces deux problèmes ne sont pas suffisants pour établir la généricité de la méthode mais ses principes sont généraux et elle peut s'appliquer sur d'autres CSP. Elle peut par ailleurs accueillir des heuristiques spécifiques aux problèmes, nous l'avons d'ailleurs pratiqué sur les problèmes cités, et en ce sens nous pensons pouvoir qualifier la méthode de métaheuristique sans abuser de cette définition.

Dans le premier chapitre, nous avons introduit les problèmes de satisfaction de contraintes en montrant la capacité du CSP à modéliser un très grand nombre de problèmes combinatoires. Ensuite nous avons présenté les principales méthodes de résolution exactes et approchées d'un CSP. La résolution par des méthodes exactes repose généralement sur l'utilisation d'une démarche constructive avec un mécanisme de *backtrack* qui réagit face à un blocage. Nous avons fourni une description assez fouillée sur les techniques d'amélioration de ce type de méthodes, soit en essayant de détecter le blocage le plus tôt possible, soit en essayant de faire le moins de *backtrack* possible pendant la recherche, soit en combinant les deux. En particulier nous avons revu les différentes versions de l'algorithme d'arc-consistance et nous avons tenté de présenter les avantages et inconvénients de celles-ci. Dans la section dédiée aux méthodes approchées ou aux métaheuristiques de résolution, nous nous sommes concentrés sur la recherche locale et les mécanismes d'intensification et de diversification. Nous avons aussi relevé l'intérêt de l'hybridation des méthodes exactes avec des méthodes approchées et notamment la PPC avec la recherche locale. Nous avons présenté deux

méthodes de résolution hybride très intéressantes et parmi les plus performantes actuellement, *CN-Tabu* et *Decision-Repair* en montrant la particularité de chacune d'elles. Nous avons utilisé ces deux méthodes comme point de départ de notre réflexion pour proposer la nouvelle méthode de résolution hybride *Tabu-NG*. Le chapitre se termine par une description des concepts généraux de *Tabu-NG*. Il s'agit donc d'une méthode qui hybride la programmation par contraintes, en manipulant des configurations partielles et en appliquant la propagation à chaque nœud de l'arbre, avec une recherche locale responsable de la définition du voisinage et de la réparation de l'inconsistance d'une configuration partielle. *Tabu-NG* intègre aussi une recherche de *nogoods*, responsables des blocages rencontrés, et un mécanisme d'apprentissage sur l'espace de recherche basé sur les *nogoods* (utilisation d'une liste Tabou de *nogoods*). Dans *Tabu-NG*, chaque composante à son rôle propre : la propagation permet de filtrer les domaines des variables libres et de réduire l'espace de recherche ; la gestion des *nogoods* permet de faire des *backtrack* intelligents et de créer des coupes sur l'espace de recherche ; la recherche locale permet d'apporter des mécanismes d'intensification et de diversification.

Au second chapitre, nous présentons la méthode *Tabu-NG* sur un problème réel, l'affectation de fréquences pour les réseaux de radiocommunications militaires. Dans le cadre d'un contrat de recherche, le CELAR a proposé une nouvelle modélisation générique pour les problèmes d'affectation de fréquences incluant tous les modèles définis depuis 1993 avec de nouveaux types de contraintes d'interférences et de nouveaux modes d'optimisation. Ce modèle est introduit au début de ce chapitre avec tous les modes d'optimisation inclus. Il est assez complexe et fait appel à des notions avancées sur le plan de la modélisation des problèmes physiques tels les interférences multiples. Un état de l'art sur les méthodes de résolution existantes a montré que le problème proposé a été très peu étudié dans la littérature, cependant il est suffisant global pour mettre au point une méthode réutilisable sur des problèmes antérieurs pour lesquels nous disposions de travaux de recherche conséquents et de nombreux résultats publiés. Ainsi pour évaluer *Tabu-NG* nous avons considéré deux jeux d'instances. Le premier est un ensemble d'instances publiques de taille moyenne notées CELAR et GRAPH ; ces jeux sont très bien référencés mais ils ne couvrent pas tous les éléments des problèmes réels. Le deuxième est un ensemble d'instances privées fourni par le CELAR en 2008 notées SCEN ; ces jeux font référence au problème complet mais les seules comparaisons disponibles sont celles dont dispose le CELAR. Notons que le CELAR a fait un travail considérable en lançant ses outils d'affectation sur ces instances pour nous fournir pour le moins des bonnes solutions si ce n'est des solutions optimales.

Nous avons d'abord appliqué la méthode *Tabu-NG* sur les instances publiques pour les problèmes *MinFreq* (minimiser le nombre de fréquences utilisées) et *MinSpec* (minimiser la largeur du spectre utilisé). Elles nous ont permis de tester les composantes de la méthode sur des contraintes unaires et binaires : propagation des contraintes, filtrage des domaines, identification des *nogoods*, heuristique de choix du voisinage, heuristique de réparation, listes et durées Tabou. Pour ce problème, et vue la taille moyenne des instances, nous avons pu identifier les *nogoods* en appliquant des méthodes de la littérature. La taille de la liste Tabou posait cependant un problème d'explosion éventuelle, mais la notion de dominance entre *nogoods* a permis de contenir la taille de la liste (très faible variation en régime de croisière de la méthode) et de ne pas alourdir les temps d'accès à cette liste. Sur ces instances *Tabu-NG* est au niveau des meilleures méthodes connues dans la littérature ; certaines instances ont même été améliorées et constituent une nouvelle référence. Ensuite, nous avons appliqué *Tabu-NG* sur les instances privées. La première difficulté a été la propagation des contraintes n-aires ; nous avons proposé un renforcement de ce type de contraintes par l'écriture de nouvelles contraintes binaires qui viennent accélérer le processus de décision sur les variables.

