

Variantes d'algorithmes génétiques appliquées aux problèmes d'ordonnancement

THÈSE

présentée et soutenue publiquement le 30 novembre 2006

pour l'obtention du

Doctorat de l'Université du Havre
(spécialité Mathématiques Appliquées et Informatique)

par

Saïd BOURAZZA

Composition du jury

<i>Présidente :</i>	Marie-Claude PORTMANN, Professeur,	École des mines de Nancy, INPL
<i>Rapporteurs :</i>	Gérard DUCHAMP, Professeur, Cyril FONLUPT, Professeur,	Université de Paris Nord Université du Littoral Côte d'Opale
<i>Examineurs :</i>	Frédéric GUINAND, Professeur, Sérigne GUEYE, Maître de conférences,	Université du Havre Université du Havre
<i>Directeur de thèse :</i>	Adnan YASSINE, Professeur,	Université du Havre

Remerciements

Je tiens à remercier les membres de mon jury de thèse :
Marie-Claude PORTMANN pour l'honneur qu'elle m'a fait de présider ce jury.

Gérard DUCHAMP, Cyril FONLUPT pour l'honneur qu'ils m'ont fait d'avoir accepté d'être rapporteurs de ma thèse. C'est pour moi une grande joie que de savoir que des chercheurs d'une telle renommée internationale aient apprécié mon travail.

Je tiens également à remercier chaleureusement Frédéric GUINAND, Sérigne GUEYE pour leurs participations à mon jury et pour leurs lectures attentives de ma thèse.

Je suis extrêmement reconnaissant à Adnan YASSINE pour avoir accepté d'être mon directeur de thèse. Ses très nombreuses remarques, commentaires et suggestions tout au long de ma thèse ont considérablement amélioré à la fois le contenu et la présentation de cette thèse.

Je tiens à exprimer ma profonde gratitude envers tout les membres du Laboratoire de Mathématiques Appliquées du Havre ; les membres du groupe de travail d'optimisation, Professeurs, Maîtres de conférences et Doctorants pour leurs qualités humaines. Nul doute qu'il fut pour moi une importante source de stimulation intellectuelle et que j'ai beaucoup appris avec eux.

Je considère que c'est une chance que d'avoir travaillé avec eux.

Je tiens à remercier TOUS mes amis au Havre, Rouen, Paris, Brest, en France, en Algérie et au Maroc.

*Je dédie cette thèse
à mes parents et à mes frères et leurs familles.
à ma famille à Vienne.*

Table des matières

Introduction générale	xv
-----------------------	----

Partie I Algorithme génétique (AG)	1
------------------------------------	---

Chapitre 1

Algorithme Génétique

1.1 Introduction	3
1.2 Nomenclature de l'algorithme génétique (AG)	3
1.3 Les avantages de l'AG	4
1.4 Principes de base des AG	4
1.5 Fonctionnement des AG	5
1.5.1 Codage du chromosome	5
1.5.2 Génération de la population initiale	6
1.5.3 Méthodes de sélection	6
1.5.4 Opérateurs de croisement	8
1.5.5 Opérateurs de mutation	9
1.5.6 Méthode d'insertion	9
1.5.7 Test d'arrêt	10
1.6 Conclusion	10

Chapitre 2

L'analyse théorique des algorithmes génétiques

2.1 Introduction	13
2.2 Théorie du schéma	14
2.2.1 Définitions fondamentales	14
2.2.2 Effets de la reproduction	15

2.2.3	Effets des croisements	16
2.2.4	Effets des mutations	16
2.3	Modélisation par chaîne de Markov	17
2.3.1	Modélisation de l'algorithme génétique	17
2.3.2	Description rapide de l'algorithme	17
2.3.3	Modélisation	18
2.3.4	Processus de fond (X_k^∞)	20
2.3.5	Processus perturbé (X_k^l)	20
2.3.6	La théorie de Freidlin et Wentzell	23
2.3.7	Résultats de convergence	26
2.4	Conclusion	28

Partie II Le problème du voyageur de commerce (T.S.P.) 29

Introduction

Chapitre 1
Problème du voyageur de commerce (T.S.P.)

1.1	Historique du problème du voyageur de commerce (T.S.P.)	33
1.2	Méthodes de résolutions du problème T.S.P.	35
1.3	Algorithmes déterministes	35
1.3.1	Méthode de séparation et d'évaluation (Branch and Bound) . .	36
1.3.2	Méthode des plans sécants (Cutting plane)	37
1.4	Algorithmes approximatifs	37
1.4.1	Le recuit simulé	37
1.4.2	L'algorithme de colonies de fourmis	39
1.4.3	La méthode recherche tabou	42
1.4.4	Algorithme de Lin et Kernighan	43

Chapitre 2
Algorithme génétique appliqué au problème du voyageur de commerce

2.1	Les représentations possibles du problème	48
2.1.1	La représentation adjacente	48
2.1.2	La représentation ordinale	48

2.1.3	Représentation chemin	50
2.2	Sélection	50
2.3	Opérateurs de croisement	50
2.3.1	Opérateur de croisement cyclique (cycle par Olivier et al. [84]) :	50
2.3.2	Opérateur de croisement de recombinaison des arcs (edrx par Whitley [105]) :	51
2.3.3	Opérateur de croisement de préservation maximale (MPX par Mühlenbein et al.[81] :)	51
2.3.4	Opérateur de croisement d'ordre 1 (order 1 par Davis et al.[38]) :	52
2.3.5	Opérateur de croisement d'ordre 2 (order 2 par Syswerda[99]) :	52
2.3.6	Opérateur de croisement de position (position par Syswerda [99]) :	53
2.3.7	Opérateur de croisement assorti partiel (pmx par Goldberg et al.[51]) :	53
2.3.8	Opérateur de croisement uniforme (uox) :	53
2.4	Opérateurs de mutation	54
2.4.1	Mutation twors :	54
2.4.2	Mutation de centre inverse (cim) :	54
2.4.3	Mutation inverse (im) :	54
2.4.4	Mutation throas :	55
2.4.5	Mutation thrors :	55
2.5	Méthode d'insertion	55
2.6	Résultats numériques	55
2.6.1	Environnement	55
2.6.2	Analyse	56
2.6.3	Comparaison entre les opérateurs de croisement	57
2.6.4	Comparaison entre les opérateurs de mutation	59
2.6.5	Comparaison entre les interactions des opérateurs de croisement et mutation	61
2.6.6	Discussion sur la taille de la population	61
2.7	Conclusion	64

<p>Chapitre 3</p> <p>Comparaison entre les méthodes de sélection appliquées dans l'AG</p>

3.1	Introduction	65
-----	------------------------	----

3.2	Composantes de l'algorithme	65
3.3	Méthode de sélection	66
3.4	Résultats	68

<p>Chapitre 4 Génération de la population initiale concernant le problème T.S.P.</p>

4.1	Introduction	71
4.2	Implémentation de l'algorithme génétique	72
4.2.1	Génération de la population initiale	72
4.3	Résultats numériques	75
4.3.1	Instance Berlin 52	75
4.3.2	Instance Eil 101	78
4.3.3	Instance de Kroa200	80
4.3.4	Instance a280	83
4.4	Conclusion	85

<p>Chapitre 5 Notre variante de l'algorithme génétique</p>

5.1	Introduction	87
5.2	Codage de chromosome	87
5.3	Génération de la population initiale	88
5.4	Opérateurs de croisement	89
5.4.1	Opérateur de croisement de recombinaison des arcs (Whitley 1989)	89
5.4.2	Notre opérateur de croisement Cedrx	90
5.5	Opérateurs de mutation	90
5.6	Implémentation de l'algorithme génétique	90
5.7	Résultats numériques	92

Discussion **95**

Partie III Applications de l'algorithme génétique **99**

<p>Chapitre 1 Le problème de Job Shop (JSP)</p>
--

tel-00126292, version 2 - 8 Mar 2007

1.1	Rappel	103
1.2	Les générateurs de solution d'ordonnancement	105
1.3	Règles de priorité	107
1.4	Formulation du problème JSP	108
1.5	Codage d'une solution de JSP	108
1.6	Implémentation de l'algorithme génétique	110
1.6.1	Représentation	110
1.6.2	Génération de la population initiale	110
1.6.3	Calcul du Makespan	113
1.6.4	Sélection	113
1.6.5	Opérateur de croisement	113
1.6.6	Opérateur de mutation	113
1.6.7	Mécanisme de réparation des individus générés	115
1.6.8	Méthode d'insertion	115
1.7	Résultats numériques	115
1.8	Conclusion	116