La deuxième difficulté a été l'identification et la gestion des *nogoods* du fait de la taille gigantesque des instances à traiter ; nous avons introduit la notion de *nogood* partiel qui revient à définir des *nogoods* inexacts en stockant seulement une partie des informations des *nogoods* complets. Une analyse sur la taille de la liste Tabou des *nogoods* stockés a montré que la taille de cette liste est restée parfaitement maîtrisée pendant la recherche. Enfin sous certaines conditions nous avons introduit une procédure de restart qui bénéficie de l'apprentissage des *nogoods* pour redémarrer *Tabu-NG* très tôt sur de nouvelles branches. Une stratégie d'optimisation a été proposée dans ce chapitre pour les 9 modes d'optimisation du problème. *Tabu-NG* a été évaluée par le CELAR sur les 30 instances fournies (soit potentiellement 270 cas à traiter) ; elle a rempli toutes les exigences de performances demandées pour tous les objectifs traités : au moins 80% de résultats supérieurs ou égaux aux meilleurs connus pour chaque mode d'optimisation appliqué sur les 30 instances.

Le troisième chapitre présente l'application de la méthode *Tabu-NG* sur le problème de *k*-coloration de graphe. Ce problème a été très utilisé dans la littérature et constitue une bonne plateforme d'étude. La première partie de ce chapitre est consacrée à la description du problème, des instances utilisées ainsi que de différentes méthodes de résolution proposées dans la littérature selon 3 classes, la recherche locale, l'hybridation de méthodes approchées et les méthodes de propagation de contraintes. Toutes les études ont été effectuées sur les instances DIMACS qui sont très référencées. L'application directe de *Tabu-NG* implémentée pour l'affectation de fréquences proche de la *T*-coloration avec ensembles sur la *k*-coloration de graphe n'a pas donné de bons résultats notamment en termes de temps de calcul. Nous avons tout d'abord changé la procédure de propagation et de filtrage des domaines, puis nous avons fait un travail d'ingénierie logicielle sur les structures de données utilisées pour accélérer les mécanismes de propagation. Ensuite, nous avons appliqué *Tabu-NG* modifiée sur les instances fournies ; la méthode s'est montrée très rapide mais sans efficacité en termes de résultats, loin des optimums connus dans la littérature. Nous avons donc mené un ensemble de travaux sur les heuristiques en changeant la durée Tabou, les heuristiques de définition de voisinage et de réparation d'une configuration inconsistante et en proposant un mécanisme de diversification. Les résultats finaux étaient plus favorables, proches des meilleures méthodes connues dans la littérature sur cette catégorie de problème, mais sans toutefois apporter d'amélioration sur les instances. Nous avons constaté en particulier que les *nogoods* tels que nous les avions définis pour l'allocation de fréquences n'ont aucune incidence sur la *k*-coloration ; la symétrie des problèmes en *k*-coloration rend l'apprentissage sur les coupes beaucoup plus ardu, il faudrait des mécanismes d'apprentissage sur la symétrie elle-même. En dernier lieu, une étude sur la gestion des phases d'intensification et de diversification en extension et en réparation nous a permis d'améliorer vraiment le comportement de la méthode sans en modifier la procédure générale. Nous avons en particulier réussi à trouver l'optimalité sur un des problèmes *flat* particulièrement difficile.

Au cours des études menées pendant cette thèse, la méthode a montré plusieurs faiblesses qui ouvrent la voie à de nouvelles recherches afin de gagner en maturité, ou pourrait-on dire en robustesse, pour passer sur d'autres problèmes plus efficacement. Nous proposons en particulier deux grandes pistes sur l'optimisation des problèmes sous contraintes lorsqu'ils sont non-réalisables et sur la définition de *nogood* pour prendre en compte les questions de symétries dans les espaces de recherche.

Tabu-NG a montré son efficacité pour chercher la réalisabilité pour un CSP donné notamment pour AFD. Pour les modes d'optimisation qui consistent à minimiser une somme pondérée de contraintes violées (problème d'optimisation sous contraintes non réalisable), nous avons proposé dans cette thèse une stratégie qui repose entièrement sur la recherche d'une solution réalisable du problème relâché. Cependant si l'ensemble de solutions

réalisables du niveau relâché est très grand, la recherche d'une solution qui minimise cette somme est difficile. Nous envisageons de procéder d'une façon différente en intégrant dans l'heuristique de choix d'une configuration voisine, non seulement l'évaluation de la consistance, mais aussi la minimisation d'une fonction d'évaluation. La visibilité directe de la fonction à optimiser au même niveau que la réalisabilité devrait permettre à la méthode d'accéder autrement aux bonnes solutions, via des solutions non réalisables, et pas seulement via les solutions exclusivement réalisables.

Pour les problèmes de *k*-coloration, la notion de *nogood* n'a pas été suffisamment approfondie au cours de la thèse. Nous avons constaté que la liste Tabou des *nogoods* ne permet pas de guider correctement le processus de recherche et de ce fait alourdit inutilement les temps de calcul du fait de la gestion de la liste. Nous croyons toujours qu'une bonne exploitation des *nogoods* rencontrées pendant la recherche est un élément clef de *Tabu-NG*. Cependant la définition des *nogoods* demande à être adaptée au problème ; en particulier les symétries des solutions, ou des sous-espaces, en *k*-coloration nous paraissent une excellente piste de réflexion pour définir de nouveaux *nogoods* et mieux maîtriser leur stockage et leur exploitation. Par exemple, dans un problème de coloriage de graphe, un *nogood ng* = $\{(x_1,1)(x_2,2)(x_4,6)\}$ inclut directement plusieurs autres *nogoods* comme $\{(x_1,2)(x_2,1)(x_4,6)\}$ et $\{(x_1,1)(x_2,6)(x_4,2)\}$. On voit donc que la notion de dominance doit être élargie avec une plus grande généricité que la simple prise en compte des couples (*variables, valeurs*) pour établir des coupes plus productives pour la méthode.

Publications

Conférences Internationales avec actes et comité de sélection :

[1] **M. Dib**, R. Abdallah; A. Caminada, 2010. Arc-consistency in Constraint Satisfaction Problems: A survey, 2nd International Conference on Computational Intelligence, Modelling and Simulation Bali, Indonesia, 28–30 September 2010.

[2] **M. Dib**, A. Caminada, H. Mabed. Frequency management in Radio military Networks. The 10th INFORMS Telecommunications Conference, Concordia University (CANADA) , May 5 - 7, 2010

[3] **M. Dib**, A. Caminada, H. Mabed. A New Method to Solve and Optimize FAP with N-ary Constraints. 8th International Conference of Modeling and Simulation - MOSIM'10 - May 10-12, 2010 - Hammamet - Tunisia.

[4] **M. Dib**, A. Caminada, H. Mabed. MinOrder optimization in FAP with NoGood-Tabu Search. 39th International Conference on Computers & Industrial Engineering (CIE39), Troyes France, July 6-8, 2009.

[5] **M. Dib**, A. Caminada, H. Mabed. A fast Algorithm to solve frequency assignment problem. 6th international conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming for Combinatorial Optimization Problems, Pittsburgh USA, May 27-31, LNCS Springer, 2009.

[6] **M. Dib**, A. Caminada, H. Mabed. Nogood Recording with Tabu Search for CSP (Application to FAP). Third Asia International Conference on Modelling & Simulation, 25-26 May, Indonesia - Bali, IEEE, 2009.

[7] **M. Dib**, A. Caminada, H. Mabed. Constraint Propagation with Tabu List for Min-Span Frequency Assignment Problem. 2nd international conference on Modelling, Computation and Optimization in Information Systems and Management Sciences MCO2008, Springer CCIS, CCIS14, pp. 97-106, 2008.

Conférences nationales avec actes et comité de sélection :

[1] **M. Dib**, A. Caminada. H. Mabed, Renforcement des contraintes n-aires en binaires dans un FAP. 11ème conférence ROADEF'10, 24-26 février, Toulouse, 2010.

[2] **M. Dib**, I. Devarenne, H. Mabed, A. Caminada. Etude comparative de recherche locale et de propagation de contraintes en CSP n-aire. 10ème conférence ROADEF'09, 10-12 février, Nancy, 2009.

[3] J. Hu, **M. Dib**, A. Caminada, H. Mabed, Recherche de zone de blocage dans un graphe. 10ème conférence ROADEF'09, 10-12 février, Nancy, 2009.

[4] **M. Dib**, A. Caminada, H. Mabed. Propagation de Contraintes avec une liste Tabou pour le CSP. JFPC2008, PP. 409 – 413, 2008.

[5] **M. Dib**, A. Caminada, H. Mabed, J. Hu. Propagation de contraintes et gestion de Nogood pour le CSP. 9ème conférence ROADEF'08, 25-27 février, Clermont, 2008.

Bibliographie

[6] G. Lin, D. Xu, Z.-Z. Chen, T. Jiang, J. Wen, and Y. Xu. An efficient branch-and-bound algorithm for the assignment of protein backbone NMR peaks. In Proceedings of the IEEE Computer Society Bioinformatics Conference (CSB'02), 2002.

[7] I. Méndez-Díaz and P. Zabala. A branch-and-cut algorithm for graph coloring. Discrete Appl. Math., 154(5) :826–847, 2006.

[8] A. Hertz and M Widmer. Guidelines for the use of meta-heuristics in combinatorial optimization. European Journal of Operational Research, 151:247–252, 2003.

[9] M. Widmer, A. Hertz, and D. Costa. Ordonnancement de la Production, chapitre Les Métaheuristiques. HERMES science publications, 2000

[10] J.-C. Régin, A filtering algorithm for constraints of difference in CSPs, dans Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94), pages 362-367, 1994

[11] R. Mohr and T.C. Henderson, 1986. Arc and path consistency revisited. Artificial Intelligence, 28:225–233

[12] A.K Mackworht, 1977. Consistency in Networks of Relations. Artificial Intelligence 8(1) : 99-118

[13] A.K Mackworth et Freuder, 1985. The complexity of Some Polynomial Network Consitency Algorithms for Constraint Satisfaction Problems. Artificial Intelligence 25:65-74

[14] C. Bessière, 1994. Arc-consistency and arc-consistency again. Artificial Intelligence, 65(1) : 179