<p>Chapitre 2</p> <p>Le problème d'atterrissage des avions</p>
--

2.1	Introduction	119
2.2	Le problème d'atterrissage des avions	120
2.3	État de l'art	120
2.4	Algorithme génétique	122
2.4.1	Définition des paramètres	122
2.4.2	Génération de la population initiale	122
2.4.3	Calcul des heures d'atterrissage des avions	122
2.4.4	Opérateur de croisement	125
2.4.5	Opérateur de mutation	125
2.4.6	Organigramme de notre algorithme génétique	126
2.5	Résultats numériques	126
2.6	Algorithmes de population heuristique	127
2.6.1	Algorithme génétique (AG)	128
2.6.2	Méthode de recherche Scatter (SS)	130
2.6.3	Algorithme Bionomique (BA)	131
2.7	Notre algorithme de population	133

2.8	Résultats	133
2.9	Conclusion	134

Chapitre 3 Ordonnancement des véhicules dans une chaîne de production
--

3.1	Description du problème de séquençement des véhicules dans une chaîne de fabrication	137
3.1.1	Processus d'ordonnancement	137
3.1.2	Les contraintes du problème	138
3.1.3	Le problème à résoudre	139
3.1.4	Comptabilisation des violations de ratio	140
3.2	Les instances	141
3.3	Algorithmes génétiques	142
3.3.1	Méthode pas-à-pas	142
3.3.2	L'algorithme génétique avec une fonction d'évaluation pondérée	146
3.4	Résultats numériques	146
3.5	Conclusion	148

Discussion	151
-------------------	------------

Conclusion générale	155
----------------------------	------------

Bibliographie	161
----------------------	------------

Table des figures

1.1	L'organigramme de l'algorithme génétique.	6
2.1	Les boîtes à moustaches de RMSE en fonction des variations des opérateurs de croisement -Berlin52.	57
2.2	Les boîtes à moustaches de RMSE en fonction des variations des opérateurs de croisement- eil101.	58
2.3	Les boîtes à moustaches de RMSE en fonction des variations des opérateurs de croisement- kroA200.	58
2.4	Les boîtes à moustaches de RMSE en fonction des variations des opérateurs de mutations pour les instances Berlin52 (en haut), eil101 (au milieu) et kroA200 (en bas).	60
2.5	La variation du RMSE en fonction du changement de la taille de la population pour les instances berlin52, eil101 et kroA200.	63
3.1	Comparaison entre les méthodes de sélection en utilisant RMSE -Berlin52. . . .	69
3.2	Comparaison entre les méthodes de sélection en utilisant RMSE -Eil101.	69
4.1	L'organigramme de notre algorithme génétique.	72
4.2	Les boîtes à moustaches des valeurs de la fonction objectif en fonction de six méthodes.	76
4.3	Comparaison des six méthodes en fonction des valeurs de la fonction objectif pour eil101.	78
4.4	Écart type des valeurs de la fonction objectif pour les six méthodes pour eil101.	78
4.5	Nombre d'itérations des six méthodes pour eil 101.	79
4.6	Nombre d'itérations sans la méthode 3 pour eil101.	79
4.7	Temps d'exécution des six méthodes pour eil101.	80
4.8	Comparaison des six méthodes en fonction des valeurs de la fonction objectif pour Kroa200.	80
4.9	Écart type des valeurs de la fonction objectif du Kroa200 pour les six méthodes.	81
4.10	Comparaison des six méthodes en fonction de nombre d'itérations pour Kroa200.	82
4.11	Temps d'exécution des six méthodes pour Kroa200.	82
4.12	Comparaison des six méthodes en fonction des valeurs de la fonction objectif de l'instance a280.	83
4.13	Écart type des valeurs de la fonction objectif de a280 pour les six méthodes.	83
4.14	Comparaison des six méthodes en fonction de nombre d'itérations pour a280.	84

4.15	Temps d'exécution des six méthodes pour a280.	84
5.1	La boucle principale de notre algorithme génétique lorsque la valeur de la fonction objectif du meilleur individu varie et ne reste pas fixe plus que 10 itérations.	91
5.2	La boucle de notre algorithme génétique lorsque la valeur de la fonction objectif est restée constante pendant 10 itérations.	92
2.1	Opérateur de croisement.	125
2.2	Opérateur de mutation.	125
2.3	L'organigramme de notre algorithme génétique pour la résolution du ALP avec une seule piste.	126
3.1	Succession des procédures	143
3.2	Algorithme génétique pour minimiser le critère F «Gen_Critère».	144

tel-00126292, version 2 - 8 Mar 2007

Introduction générale

Les lois qui régissent l'évolution des espèces ou des organismes sont connues depuis les travaux du naturaliste britannique Charles Darwin au siècle dernier. Nous savons ainsi que la nature fait appel à plusieurs mécanismes qui ont conduit, au fil du temps, à l'apparition des espèces nouvelles, toujours mieux adaptées à leurs milieux respectifs.

Darwin constata que l'évolution des espèces est basée sur deux composantes : la sélection et la reproduction. La sélection garantit une reproduction plus fréquente des chromosomes des êtres vivants les plus robustes, tandis que la reproduction est une phase durant laquelle l'évolution se réalisera et les descendants obtenus ne sont pas reproduits à l'identique.

Les algorithmes génétiques font partie de la famille des algorithmes dits "évolutifs". Ces algorithmes ont pour particularité de s'inspirer de mécanismes de l'évolution naturelle. À ces notions d'évolution, on associe les propriétés observées en génétique (codage, sélection, croisement et mutation) d'où le nom d'algorithmes génétiques.

Si l'ensemble de la théorie de l'évolution des organismes employée est dû à Darwin ([36]), c'est à John Henry Holland et son équipe que nous devons les algorithmes génétiques. Dans les années soixante, Holland chercha une manière permettant aux ordinateurs d'imiter le fonctionnement des êtres vivants pour résoudre un problème donné. Il arrive enfin à initialiser et à développer l'Algorithme Génétique Canonique (**AGC**) pour résoudre des problèmes d'optimisation ([59]).

Les algorithmes génétiques (**AG**) sont des algorithmes d'optimisation stochastiques, appartenant à la famille des algorithmes évolutionnaires, fondés sur les mécanismes de la sélection naturelle et de la génétique. Leur fonctionnement est extrêmement simple. On part d'une population de solutions potentielles (*chromosomes*) initiales arbitrairement choisies. On évalue leur performance relative (*fitness*) qui permet de quantifier sa qualité. Sur la base de ces performances on crée une nouvelle population de solutions potentielles en utilisant des opérateurs évolutionnaires simples : la sélection, le croisement et la mutation. On recommence ce cycle jusqu'à l'obtention d'une solution satisfaisante.

C'est aux travaux de Goldberg ([53]) que nous devons leur popularisation. Leurs champs d'application sont très vastes. Outre l'économie, ils sont utilisés pour l'optimisation de fonctions par Jong ([64]) en 1980, en finance par Pereira ([92]) en 2000, en théorie du

contrôle optimal par Krishnakumar et Goldberg en 1992 ([68]), Michalewicz, Janikow et Krawczyk ([75]) et Marco et al. ([73]), ou encore en théorie des jeux répétés par Axelrod ([7]) en 1987 et différentiels par Özyildirim ([87], [88]) et par Özyildirim et Alemdar ([88]).

Cette thèse est composée de trois parties.

Le premier chapitre de la première partie est consacré à une représentation des grandes structures d'algorithmes génétiques ainsi que leurs mécanismes généraux de contrôle et les principaux types d'opérateurs génétiques utilisés. Le deuxième chapitre traite les études théoriques de l'algorithme génétique.

La deuxième partie est composée de quatre chapitres.

Dans le premier, nous présenterons le problème du voyageur de commerce (**T**ravelling **S**alesman **P**roblem) de taille V , noté T.S.P., qui consiste à trouver le circuit de longueur minimal reliant V localités différentes et passant une et une seule fois par chaque localité. Par la suite, nous décrivons l'historique du T.S.P. ainsi que certaines méthodes de résolution existantes dans la littérature.