[15] Bessière, E.C. Freuder and J-C Régin, 1999. Using constraint metaknowledge to reduce arc consistency computation. Artificial Intelligence, 107(1):125-148

[16] C. Bessière, E.C Freuder and J-C. Régin 1995,Using Inference to Reduce Arc Consistency Computation. IJCAI-95, Montréal, Canada, pp 592-598

[17] C. Bessière and J-C. Régin, 2001. Refining the basic constraint propagation algorithm. IJCAI'01, pages 309-315, Seattle, WA, USA

[18] Y. Zhang and R. Yap, 2001. Making ac-3 an optimal algorithm. IJCAI'01, pages 316-321, Seatle, WA, USA

[19] Bessière, C., and Regin, J.-C. 2001. Refining the basic constraint propagation algorithm. In IJCAI-2001, 309–315

[20] C. Lecoutre, F. Boussemart, and F. Hemery, 2003. Exploiting multidirectionnality in coarse-grained arc consistency algorithm. CP'03, pages 480–494, Cork, Ireland

[21] A. Chmeiss and P. Jégou, 1998. Efficient path consistency propagation. Journal on Artificial Intelligence Tools, 7(2):79-89

[22] M.R. van Dongen, 2002. Ac-3d an efficient arc-consistency algorithm with a low spacecomplexity. CP'02, pages 755–760, Ithaca, NY, USA

[23] C. Bessière, Constraint propagation, Tech. report, LIRMM 06020, CNRS/University of Montpellier, March 2006

[24] M. Dib, R. Abdallah; A. Caminada, 2010. Arc-consistency in Constraint Satisfaction Problems: A survey, 2nd International Conference on

Computational Intelligence, Modelling and Simulation Bali, Indonesia, 28–30 September 2010

[25] M. C. Cooper. An optimal k-consistency algorithm. Artif. Intell., 41(1) :89{95, 1989

[26] C. Bessière et J.C Régin, 1996. Mac and combined heuristics : Two reasons to forsake fc (and cbj ?) on hard problems. In CP, pages 61-75, 1996

[27] M. Haralick and G.L. Elliot, Increasing tree-search efficiency for constraint satisfaction problems, Artificial Intelligence 14 (1980), 263–313

[28] Dent, M. J. et R. E. Mercer (1994). Minimal Forward Checking, Dans Proceedings ICTAI, 432-438, New Orleans, Louisiana, USA

[29] Kwan, A. C. M. et E. P. K. Tsang (1996). Minimal Forward Checking with Backmarking, Dans Proceedings ICTAI, 290-298, Toulouse, France

[30] E. Tsang. No more "partial" and "full looking ahead". Artificial Intelligence, 98:351-361, 1997

[31] E. Tsang. Foundations of constraint satisfaction. 1993

[32] Principles of constraint programming By Krzysztof R. Apt. 1999

[33] http://homepages.laas.fr/lopez/publis/Contraintes-GRP.pdf

[34] R.M. Stallman and G. J. Sussman. Forward reasoning and dependency directed backtracking in a system for computer aided circuit analysis. Artificial Intelligence, 9:135–196, 1977

[35] Handbook of constraint programming, Volume 35. By Francesca Rossi, Peter Van Beek, Toby Walsh. 2006

[36] M. Moskewicz, C. Madigan,Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In Proceedings of 39th Design Automation Conference, Las Vegas, 2001

[37] G. Katsirelos, F. Bacchus: Unrestricted Nogood Recording in CSP Search. CP 2003: 873-877

[38] J. Gaschnig. Performance measurement and analysis of search algorithms. Rapport technique, 1979

[39] R. Dechter et D. Frost. Backjumping-based backtracking for constraint satisfaction problem. Artificial Intelligence, 136:147-188, 2002

[40] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. Artificial Intelligence, 41:273-312, 1990

[41] P. Prosser. Forward checking with backmarking. In Constraint Processing, Selected Papers, pages 185-204, 1995

[42] M.L. Ginsberg. Dynamic backtracking. Journal of Artificial Intelligence Research (JAIR), 1:25-46,1993

[43] P. Prosser, MAC-CBJ: maintaining arc consistency with conflict-directed backjumping, Rapport technique Research Report/95/177, Dept. of Computer Science, University of Strathclyde, 1995

[44] T. Schiex et G. Verfaillie, Nogood Recording for Static and Dynamic Constraint Satisfaction Problem, International Journal of Artificial Intelligence Tools, 3(2):187-207, 1994

[45] N. Jussien, R. Debruyne et P. Boizumault, Maintaining arc-consistency within dynamic backtracking, dans Principles and Practice of Constraint Programming, pages 249-261, 2000

[46] J. Dréo, A. Pétrowski, P. Siarry et E. Taillard. Métaheuristiques pour l'optimisation difficile. Eyrolles, 2003

[47] J .K. Hao, P. Galinnier, et M. Habib. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. Revue d'Intelligence Artificielle, 13(2) : 283-324, 1999
[48] P. Van Hentenryck. Constraint Satisfaction in Logic Programming. MIT Press, 1989
[49] P. Baptiste, C. Le Pape, and W. Nuijten. Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Kluwer, 2001
[50] F. Glover et M. Laguna. Tabu Search. Kluwer Academic, Dordrecht, 1997
[51] F. Glover. Tabu Search - part I. ORSA Journal on Computing, 1(3):190–206, 1989
[52] F. Glover. Tabu Search - part II. ORSA Journal on Computing, 2 :4–32, 1990
[53] P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. Computing, 44(4) :279–303, 1990
[54] J-K. Hao, R. Dorne, and P. Galinier. Tabu search for frequency assignment in mobile radio networks. Journal of Heuristics, 4(1):47–62, 1998
[55] J-K. Hao and P. Galinier. Tabu search for maximal constraint satisfaction problems. Lecture Notes in Computer Science, 1330 :196–208, 1997
[56] A. Hertz, E. Taillard, and D. De Werra. A tutorial on tabu search. In Proc. Of Giornate di Lavoro AIRO'95 (Enterprise Systems: Management of Technological and Organizational Changes), pages 13–24, Italy, 1995
[57] R. Dorne and J-K. Hao. Tabu Search for graph coloring, T-coloring and Set Tcolorings. In I.H. Osman S. Voss, S. Martello and C. Roucairol, editors, Metaheuristics : Advances and Trends in Local Search Paradigms for Optimization, chapter 6, pages 77–92. Kluwer, 1998
[58] R. Battiti and G. Tecchiolli. The reactive tabu search. ORSA Journal on Computing, 6(2) :126–140, 1994
[59] F. Glover, J. P. Kelly, and M. Laguna. Genetic algorithms and tabu search : hybrids for optimization. Comput. Oper. Res., 22(1) :111–134, 1995
[60] E. J. Anderson, C. A. Glass, and C. N Potts. Local search in combinatorial optimization, chapter Machine scheduling, pages 361–415. John Wiley & Sons, 1997
[61] P. Galinier and A. Hertz. A survey of local search methods for graph coloring. Comput. Oper. Res., 33(9) :2547–2562, 2006
[62] J. B. Orlin, A. P. Punnen, and A. S. Schulz. Approximate local search in combinatorial optimization. In SODA '04 : Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, pages 587–596, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics
[63] T. Stützle. Local Search Algorithms for Combinatorial Problems : Analysis, Improvements, and New Applications. PhD thesis, Am Fachbereich Informatik der Technischen Universitat Darmstadt vorgelegte, 1998
[64] C. Duhamel. Un Cadre Formel pour les Méthodes par Amélioration Itérative – Application à deux problèmes d'Optimisation dans les Réseaux -. PhD thesis, Université Blaise Pascal - Clermont-Ferrand II, 2001
[65] B. Weinberg. Analyse et résolution approchée de problèmes d'optimisation combinatoire : application au problème de coloration de graphes. PhD thesis, Université des sciences et technologies de lille, 2004
[66] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598) :671–680, 13 May 1983
[67] W. ben Ameur. Computing the initial temperature of simulated annealing. Computational Optimization and Applications, 29(3) :369–385, 2004

[68] A. Caminada, J-K. Hao, J-L. Lutton, and V. Martin. Recherche opérationnelle et réseaux, chapter Réseaux de communications. Gerd Finke, 2002

[69] EG Talbi and T Muntean. Hill climbing, simulated annealing and genetic algorithm: a comparative study and application to the mapping problem. In Proc. of the Twenty-Sixth Hawaii International Conference on System Sciences, volume 2, pages 565–573, 1993

[70] P. Hansen and N. Mladenovic. Recherche à voisinage variable. Technical report, Les cahiers de Gerad G-2000-08, 2000

[71] P. Hansen, N. Mladenovic, and D Urosevic. Variable Neighborhood Search and local branching. Computers and Operations Research, 33(10) :3034–3045, 2006

[72] H. Lourenço, O. Martin, and T. stutzle. Iterated Local Search, chapter Iterated Local Search, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002

[73] N. Mladenovic and P. Hansen. Variable Neighborhood Search. Comps. in Opns.Res., 24 :1097–1100, 1997

[74] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. European Journal of Operational Research, 151 :379–388, 2003

[75] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. Computing, 39(4) :345–351, 1987

[76] H. Lourenço, O. Martin, and T. stutzle. A beginner's introduction to iterated local search. In Proceedings of MIC 2001, pages 1–6, Porto, Portugal, July 2001

[77] T. Wong, P. Bigras, and B. de Kelper. A multi-neighborhood and multi-operator strategy for the uncapacitated exam proximity problem. Systems, Man and Cybernetics, 2005 IEEE International Conference on, 4 :3810– 3816, 2005

[78] I. Devarenne. Etudes en recherche locale adaptative pour l'optimisation combinatoire. PH.D. thesis, Université de Technologie de Belfort Montbéliard, France. 2007

[79] U. Montanari, Networks of constraints: Fundamental properties and applications to picture processing, Information Sciences, Volume 7, 1974, Pages 95-132

[80] M. R. Garey and D. S. Johnson. Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman & Company, San Francisco, 1978

[81] C. H. Papadimitriou. Computational Complexity. Addison-Wesley Publishing Company, 1994

[82] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In Alan Borning, editor, Principles and Practice of Constraint Programming, volume 874 of Lecture Notes in Computer Science. Springer, May 1994. PPCP'94: Second International Workshop, Orcas Island, Seattle, USA)

[83] P. David. A constraint-based approach for examination timetabling using local repair techniques. In Proceedings of the Second International Conference on the Practice And Theory of Automated Timetabling (Patat'97), pages 132-145, Toronto, Canada, August 1997

[84] C. Gervet. Large combinatorial optimization problem methodology for hybrid models and solutions (invited talk). In JFPLC, 1998