Le deuxième chapitre est dédié à présenter un état d'art de l'application de l'algorithme génétique sur le problème du voyageur de commerce. Nous analysons l'influence des opérateurs de croisement et de mutation sur les valeurs de la fonction objectif et sur la convergence vers l'optimum.

Nous nous concentrons dans le troisième chapitre sur l'importance de la génération de la population initiale et son influence sur le comportement de l'algorithme génétique. Dans le quatrième chapitre, nous présentons notre variante de l'algorithme génétique pour la résolution de ce type de problèmes.

La troisième partie est consacrée à quelques applications des (AG). Elle est décomposée de trois chapitres.

Dans le premier, nous présentons un problème d'ordonnancement, dit Job Shop, qui se situe à mi-chemin entre le Flow Shop (les opérations de chaque produit ont le même ordre sur toutes les machines) et l'Open Shop (il n'y a pas de contraintes d'ordre sur les opérations pour chaque produit). En effet, le problème de Job Shop est un problème d'ordonnancement d'ateliers qui correspond au problème de fabrication de N produits sur M machines. Chaque produit doit respecter des contraintes de type Flow Shop et dont l'objectif est de minimiser la durée totale pour fabriquer tous les produits. La fin de ce chapitre est consacrée à la présentation de notre algorithme génétique appliqué à la résolution de ce type de problèmes.

Dans le deuxième chapitre, nous nous intéressons au problème d'atterrissage des avions sur une piste d'aéroport. Après une introduction qui modélise ce problème concret, nous rappelons quelques propriétés existantes dans la littérature et puis nous présentons notre variante appliquée à ce problème.

Le troisième chapitre est accordé à la résolution d'un problème concret qui a fait l'objet du challenge de ROADEF 2005 [29]. Il s'agit d'ordonner les véhicules à l'intérieur de la même journée en satisfaisant au mieux les besoins respectifs des ateliers de la ligne de production : ateliers de peinture et de montage. La séquence de véhicules d'aujourd'hui ne doit pas remettre en question l'ordre des véhicules restants de la journée d'hier. Autrement dit, nous nous intéressons principalement à ordonner les véhicules d'aujourd'hui en se basant sur la séquence des véhicules réalisés hier qui reste figée. Dans un atelier de peinture, il est nécessaire de minimiser le nombre de purge des pistolets de peinture sachant que chacune de ces opérations de purge s'effectue automatiquement à chaque changement de couleur de véhicule ou obligatoirement après un nombre fixé de véhicules de même couleur. Dans un atelier de montage, on cherche à écarter dans la séquence le plus possible les véhicules pour lesquels les équipements nécessitent des opérations très lourdes. Il s'agit des problèmes d'ordonnement de véhicules sur une chaîne de production, avec deux objectifs contradictoires : regrouper les véhicules de même teinte et écarter ceux ayant besoin d'opérations lourdes. Ce problème ainsi que l'algorithme génétique adapté à sa résolution seront présentés dans ce chapitre.

Nous terminons ce document, par une conclusion générale qui résume les avantages, les inconvénients, les difficultés numériques des algorithmes génétiques ainsi que nos apports personnels pour les améliorer et adapter des variantes efficaces pour résoudre chaque problème.

Première partie

Algorithme génétique (AG)

Chapitre 1

Algorithme Génétique

Sommaire

1.1	Introduction	3
1.2	Nomenclature de l'algorithme génétique (AG)	3
1.3	Les avantages de l'AG	4
1.4	Principes de base des AG	4
1.5	Fonctionnement des AG	5
1.5.1	Codage du chromosome	5
1.5.2	Génération de la population initiale	6
1.5.3	Méthodes de sélection	6
1.5.4	Opérateurs de croisement	8
1.5.5	Opérateurs de mutation	9
1.5.6	Méthode d'insertion	9
1.5.7	Test d'arrêt	10
1.6	Conclusion	10

1.1 Introduction

Les algorithmes génétiques font partie de la famille des algorithmes *évolutifs*. Ils s'inspirent du credo de la nature "la survie est pour l'individu le mieux adapté à l'environnement". Ces algorithmes s'inspirent de l'évolution naturelle des espèces. Ils ont suscité l'intérêt de nombreux chercheurs. Citons d'abord Holland [59], qui a développé les principes fondamentaux, puis Goldberg [53] qui les a utilisés pour résoudre des problèmes concrets d'optimisation. D'autres chercheurs ont suivi cette voie notamment Davis [39], Mahfoud ([71] et [72]), Michalewicz ([77] et [78]), Deb ([40]), etc.

1.2 Nomenclature de l'algorithme génétique (AG)

Comme les algorithmes génétiques ont leurs racines à la fois dans la biologie et l'informatique, la terminologie utilisée est empreintée aux deux domaines (TAB. 1.1). Nous

allons passer en revue le lien entre les termes utilisés et leurs équivalents naturels pour ainsi nous aligner sur la littérature des algorithmes génétiques, en plein développement, et aussi pour nous permettre dans la suite de définir quelques analogies terminologiques.

Terme	Algorithme génétique	Signification biologique
Gène	trait, caractéristique	une unité d'information génétique transmise par un individu à sa descendance
Locus	position dans la chaîne	l'emplacement d'un gène dans son chromosome
Allèle	valeur de caractéristique	une des différentes formes que peut prendre un gène, les allèles occupent le même locus
Chromosome	chaîne	une structure contenant les gènes
Génotype	structure	l'ensemble des allèles d'un individu portés par l'ADN d'une cellule vivante
Phénotype	ensemble de paramètres ou une structure décodé	aspect physique et physiologique observable de l'individu obtenu à partir de son génotype
Épistasie	non-linéarité	terme utilisé pour définir les relations entre deux gènes "distincts", lorsque la présence d'un gène empêche la présence d'un autre gène non-allèle.

TAB. 1.1: Résumé de la terminologie utilisée en **AG**

1.3 Les avantages de l'AG

Par rapport aux algorithmes classiques d'optimisation, l'algorithme génétique présente plusieurs points forts comme :

- Le fait d'utiliser seulement l'évaluation de la fonction objectif sans se soucier de sa nature. En effet, nous n'avons besoin d'aucune propriété particulière sur la fonction à optimiser (continuité, dérivabilité, convexité, etc.), ce qui lui donne plus de souplesse et un large domaine d'applications ;
- Génération d'une forme de parallélisme en travaillant sur plusieurs points en même temps (population de taille N) au lieu d'un seul itéré dans les algorithmes classiques ;
- L'utilisation des règles de transition probabilistes (probabilités de croisement et de mutation), contrairement aux algorithmes déterministes où la transition entre deux itérations successives est imposée par la structure et la nature de l'algorithme. Cette utilisation permet dans certaines situations aux algorithmes génétiques d'éviter des optimums locaux et de se diriger vers un optimum global.

1.4 Principes de base des AG

Indépendamment de la problématique traitée, les algorithmes génétiques sont basés sur six principes :

1. Choisir le codage des solutions ;
2. Générer une population initiale de taille fixe N , formée d'un ensemble fini de solutions, dite *génération initiale* ;
3. Définir une fonction d'évaluation (fitness) permettant d'évaluer une solution et la comparer aux autres ;
4. Choisir les solutions par un mécanisme de sélection qui choisit pour un éventuel couplage ;
5. Générer de nouvelles solutions à l'aide des opérateurs génétiques en utilisant :
 - ★ Opérateur de croisement : il manipule la structure des chromosomes des parents afin de produire des individus meilleurs ou différents. Cet opérateur est effectué selon une probabilité P_x .
 - ★ Opérateur de mutation : il évite d'établir des populations uniformes incapables d'évoluer. Il consiste à modifier les valeurs des gènes de chromosomes selon une probabilité de mutation P_m .
6. Établir un compromis entre les solutions produites (progénitures) et les solutions productrices (les parents) en utilisant un mécanisme d'insertion. En d'autres termes, et suite à des informations précises, décider ce qui doit rester et ce qui doit disparaître. Tout ceci, en sauvegardant à chaque génération une taille de la population N fixe.

Ces six principes seront explicités en détail dans les paragraphes (1.5.1-1.5.6).

1.5 Fonctionnement des AG

L'algorithme génétique, présenté dans la figure (FIG.1.1), débute par une génération d'une population initiale de N individus, pour lesquels, nous calculons les valeurs de leur fonction objectif et nous sélectionnons les individus par une méthode de sélection. Les individus, sujets de croisement par l'opérateur de croisement, sont choisis selon une probabilité P_x . Leurs résultats peuvent être mutés par un opérateur de mutation avec une probabilité de mutation P_m . Les individus issus de ces opérateurs génétiques seront insérés par une méthode d'insertion dans la nouvelle population dont nous évaluons la valeur de la fonction objective de chacun de ses individus. Un test d'arrêt sera effectué pour vérifier la qualité des individus obtenus. Si ce test est vérifié alors l'algorithme s'arrête avec une solution optimale, sinon on réitère le processus pour la nouvelle génération.

1.5.1 Codage du chromosome

Le choix du codage des données dépend de la spécificité du problème traité. Il conditionne fortement l'efficacité de l'algorithme génétique. Un chromosome (une solution particulière) a différentes manières d'être codé selon l'alphabet utilisé. Nous distinguons trois types de codages :

- Numérique si l'alphabet est constitué de chiffres ;
- Symbolique si l'alphabet est un ensemble de lettres alphabétiques ou des symboles ;
- Alpha-numérique si nous utilisons un alphabet combinant les lettres et les chiffres.

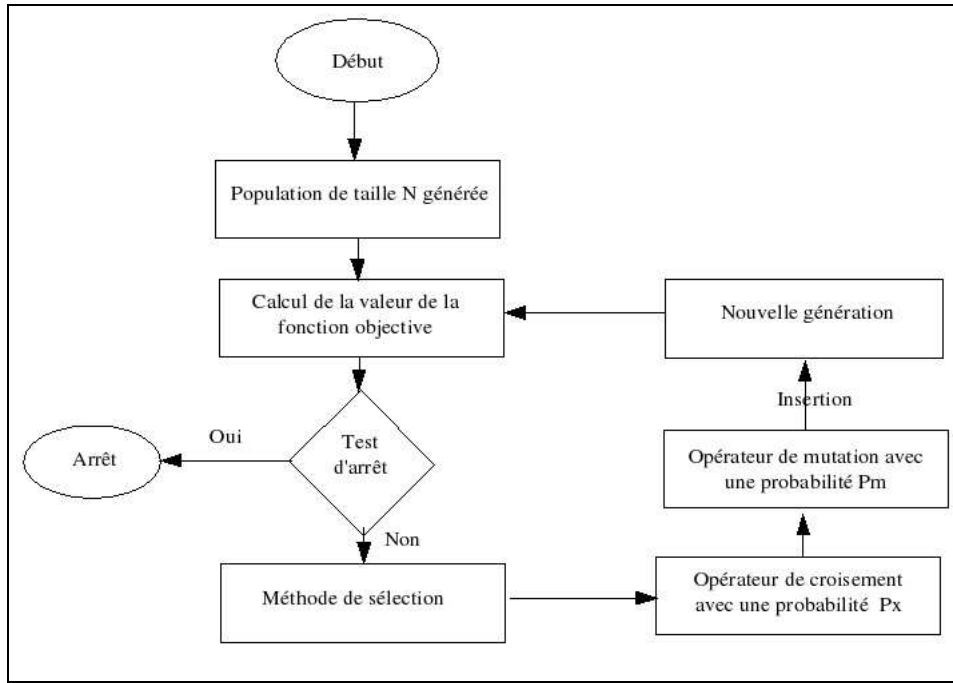


FIG. 1.1: L'organigramme de l'algorithme génétique.

1.5.2 Génération de la population initiale

Dans les problèmes d'optimisation, une connaissance de "candidats de bonne qualité" comme points d'initialisation conditionne la rapidité de la convergence vers l'optimum. Si la position de l'optimum dans l'ensemble des solutions réalisables est totalement inconnue, il est naturel de générer aléatoirement des individus. Les tirages sont réalisés en respectant les contraintes et d'une manière uniforme dans chacun des domaines associés aux composantes de l'ensemble de solutions. Des connaissances à priori sur le problème traité permettent de générer les individus dans un domaine particulier afin d'accélérer la convergence de l'algorithme génétique. Les opérateurs de croisement et de mutation permettent d'entretenir la diversité d'une population non homogène au cours des générations afin de parcourir l'ensemble de solutions le plus largement possible.

1.5.3 Méthodes de sélection

La sélection permet d'identifier les individus susceptibles d'être croisés dans une population. Nous trouvons dans la littérature plusieurs principes de sélection :

Sélection par rang : Il consiste à attribuer à chaque individu son classement par ordre d'adaptation.

Pour un problème de maximisation, nous classons les individus selon l'ordre croissant des valeurs de la fonction objectif. Ainsi, le plus mauvais individu (c'est-à-dire celui qui possède la plus petite valeur de la fonction objectif) prendra le numéro 1 et ainsi de suite (*voir* TAB.1.2). Pour un problème de minimisation, nous ordonnons les individus selon l'ordre décroissant. On prélève ensuite une nouvelle population

à partir de cet ensemble d'individus ordonnés, en utilisant des probabilités indexées sur les rangs des individus :

$$\text{Probabilité de sélection}(Parent_i) = \frac{\text{Rang}(Parent_i)}{\sum_{j \in \text{population}} \text{Rang}(Parent_j)}.$$

Cette procédure est très simple et exagère le rôle du meilleur élément au détriment d'autres éléments potentiellement exploitables. Le second, par exemple, aura une probabilité d'être sélectionné plus faible que le premier, bien qu'il soit peut-être situé dans une région d'intérêt.

	Chromosome	Fitness	Rang	Prob. de sélection = $\frac{\text{Rang}}{\text{Total}(\text{Rang})}$
	Parent1	30	2	33.33%
	Parent2	60	3	50%
	Parent3	10	1	16.67%
Total		100	6	100%

TAB. 1.2: Sélection par rang pour un problème de maximisation.

Sélection par la roulette : Dans un problème d'optimisation de maximisation, on associe à chaque individu i une probabilité de sélection, noté $Prob_i$, proportionnelle à sa valeur F_i de la fonction objectif :

$$Prob_i = \frac{F_i}{\sum_{j \in \text{population}} (F_j)}.$$

Chaque individu est alors reproduit avec la probabilité $Prob_i$. Certains individus (les "bons") seront alors "plus" reproduits et d'autres (les "mauvais") éliminés (*voir* TAB. 1.3).

	Chromosome	Fitness	Prob. de sélection = $\frac{\text{Fitness}}{\text{Total}(\text{Fitness})}$
	Parent1	30	30%
	Parent2	60	60%
	Parent3	10	10%
Total		100	100%

TAB. 1.3: Sélection par la roulette pour un problème de maximisation.

Pour un problème de minimisation, on utilise une probabilité de sélection pour un individu i égale à : $\frac{(1-Prob_i)}{(N-1)}$.

Sélection aléatoire : La sélection se fait aléatoirement, uniformément et sans intervention de la valeur d'adaptation. Chaque individu a donc une probabilité uniforme ($1/N$) d'être sélectionné. En général, la convergence de l'algorithme génétique est lente en utilisant cette méthode.

Sélection par Tournoi : Cette méthode de sélection augmente les chances des individus de mauvaise qualité par rapport à leur fitness, de participer à l'amélioration de la population. En effet, c'est une compétition entre les individus d'une sous-population de taille M ($M \leq N$) prise au hasard dans la population. Le paramètre M est fixé a priori par l'utilisateur. L'individu de meilleure qualité par rapport à la sous-population sera considéré comme vainqueur et sera sélectionné pour l'application de l'opérateur de croisement. Le paramètre M joue un rôle important dans la méthode du tournoi.