[85] G. Pesant and M. Gendreau. A view of local search in constraint programming. In Proc. of the Principles and Practice of Constraint Programming, pages 353-366. Springer-Verlag, 1996

[86] E. T. Richards and E. B. Richards. Non-systematic search and learning: An empirical study. In Proc. of the the Conference on Principles and Practice of Constraint Programming, Pisa, 1998

[87] Andrea Schaerf. Combining local search and look-ahead for scheduling and constraint satisfaction problems. In Proc. of the 15th International Joint Conf. n Arti_cial Intelligence (IJCAI-96), pages 1254-1259, Nagoya, Japan, 1997. Morgan Kaufmann

[88] M. Yokoo. Weak-commitment search for solving constraint satisfaction problems. In Proceedings of AAAI, 1994

[89] R. Russell, Hybrid heuristics for the vehicle routing problem with time windows, TRansportation Science, 29 :156–166, 1995

[90] Y. Caseau et F. Laburthe, Heuristics for large constrained routing problems, Journal of Heuristics, 5:281–303, 1999

[91] N. Jussien and O. Lhomme, Local search with constraint propagation and conflict-based heuristics. Artificial Intelligence, 139(1) :21–45, 2002

[92] S. Prestwich, Incomplete dynamic backtracking for linear pseudo-boolean problems, Annals of Operations Research, 130 :57–73, 2004

[93] M. Vasquez, A. Dupont, and D. Habet. Consistent Neighbourhood in a Tabu Search. MIC-Kluwer, Book "Metaheuristics : Progress as real Problem Solvers", chapter 17, pages 367–386, June 2005

[94] N. Jussien and O. Lhomme, The path-repair algorithm. CP99 Post-conference workshop on Large scale combinatorial optimisation and constraints, Electronic Notes in Discrete Mathematics, 4, Elsevier Science, 2000

[95] S. Demassey, Méthodes hybrides de programmation par contraintes et programmation linéaire pour le problème d'ordonnancement de projet à contraintes de ressources, Thèse de doctorat, Université d'Avignon et des Pays de Vaucluse, 2003

[96] N. Jussien et O. Lhomme, Dynamic domain splitting for numeric csp. In European Conference on Artificial Intelligence, 224–228. 1998

[97] M. Wallace, Practical Applications of Constraint Programming. Constraints, 1:139–168, 1996

[98] N. Jussien, e-constraints: explanation-based constraint programming. In CP01 Workshop on User-Interaction in Constraint Satisfaction, Paphos, Cyprus, 1 December 2001

[99] Handbook of Metaheuristics by Fred Glover and Gary Kochenberger, 2003

[100] Pascal Van Hentenryck, Yves Deville and Choh-Man Teng, 1992, A Generic Arc-Consistency Algorithm and its Specialization. Artificial Intelligence, 57:291—321

[101] R. Bayardo Jr. and D. Miranker. A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, pages 298–304, Portland, Oregon, 1996

[102]I. Lynce and J. P. Marques-Silva, The Effect of Nogood Recording in DPLL-CBJ SAT Algorithms, ERCIM/CologNet Workshop on Constraint Solving and Constraint Logic Programming, June 2002

[103]Bayardo, R. J. and Schrag, R. 1996. Using CSP Look-Back Techniques to Solve Exceptionally Hard SAT Instances. In Proc. Second Int'l Conf. on Principles and Practice of Constraint Programming (Lecture Notes in Computer Science v. 1118), Springer, 46-60

[104]I. Devarenne, H. Mabed, A. Caminada, (2006). Intelligent neighborhood exploration in local search heuristics, ICTAI'06 : Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence, p. 144-150, Washington D.C., USA, November 2006

[105] I. Devarenne, A. Caminada, H. Mabed, (2005). Analysis of Adaptive Local Search for the Graph Coloring Problem, 6th Metaheuristics International Conference – MIC 2005, Vienne Autriche, August 2005

[106] Eugene C. Freuder, Rina Dechter, Mattew L. Ginsberg, Bart Selman et Edward P. K. Tsang. Systematic versus stochastic constraint satisfaction. IJCAI. 1995, p 2027-2032

[107] Renaud, D., & Caminada, A. (1997). Evolutionary methods and operators for the frequency assignment problem. SpeedUp, 11/2, 27-32

[108] Jaimes-Romero F, Munoz-Rodreguez D (1996) Channel Assignment in Cellular Systems Using Genetic Algorithms proc. IEEE VTC, n° 45, 741-745

[109] Malek Rahoual, Patrick Siarry (2006) Réseaux informatiques : conception et optimisation, Editions TECHNIP, livre de 345 pages

[110] Palpant M., Oliva C., Artigues C., Michelon P., Didi Biha M. et Defaix T., Affectation de fréquences avec sommation des perturbateurs : modèles et heuristiques, dans 4ème Conférence Francophone de MOdélisation et SIMulation, MOSIM'03 , pages 299–304, Toulouse, France, 2003

[111] Vlasak J. et Vasquez M., Résolution du problème d'attribution de fréquences avec sommation de perturbateurs, dans 5ème congrés de la société Française de Recherche Opérationnelle et d'Aide à la Décision, ROADEF 2003 , Avignon, France, 2003

[112]Sarzeaud O., Allocation de fréquences par échantillonnage de gibbs, recuit simulé et apprentissage par renforcement, dans 5ème congrès de la société Française de recherche Opérationnelle et d'Aide à la Décision, ROADEF 2003 , Avignon, France, 2003