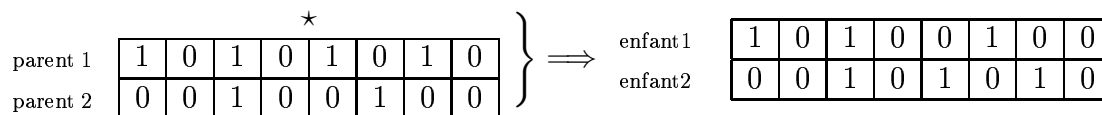
Dans le cas où $M = N$ avec N est la taille de la population. Le résultat par la sélection de la méthode du tournoi donne à chaque fois un seul individu qui est le meilleur individu par rapport à la valeur de la fonction objective. ce qui réduit l'algorithme génétique a un algorithme de recherche local travaillant sur une seule solution à la fois. Ce type d'algorithmes a pour inconvénient de converger parfois rapidement vers un optimum local.

Dans le cas $M = 1$, la méthode de sélection du tournoi correspond à la sélection aléatoire.

1.5.4 Opérateurs de croisement

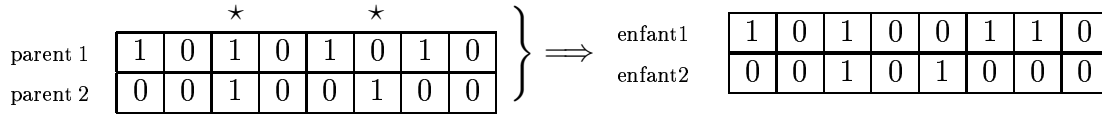
Le croisement a pour but d'enrichir la diversité de la population en manipulant les composantes des chromosomes. Classiquement, les croisements sont envisagés avec deux parents et génèrent deux enfants. Il est appliqué avec une probabilité P_x , communément appelée probabilité de croisement. Après l'utilisation de la méthode de sélection pour le choix de deux individus, nous générons un nombre aléatoire $\alpha \in [0, 1]$.

Si ($\alpha \leq P_x$), nous appliquons l'opérateur de croisement sur le couple. Les plus anciens opérateurs de croisement utilisés sont l'opérateur de croisement à un point et à deux points sur deux chromosomes à codage binaire. Ils constituent la base des opérateurs de croisement. L'opérateur à un point de croisement consiste à diviser chacun des deux parents en deux parties à la même position, choisie au hasard. L'enfant1 est composé de la première partie du premier parent et de la deuxième partie du deuxième parent alors que l'enfant 2 est constitué de la première partie du deuxième parent et de la deuxième partie du premier parent (TAB.1.4).



TAB. 1.4: L'opérateur de croisement à 1 point.

L'opérateur à deux points de croisement est illustré dans le tableau (TAB.1.5). Il consiste à fixer deux positions. L'enfant1 sera la copie du parent 1 en remplaçant sa partie entre les deux positions par celle du parent2. On effectuera la même opération pour déterminer l'enfant2 en intervertissant les rôles des parent1 et parent2.



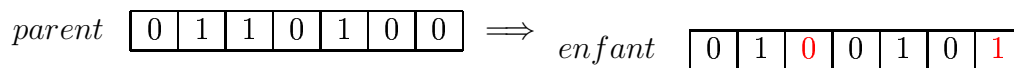
TAB. 1.5: L'opérateur de croisement à 2 points.

Dans les chapitres suivants, nous présentons d'autres opérateurs de croisement plus adaptés à la nature de chaque problème à résoudre.

1.5.5 Opérateurs de mutation

L'opérateur de mutation apporte aux algorithmes génétiques l'aléa nécessaire à une exploration efficace de l'espace. Cet opérateur nous garantit que l'algorithme génétique sera susceptible d'atteindre la plupart des points du domaine réalisable. Cerf en 1994 ([27]) a démontré théoriquement que l'algorithme génétique converge en probabilité en utilisant l'opérateur de mutation et sans croisement. Les propriétés de convergence des algorithmes génétiques sont donc fortement dépendantes de cet opérateur.

Cet opérateur de mutation est utilisé avec une probabilité (P_m) nommée probabilité de mutation. Dans les algorithmes génétiques à codage binaire, cette probabilité s'effectue sur les gènes en échangeant sa valeur de 0 à 1 ou de 1 à 0 et non sur le chromosome tout entier.



TAB. 1.6: Opérateur de mutation d'un bit

Mais avec un algorithme génétique codé autrement, on applique cette probabilité par rapport à l'individu tout entier et non sur les gènes. Si β , généré aléatoirement, appartient à $[0, P_m]$, nous appliquons l'opérateur de mutation sur cet individu.

1.5.6 Méthode d'insertion

Après l'étape de mutation, on utilise une méthode d'insertion pour générer une nouvelle population. Lors de la construction de cette population, on se trouve devant un vrai problème : faut-il garder les enfants ou les parents ou bien un certain pourcentage des deux en respectant que la taille de la population (N) reste constante ?

Il s'agit de concevoir une stratégie d'évolution de la population. Nous distinguons dans la littérature deux stratégies :

- La première stratégie, notée $(\mathbf{N}, \mathbf{N}_f)$, consiste à choisir les N individus à partir de N_f enfants déjà créés par les opérateurs de croisement et de mutation. Dans cette stratégie, on suppose que $N_f \geq N$. Quand $N_f = N$, nous parlerons de la méthode générationnelle qui remplace les parents par les enfants.
- La seconde, notée $(\mathbf{N} + \mathbf{N}_f)$, consiste à choisir les N individus à partir des N parents de la population précédente et de N_f nouveaux enfants. Un cas particulier de cette stratégie, appelé méthode d'état d'équilibre, a pour principe de sauvegarder une grande partie de la population dans la génération suivante. A chaque itération quelques chromosomes (parents) ayant les meilleurs coûts seront sélectionnés afin de créer des chromosomes fils qui remplaceront les plus mauvais parents. Le reste de la population survie et sera copié dans la nouvelle génération.

L'élitisme est une stratégie complémentaire de la première stratégie. Il consiste à copier quelques meilleurs chromosomes dans la nouvelle population. Il accroît l'efficacité de l'algorithme génétique basé sur la méthode d'insertion générationnelle. L'objectif est d'éviter que les meilleurs chromosomes soient perdus après les opérations de croisement et de mutation. Cette méthode améliore considérablement les algorithmes génétiques, car elle permet de conserver, à une itération k , le meilleur individu trouvé dans toutes les populations générées antérieurement.

1.5.7 Test d'arrêt

Le test d'arrêt joue un rôle primordial dans le jugement de la qualité des individus. Son but est de nous assurer l'optimalité, de la solution finale obtenue par l'algorithme génétique.

Les critères d'arrêts sont de deux natures :

1. Arrêt après un nombre fixé a priori de générations. C'est la solution retenue lorsqu'une durée maximale de temps de calcul est imposée.
2. Arrêt lorsque la population cesse d'évoluer ou n'évolue plus suffisamment. Nous sommes alors en présence d'une population homogène dont on peut penser qu'elle se situe à la proximité de l'optimum. Ce test d'arrêt reste le plus objectif et le plus utilisé.

Il est à noter qu'aucune certitude concernant la bonne convergence de l'algorithme n'est assurée. Comme dans toute procédure d'optimisation l'arrêt est arbitraire, et la solution "en temps fini" ne constitue qu'une approximation de l'optimum.

1.6 Conclusion

Dans ce chapitre, nous avons établi les fondations nécessaires à la compréhension des algorithmes génétiques. Nous avons exposé en détail les différentes étapes qui constituent la structure générale d'un algorithme génétique : Codage, méthode de sélection, opérateurs de croisement et de mutation avec leurs probabilités, méthode d'insertion et le test

d'arrêt. Pour chacune de ces étapes, il existe plusieurs possibilités. Le choix entre ces différentes possibilités nous permet de créer plusieurs variantes de l'algorithme génétique. Notre travail par la suite s'intègre dans la réponse à cette perspective pour trouver une solution à ce problème combinatoire : Quels sont les meilleurs paramètres essentiels qui créent une variante efficace pour la résolution d'un problème d'ordonnancement dont l'ensemble des solutions réalisables c'est l'ensemble de permutations.

Dans les chapitres suivants, nous évaluons la qualité de nos variantes de l'algorithme génétique appliquées aux problèmes d'ordonnancement concernant le problème du voyageur de commerce, Jobshop, le problème d'atterrissage des avions et le problème d'ordonnancement des véhicules dans une chaîne de production d'une usine (challenge ROADEF 2005 [29]) par rapport à la détermination des meilleures valeurs des paramètres, cités ci-dessus, et appliqués à chaque type de problèmes.

tel-00126292, version 2 - 8 Mar 2007

Chapitre 2

L'analyse théorique des algorithmes génétiques

Sommaire

2.1	Introduction	13
2.2	Théorie du schéma	14
2.2.1	Définitions fondamentales	14
2.2.2	Effets de la reproduction	15
2.2.3	Effets des croisements	16
2.2.4	Effets des mutations	16
2.3	Modélisation par chaîne de Markov	17
2.3.1	Modélisation de l'algorithme génétique	17
2.3.2	Description rapide de l'algorithme	17
2.3.3	Modélisation	18
2.3.4	Processus de fond (X_k^∞)	20
2.3.5	Processus perturbé (X_k^l)	20
2.3.6	La théorie de Freidlin et Wentzell	23
2.3.7	Résultats de convergence	26
2.4	Conclusion	28

2.1 Introduction

Dans le chapitre précédent, nous avons présenté les différentes composantes de l'algorithme génétique que ce soit le codage, la sélection, les opérateurs génétiques de croisement et de mutation et les méthodes d'insertion. Maintenant, nous nous intéresserons aux études théoriques effectuées sur certaines variantes d'algorithme génétique. L'objectif escompté est de prouver la convergence de celles-ci.

Nous présenterons, tout d'abord, la théorie des schémas qui se focalisent principalement sur l'estimation de la survie d'un type de chromosome d'une génération à la génération suivante en utilisant l'algorithme génétique binaire avec un opérateur de croisement avec

un point et l'opérateur de mutation sur les bits et la méthode de sélection par la roulette. Puis, nous nous intéresserons à la modélisation de l'algorithme génétique avec les chaînes du Markov. Notamment, au travaux de Cerf [27] qui a montré la convergence en probabilité de l'algorithme génétique.

2.2 Théorie du schéma

Historiquement, les algorithmes génétiques binaires sont les plus anciens et ont donc été les plus étudiés sur le plan théorique en prenant en compte l'évolution des schémas. Holland était le premier à proposer la première version de la théorie des schémas, largement développée, ensuite par Goldberg [52]. Nous traitons le cas d'un problème d'optimisation dont la fonction objectif est à maximiser.

2.2.1 Définitions fondamentales

Avant d'entrer dans le vif du sujet, nous présentons quelques définitions de base :

Définition 2.2.1 (Séquence). On appelle séquence A de longueur $l(A)$ une suite $A = a_1a_2\dots a_i\dots a_l$ avec $i \in \{1, \dots, l\}$, $a_i \in V = \{0, 1\}$. En codage binaire, les chromosomes sont des séquences.

Définition 2.2.2 (Schéma). On appelle schéma H de longueur l une suite $H = a_1a_2\dots a_l$ avec $\forall i \in \{1, \dots, l\}$, $a_i \in V_+ = \{0, 1, *\}$. Le signe $(*)$ en position i signifie que a_i peut être indifféremment 0 ou 1.

Définition 2.2.3 (Instance). Une séquence $A = a_1a_2\dots a_l$ est une instance d'un schéma $H = b_1\dots b_l$ si pour tout i tel que $b_i \neq *$ on a $a_i = b_i$.

Ainsi, $H = 010 * 01$ est un schéma et les séquences 010001 et 010101 sont ses seules instances.

Définition 2.2.4 (Position fixe, position libre). Soit un schéma H . On dit que i est une position fixe de H si $a_i = 1$ ou $a_i = 0$. Par contre, si $a_i = *$, i est une position libre de H .

Définition 2.2.5 (Ordre d'un schéma). On appelle ordre du schéma H , noté $o(H)$, le nombre de positions fixes de H .

Par exemple, le schéma $H = 01 **10 * 1$ a pour ordre $o(H) = 5$, le schéma $H' = ****101$ a pour ordre $o(H') = 3$. On note qu'un schéma H de longueur $l(H)$ et d'ordre $o(H)$ admet $2^{(l(H)-o(H))}$ instances différentes.

Définition 2.2.6 (Longueur fondamentale). On appelle longueur fondamentale du schéma H la distance séparant la première position fixe de H de la dernière position fixe de H . On note cette longueur fondamentale $d(H)$.

Ainsi, le schéma $H = 1 * * 01 * * *$ a pour longueur fondamentale $d(H) = 5 - 1 = 4$, le schéma $H' = 1 * * * * * 1$ a $d(H') = 8 - 1 = 7$, et pour le schéma $H'' = * * 1 * * * * *$ nous avons $d(H'') = 3 - 3 = 0$.

Définition 2.2.7 (Adaptation d'une séquence). On appelle adaptation d'une séquence A une valeur positive que nous noterons $f(A)$. f est la fonction objectif ou fitness du problème à résoudre.

Définition 2.2.8 (Adaptation d'un schéma). On appelle adaptation d'un schéma H la valeur

$$f(H) = \frac{\sum_{i=1}^{2^{(l(H)-o(H))}} f(A_i)}{2^{(l(H)-o(H))}}$$

où les A_i décrivent l'ensemble des instances de H . Ainsi, une adaptation d'un schéma H c'est la moyenne des adaptations de ses instances.

2.2.2 Effets de la reproduction

Soit un ensemble $S = \{A_1, \dots, A_i, \dots, A_n\}$ de n séquences de bits tirées aléatoirement. Durant la reproduction, chaque séquence A_i est reproduite avec une probabilité :

$$p_i = \frac{f(A_i)}{\sum_{i=1}^n f(A_i)}$$

Supposons qu'il y ait à l'instant t un nombre $m(H, t)$ de séquences représentant le schéma H dans la population S . À l'instant $(t + 1)$, statistiquement, ce nombre vaut :

$$m(H, t + 1) = m(H, t) \cdot n \cdot \frac{f(H)}{\sum_{i=1}^n f(A_i)}$$

Posons

$$\bar{f}_t = \frac{\sum_{i=1}^n f(A_i)}{n}$$

\bar{f}_t représente la moyenne de l'adaptation des séquences à l'instant t . La formule précédente devient :

$$m(H, t + 1) = m(H, t) \frac{f(H)}{\bar{f}_t}$$

Posons $c_t(H) = \frac{f(H)}{\bar{f}_t} - 1$. On obtient alors $m(H, t + 1) = (1 + c_t(H))m(H, t)$ Il est donc clair qu'un schéma, dont l'adaptation est au-dessus de la moyenne, voit son nombre de représentants augmenter, suivant une progression qui est de type géométrique si nous faisons l'approximation que $c_t(H)$ est constant dans le temps. Alors : $m(H, t) = (1 + c(H))^t \cdot m(H, 0)$ Si seulement la reproduction était en jeu, les schémas forts élimineraient très rapidement les schémas faibles.

2.2.3 Effets des croisements

Nous nous intéressons dans ce paragraphe à la probabilité de survie $p_s(H)$ d'un schéma H lors d'une opération de croisement par l'opérateur de croisement en un point. Par exemple, supposons qu'une instance d'un schéma $H = **10*1**$ soit croisée avec une autre instance. Quelle est la probabilité pour que la séquence résultante soit encore une instance de H ? Il est impossible de répondre exactement à la question, tout en plus peut-on donner une borne inférieure de cette valeur. Il est clair que H ne sera pas détruit si le site de croisement qui est tiré au sort est inférieur à 3 (avant le premier 1) ou s'il est supérieur à 6 (après le dernier 1).

On voit donc immédiatement qu'une borne inférieure de la probabilité de détruire un schéma H est $\frac{d(H)}{(l-1)}$. Donc la probabilité de survie dans un croisement est $1 - \frac{d(H)}{(l-1)}$. D'autre part, on croise avec une probabilité p_x . Donc, la probabilité de survie du schéma H est donnée par :

$$p_s \simeq 1 - p_x \frac{d(H)}{l-1}$$

De ce résultat et des résultats précédents découle une loi d'évolution d'une population :

$$m(H, t+1) \simeq m(H, t)(1 + c_t(H))(1 - p_x \frac{d(H)}{l-1})$$

2.2.4 Effets des mutations

Soit p_m la probabilité de mutation d'un bit dans une séquence. Dans un schéma H , seules les positions fixes peuvent être détruites. La probabilité de survie d'un bit étant $1 - p_m$, la probabilité de survie d'un schéma H contenant $o(H)$ positions fixes est $(1 - p_m)^{o(H)}$. La probabilité de mutation étant supposée petite devant 1, un développement limité au premier ordre donne une probabilité de survie égale à $1 - o(H)p_m$.