[113]Oliva C., Allocation de fréquences par échantillonnage de gibbs, recuit simulé et apprentissage par renforcement, dans 5ème congrès de la société Française de Recherche Opérationnelle et d'Aide à la Décision, ROADEF 2003 , Avignon, France, 2003

[114] Dunkin N. W., Bater J. E., Jeavons P. G. et Cohen D. A., Towards high order constraint representations for the frequency assignment problem, Rapport technique, University of London, Egham, Surrey, UK, 1998

[115] Mannino C. et Sassano A., An enumerative algorithm for the frequency assignment problem, Discrete Applied Mathematics, 129(1) :155–169, 2003

[116] K.I. Aardal, C.P.M. van Hoesel, A.M.C.A. Koster, C. Mannino and A. Sassano, Models and solution techniques for frequency assignment problems, *Annals of Operations Research* **153** (2007), pp. 79–129

[117] Gondran A., Baala O., Caminada A. and Mabed H, Hypergraph T-Coloring for Automatic Frequency Planning problem in Wireless LAN, 19th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications 2008 (PIMRC 2008), Cannes, France, September 15-18, 2008

[118] Voudouris, C. & Tsang, E. P. K. (1999). Solving the radio link frequency assignment problem using guided local search. Frequency Assignment, Sharing and Conservation Systems (Aerospace), Research and Technology Organization (RTO) Meeting Proceedings 13, North Atlantic Treaty Organization (NATO), 14a-1-11

[119] http://fap.zib.de/problems/CALMA

[120] http://www.inra.fr/bia/T/schiex/Doc/CELAR.shtml

[121] H. P. van Benthem. GRAPH Generating Radio Link Frequency Assignment Problems Heuristically. PhD thesis, Delft University of Technology, 1995

[122] http://uma.ensta.fr/conf/roadef-2001-challenge/

[123] M. Vasquez, A. Dupont, and D. Habet. Consistent Neighbourhood in a Tabu Search. MIC-Kluwer, Book "Metaheuristics: Progress as real Problem Solvers", chapter 17, pages 367–386, June 2005

[124] A. Dupont. Étude d'une méta-heuristique hybride pour l'affectation de fréquences dans les réseaux tactiques évolutifs. PhD thesis, Université Montpellier II, Octobre 2005

[125] C. Oliva. Méthodes hybrides de programmation linéaire et propagation de contraintes. PhD thesis, Université d'Avignon, 16 avril 2004

[126] I. Devarenne. Etudes en recherche locale adaptative pour l'optimisation combinatoire. PH.D. thesis, Université de Technologie de Belfort-Montbéliard, France. 2007

[127] A. Gondran. Modélisation et optimisation des réseaux locaux sans fil Wireless local area network modeling and optimization. Ph.D thesis, Université de Technologie de Belfort-Montbéliard, 8 décembre 2008

[128] M. Dib, A. Hakim Mabed, and A. Caminada. Renforcement des contraintes n-aires en binaires dans un FAP. In ROADEF'10, 11e conférence de la Société Française de Recherche Opérationnelle et d'Aide á la Décision, February 2010

[129] F. Comellas and J. Ozón. Graph coloring algorithms for assignment problems in radio networks. In J. Alspector, R. Goodman, and T. X. Brown, editors, Applications of Neural Networks to Telecommunications 2, pages 49–56. Lawrence Erlbaum Ass., Inc., Publis., Hillsdale, NJ, 1995

[130] R. Dorne. Étude des méthodes heuristiques pour la coloration, la T-coloration et l'affectation de fréquences. PhD thesis, Université de montpellier II, mai 1998

[131] R. Dorne and J-K. Hao. Tabu Search for graph coloring, T-coloring and Set Tcolorings. In I.H. Osman S. Voss, S. Martello and C. Roucairol, editors, Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, chapter 6, pages 77–92. Kluwer, 1998

[132] S. R. Tiourine, C. A. J. Hurkens, and J. K. Lenstra. Local Search Algorithms for the Radio Link Frequency Assignment Problem. Telecommunication Systems, 13 :293_314, 2000

[133] P. Galinier, S. Bisaillon, M. Gendreau, and P. Soriano. Solving the Frequency Assignment Problem with Polarization by Local Search and Tabu. the 6th Triennal Conference of the International Federation of Operational Research Societies (IFORS'02), University of Edinburgh, UK, 8-12 July 2002

[134] W.D. Harvey and M.L. Ginsberg. Limited Discrepancy Search. in Proceeding of the 14th International Joint Conference on Artificial intelligence (IJCAI-95), pages 607_613, 1995

[135] P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In Principle and Pratice of Constraint Programming, CP'98, October 1998

[136] H. Gavranovic. Local Search for Frequency Assignment Problem with Polarisation. Conference Local Search Two Day Workshop, City University, London, UK, 16-17 April 2002

[137] A. Hertz, D. Schindl, and N. Zu_erey. A Tabu Search for the Frequency Assignment Problem with Polarization. In FRANCORO III, Quebec, Canada, may 2001

[138] A. Hertz, D. Schindl, and N. Zu_erey. Lower Bounding and Tabu Search Procedures for the Frequency Assignment Problem with Polarization Constraints. OR, 3(2) :139-161, 2005

[139] Narendra Jussien, "The versatility of using explanations within constraint programming", Habilitation thesis, Université de Nantes, 2003
[140] Narendra Jussien, "e-constraints: explanation-based Constraint Programming" , CP01 Workshop on User-Interaction in Constraint Satisfaction, 2001
[141] Narendra Jussien, Vincent Barichard, "The PaLM system: explanation-based constraint programming" , Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000, pp. 118-133, 2000
[142] Caminada, A. (1995). "Résolution du Problème de l'Affectation des Fréquences Par Programmation Par Contraintes," Technical Report FT.CNET/BEL/POH/CDI/7195/CA, CNET, Belfort
[143] Palpant, M., Artigues, C., Michelon, P., 2002. A heuristic for solving the frequency assignment problem. In Proceedings of the XI Latin-Iberian American Congress of Operations Research (CLAIO), Concepcion, Chile
[144] Appel K., Haken W., and Koch J. (1977). “Every planar map is four colorable”. Illinois Journal of Mathematics, 21, pp. 429–567
[145] Allen M., Kumaran G., and Liu T. (2002). "A combined algorithm for graph-coloring in register allocation". In Johnson et al. (2002b), pp. 100–111
[146] De Werra D. (1985). "An introduction to timetabling". European Journal of Operational Research, 19(2), pp. 151–162
[147] Gomes C. and Shmoys D. (2002). "Completing quasigroups or latin squares: A structured graph coloring problem". In Johnson et al. (2002b), pp. 22–39
[148] Hossain S. and Steihaug T. (2002). "Graph coloring in the estimation of mathematical derivatives". In Johnson et al. (2002b), pp. 9–16
[149] Lewandowski G. and Condon A. (1996). "Experiments with parallel graph coloring heuristics and applications of graph coloring". In Johnson and Trick (1996), pp. 309–334
[150] T. R. Jensen and B. Toft. Graph Coloring Problems. Wiley Interscience, 1995
[151] Michael A. Trick and David S. Johnson, editors. Cliques, Coloring, and Satisfiability : Proceedings of the Second DIMACS Implementation Challenge. American Mathematical Society, 1993
[152] http://webdocs.cs.ualberta.ca/~joe/Coloring/
[153] D. S. Johnson, C. R. Aragon, L. A. Mcgeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part II, graph coloring and number partitioning. Operations research, 39(3) :378–406, 1991
[154] F.T. Leighton. A graph coloring algorithm for large scheduling problems. Journal of research of the national bureau of standards, (84) :489–505, 1979
[155] N. Zufferey, P. Amstutz, and P. Giaccari. Graph colouring approaches for a satellite range scheduling problem. Journal of Scheduling, 11(4) :263{277, 2008
[156] A. Lim and F. Wang. Robust graph coloring for uncertain supply chain management. In Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)-Track 3-Volume 03. IEEE Computer Society, 2005
[157] P. Galinier and A. Hertz. A survey of local search methods for graph coloring. Computers and operations research, 33(9) :2547{2562, 2006
[158] www.imada.sdu.dk/~marco/gcp/
[159] Lewandowski G. and Condon A. (1996). “Experiments with parallel graph coloring heuristics and applications of graph coloring”. In Johnson and Trick (1996), pp. 309–334

[160] Caramia M. and Dell'Olmo P. (2002a). "k-fullins graphs: a proposal of a new class of benchmarks". Manuscript

[161] Knuth D. (1993). The Stanford GraphBase: a platform for combinatorial computing. ACM press, New York, NY, USA

[162] Steven Prestwich (2002). Coloration neighbourhood search with forward checking. Annals of Mathematics and Artificial Intelligence 34: 327–340, 2002

[163] Schindl D. (2003). "Graph coloring and linear programming". Presentation at First Joint Operations Research Days, Ecole Polytechnique Fédérale de Lausanne (EPFL), available on line at http://roso.epfl.ch/ibm/jord03.html. (June 2005)

[164] Mehrotra A. and Trick M. (1996). A column generation approach for graph coloring. INFORMS Journal On Computing, 8(4), pp. 344–354

[165] Caramia M. and Dell'Olmo P. (2002c). "Vertex coloring by multistage branch and bound". In Johnson et al. (2002b), pp. 40–47

[166] M. Chiarandini. Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems. PhD thesis, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany, 2005

[167] M. Chiarandini and T. Stützle. An application of iterated local search to graph coloring problem. In D. S. Johnson, A. Mehrotra, and M. Trick, editors, Proceedings of the Computational Symposium on Graph Coloring and its Generalizations, pages 112–125, Ithaca, New York, USA, September 2002

[168] DC. Porumbel. Algorithmes Heuristiques et Techniques d'Apprentissage. Applications au Problème de Coloration de Graphe. Thèse de doctorat. Université d'Angers, France 2009

[169] C. Morgenstern. Distributed coloration neighborhood search. In Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge, 1996, pages 335-358

[170] P. Galinier, A. Hertz, and N. Zufferey. An adaptive memory algorithm for the k-coloring problem. Discrete Applied Mathematics, 156(2) :267-279, 2008

[171] Z. Lu and J.K. Hao. A memetic algorithm for graph coloring. European Journal of Operational Research, 203(2) :241-250, 2010

[172] S. Prestwich. Local Search and Backtracking versus Non-systematic Backtracking. In Papers from the AAAI 2001 Fall Symposium on Using Uncertainty within Computation, pp. 109–115. Cape Cod, MA: North Falmouth.