L'équation finale s'écrit donc :

$$m(H, t+1) \simeq m(H, t)(1 + c_t(H))(1 - p_x \frac{d(H)}{l-1} - o(H)p_m).$$

Il est à noter que dans cette modélisation, la possibilité de créer par les opérateurs de croisement et mutation le schéma H a été négligée.

Des calculs précédents découlent deux résultats :

- les schémas, dont les longueurs fondamentales sont petites, sont plus favorisés que les autres, lors de la génération d'une nouvelle population.
- les schémas, dont les ordres sont petits, sont plus favorisés que les autres, lors de la génération d'une nouvelle population.

Ces résultats conditionnent la qualité du codage des données. En effet, les schémas qui codent les données *intéressantes* pour le problème considéré doivent avoir un ordre et une

longueur fondamentale faibles, alors que les données "sans intérêt" doivent être codées par des schémas qui ont un ordre et une longueur fondamentale élevés. Cette modélisation ne permet pas d'obtenir des résultats théoriques solides par rapport à la convergence de l'AG et sa complexité.

2.3 Modélisation par chaîne de Markov

Cette dernière approche est la plus satisfaisante tant sur le plan mathématique, que sur celui de la modélisation. Les différents opérateurs étant présentés comme perturbant un processus Markovien représentant la population à chaque étape. Nous présentons cette théorie et les principaux résultats de convergence.

2.3.1 Modélisation de l'algorithme génétique

Les principaux résultats asymptotiques portant directement sur les algorithmes génétiques, ont été développés par Cerf [27] sur la base des travaux de Catoni [26] et de Trounev [102]. Ces travaux sont fondés sur la théorie des petites perturbations aléatoires d'un processus dynamique de type Markovien. Plus particulièrement, la théorie de Freidlin et Wentzell [47] constitue la pierre d'angle de ces études. Nous donnons ici, quelques résultats particulièrement révélateurs de la dynamique des algorithmes génétiques, développés par Cerf.

Afin de préciser le cadre de cette section, nous travaillerons ici sur la base d'un codage binaire. P représente le nombre de bits utilisés pour le codage (longueur du chromosome). La fonction d'évaluation, f sera donc définie sur l'espace $E = \{0, 1\}^P$ à valeurs dans \mathbf{R}^+ . Le problème consiste à localiser l'ensemble des maxima globaux de f , ou, à défaut, de trouver rapidement et efficacement des régions de l'espace, où se situent ces maxima.

2.3.2 Description rapide de l'algorithme

Comme nous l'avons vu l'algorithme génétique est un algorithme stochastique itératif qui opère sur des ensembles de points. La variante de l'algorithme génétique considérée est bâtie à l'aide de trois opérateurs : mutation, croisement et sélection, que nous présentons plus formellement par la suite.

Soit N la taille fixe de la population, notons X_k la population de la génération k : il s'agit d'une matrice $X_k = (X_k^1, X_k^2, \dots, X_k^N)$ de E^N dont les N éléments sont des chaînes de bits (chromosomes) de longueur P . Le passage de la génération k à la génération $k+1$, c'est-à-dire de X_k à X_{k+1} se décompose en trois étapes :

$$X_k \xrightarrow{\text{Mutation}} Y_k \xrightarrow{\text{Croisement}} Z_k \xrightarrow{\text{Sélection}} X_{k+1}.$$

Chacune de ces étapes peut être modélisée formellement.

- Mutation $X_k \xrightarrow{\text{Mutation}} Y_k$

L'opérateur de mutation considéré est le suivant : pour chaque composante de chaque élément X_k^i , une variable de Bernoulli de paramètre P_m est tirée indépendamment et suivant le résultat l'élément binaire examiné est changé ou non. (0 est changé en 1 ou bien 1 en 0).

La probabilité P_m de mutation doit être préalablement choisie et est généralement faible.

Comme nous le verrons par la suite, cet opérateur joue un rôle clé dans la convergence de l'algorithme génétique.

- Croisement $Y_k \xrightarrow{\text{Croisement}} Z_k$

L'opérateur de croisement utilisé est l'opérateur de croisement à un point avec une probabilité de croisement P_x fixée initialement. Pour construire la population Z_k , $\frac{N}{2}$ couples sont formés à partir de la population Y_k (par exemple en faisant apparaître les individus consécutifs de Y_k , ou bien en choisissant au hasard et uniformément des individus dans Y_k). Pour chaque couple, une variable de Bernoulli de paramètre P_x est tirée pour décider si le croisement aura lieu. Si c'est le cas, un site de coupure est tiré au hasard, et les segments finaux des deux chromosomes sont échangés.

Une nouvelle paire d'individus est ainsi obtenue (identique à l'ancienne s'il n'y a pas eu de croisement) et stockée dans la population Z_k . En général, le paramètre P_x est choisi grand.

Remarquons que les opérateurs de mutation et de croisement ne font pas intervenir la fonction f , ce sont des opérateurs stochastiques d'exploration. C'est le troisième et dernier opérateur, la sélection, qui guide la population vers les valeurs élevées de la fonction f .

- Sélection $Z_k \xrightarrow{\text{Selection}} X_{k+1}$

Les N individus de la population X_{k+1} sont obtenus après sélection des individus de Z_k . On conserve ainsi les "meilleurs" individus de Z_k , indépendamment à l'aide d'une distribution de probabilité qui favorise les individus de Z_k les mieux adaptés. Le choix le plus fréquent est l'unique distribution telle que la probabilité d'un individu soit proportionnelle à son adaptation, i.e. la probabilité de sélection de l'individu Z_k^i est :

$$P_i = P(Z_k^i) = \frac{f(Z_k^i)}{\sum_{j=1}^n f(Z_k^j)}$$

En tirant les individus dans la population Z_k conformément aux probabilités P_i , on constitue la nouvelle génération X_{k+1} .

2.3.3 Modélisation

La présentation rapide des opérateurs nous permet de modéliser la suite des $(X_k)_{k \in \mathbb{N}}$ en une chaîne de Markov, d'espace d'états $E = (\{0, 1\}^P)^N$. L'algorithme génétique ne doit donc pas être interprété comme une procédure d'optimisation mais plutôt comme une marche aléatoire dans l'espace d'états, attirée vers les fortes valeurs de f .

La propriété première de cette formalisation est que la loi de X_k est déterminée de manière unique par :

- ★ la loi de la génération initiale X_0 ;
- ★ le mécanisme de transition de X_k à X_{k+1} , mécanisme scindé en trois étapes détaillées précédemment.

Ce mécanisme de transition possède toutefois des propriétés essentielles qui font l'intérêt et la puissance de cette formalisation, ([27]) :

- Il est homogène (c'est-à-dire indépendant de la génération k considérée).
- Il est irréductible, C'est-à-dire la probabilité de joindre deux points quelconques de l'espace d'états, en un nombre fini de générations est non nulle :

$$(\forall x, y \in E) (\exists r \in N) P[X_{k+r} = y | X_k = x] > 0.$$

Le mécanisme permet donc d'explorer tout point de l'espace d'états, avec une probabilité non nulle.

- Il est apériodique (cette hypothèse n'est cependant pas fondamentale).

Ces propriétés permettent de conclure à l'ergodicité de la chaîne de Markov, et à l'existence d'un processus limite.

Théorème 2.3.1. *Une chaîne de Markov homogène irréductible apériodique d'espace d'états fini est ergodique et possède une unique mesure de probabilité stationnaire ou invariante.*

Cette mesure stationnaire correspond à la loi régissant l'équilibre du processus, elle est définie, pour tout y , comme :

$$\mu(y) = \lim_{k \rightarrow +\infty} P[X_k = y | X_0 = x]$$

Nous savons également que tout élément de l'espace d'états est de probabilité non nulle pour cette mesure.

Toutefois, si ce résultat nous permet de savoir qu'il existe une dynamique de fond de l'algorithme génétique, il nous reste à en déterminer les propriétés, l'influence des opérateurs (et des paramètres associés) qui jouent un grand rôle dans le processus. Pour cela, Cerf a introduit les notations suivantes :

- Si $x = (x_1, \dots, x_N)$ est un élément de E^N et i un point de E , nous noterons :

$$\bar{f}(x) = \bar{f}(x_1, \dots, x_N) = \max\{f(x_i) : 1 \leq i \leq N\}$$

$$\bar{x} = \{x_k \in \operatorname{argmax} f(x)\}$$

$$[x] = \{x_k : 1 \leq k \leq N\}$$

De manière générale, les lettres z, y, z, u, v, \dots désignent des populations, i.e. des éléments de E^N , et les lettres i, j des points de E .

2.3.4 Processus de fond (X_k^∞)

C'est à partir de ce processus de fond qu'est reconstitué l'algorithme génétique en étudiant ses perturbations aléatoires par les différents opérateurs. Il est défini comme processus limite, lorsque les perturbations ont disparu. C'est également une chaîne de Markov sur E^N dont le mécanisme de transition est très simple puisque correspondant à la situation limite suivante :

- Les N composantes de X_{k+1}^∞ sont choisies indépendamment et suivant la loi uniforme sur l'ensemble X_k^∞ ;
- Les individus dont l'adaptation n'est pas maximale en k , sont éliminés et n'apparaissent pas dans la génération $(k + 1)$;
- Les individus dont l'adaptation est maximale, ont des chances de survie égales ;
- Cette chaîne est tout d'abord piégée dans l'ensemble S des populations ayant la même adaptation (ou ensemble des population d'équiadaptation), $S = \{x = (x_1, \dots, x_N) \in E^N : f(x_1) = f(x_2) = \dots = f(x_N)\}$.

Cette population représente les attracteurs de la chaîne (voir E.2.4 plus loin), puis elle est absorbée par une population uniforme, de sorte que :

$$(\forall x \in E^N) P[(\exists x_i \in \bar{x}) (\exists K) (\forall k \geq K) X_k^\infty = x_i | X_0^\infty = x_{ini}] = 1$$

Lorsque la population est devenue uniforme et en l'absence ici de perturbations, il ne se passe plus rien.

Ceci peut également se traduire en définissant les populations uniformes comme les états absorbants de la chaîne X_k^∞ .

2.3.5 Processus perturbé (X_k^l)

La modélisation proposée par Cerf, part du processus de fond (X_k^∞), décrit ci-dessus, qui est perturbé aléatoirement. La chaîne de Markov (X_k^∞) devient donc une suite de chaînes de Markov (X_k^l), dont le mécanisme de transition est donné par la succession des transformations générées par les opérateurs.

$$X_k^l \xrightarrow{\text{Mutation}} U_k^l \xrightarrow{\text{Croisement}} V_k^l \xrightarrow{\text{Selection}} X_{k+1}^l.$$

Il nous faut pour cela modéliser précisément les opérateurs.

Mutation $X_k^l \xrightarrow{\text{Mutation}} U_k^l$

Les mutations sont définies comme des petites perturbations aléatoires indépendante des individus, de la population X_k^l . Il est assez naturel d'introduire la probabilité $p_l(i, j)$ de transition de mutation entre les points i et j de E , comme un noyau Markovien p_l . Trivialement on a :

$$\forall i \in E \quad \sum_{j \in E} p_l(i, j) = 1$$

Sur la chaîne X_k^l , la probabilité de transition entre les points x et u de E^N est :

$$P[U_k^l = u | X_k^l = x] = p_l(x_1, u_1)p_l(x_2, u_2)\dots p_l(x_N, u_N)$$

Plus précisément et afin d'analyser la dynamique de (X_k^l) lorsque l tend vers l'infini, nous reportons ici les hypothèses sur le mode et la vitesse de convergence des probabilités de transition. Pour cela nous supposons l'existence d'un noyau irréductible, α , sur E , i.e. :

$(\forall i, j \in E), \exists \{i_0, i_1, \dots, i_r\}$ (c'est-à-dire un chemin dans E avec $i_0 = i$ et $i_r = j$) tels que :

$$\prod_{0 \leq k \leq r-1} \alpha(i_k, i_{k+1}) > 0$$

L'hypothèse d'irréductibilité du noyau α est essentielle, elle assure que tout point de l'espace est potentiellement visitable.

La vitesse de convergence du noyau p_l , est caractérisée par le réel positif a , tel que p_l admette le développement suivant :

$(\forall i, j \in E) (\forall s)$

$$p_l(i, j) = \begin{cases} \alpha(i, j)l^{-a} + o(l^{-s}) & \text{si } i \neq j \\ 1 - \alpha(i, j)l^{-a} + o(l^{-s}) & \text{si } i = j \end{cases}$$

La condition de positivité de a nous permet de faire disparaître les perturbations lorsque l tend vers l'infini. $(\forall i, j \in E)$

$$\lim_{l \rightarrow \infty} p_l(i, j) = \delta(i, j) = \begin{cases} 0 & \text{si } i \neq j \\ 1 & \text{si } i = j \end{cases}$$

Croisement $U_k^l \xrightarrow{\text{Croisement}} V_k^l$

Ici l'opérateur est modélisé comme effectuant des petites perturbations aléatoires sur des couples de la population U_k^l . Ces couples sont formés par les éléments successifs de la population, les transitions sont gérées par le noyau Markovien q_l sur $E \times E$:

$$P[V_k^l = v | U_k^l = u] = q_l((u_1, u_2), (v_1, v_2))q_l((u_3, u_4), (v_3, v_4))\dots$$

Pour ce noyau q_l , nous supposons l'existence d'un noyau irréductible β sur $E \times E$, la vitesse de convergence est alors paramétrée par le réel positif b tel que :

$$\forall (i_1, j_1) \in E \times E \quad \forall (i_2, j_2) \in E \times E \quad \forall s \\ q_l((i_1, j_1), (i_2, j_2)) = \begin{cases} \beta((i_1, j_1), (i_2, j_2))l^{-b} + o(l^{-s}) & \text{si } ((i_1, j_1) \neq (i_2, j_2)) \\ 1 - \beta((i_1, j_1), (i_2, j_2))l^{-b} + o(l^{-s}) & \text{si } ((i_1, j_1) = (i_2, j_2)) \end{cases}$$

L'évanouissement asymptotique des croisements est également imposée par la positivité de b :

$$\forall (i_1, j_1) \in E \times E \quad \forall (i_2, j_2) \in E \times E \quad \lim_{l \rightarrow \infty} q_l((i_1, j_1), (i_2, j_2)) = \delta((i_1, i_2)) \cdot \delta(j_1, j_2)$$

Sélection $V_k^l \xrightarrow{\text{Selection}} X_{k+1}^l$

C'est l'opérateur le plus compliqué et également le plus important puisqu'il permet la convergence vers les optima de f .

Il est modélisé à l'aide d'une fonction de sélection F_l dont Cerf nous donne une définition précise, pouvant être résumée par :

$$\begin{aligned} F_l : \{1, \dots, N\} \times (\mathbb{R}_+^N) &\longmapsto [0, 1] \\ (i, f_1, f_2, \dots, f_N) &\longrightarrow F_l(i, f_1, f_2, \dots, f_N) \end{aligned}$$

telle que :

- i) $F(\cdot, f_1, f_2, \dots, f_N)$ est une probabilité sur $\{1, \dots, N\}$;
- ii) Cette probabilité est indépendante de l'indexation des f_1, f_2, \dots, f_N (on peut permuter les f_i) ;
- iii) La probabilité favorise les éléments i associés à des valeurs élevées.
i.e. Si $f_1 \geq f_2 \geq \dots \geq f_N$ Alors

$$F_l(1, f_1, f_2, \dots, f_N) \geq F_l(2, f_1, f_2, \dots, f_N) \geq \dots \geq F_l(N, f_1, f_2, \dots, f_N).$$

Cet outil nous permet d'écrire la probabilité de transition correspondant à la dernière étape :

$$P[X_{k+1}^l = x | V_k^l = v] = \prod_{r=1}^N Y_l(x_r, v_r)$$

Ceci signifie que la probabilité de transition est le produit des probabilités sur chacune des N composantes de E .

La probabilité Y_l entre deux composantes (x_r, v_r) est donnée par :

$Y_l(x_r, v_r) = \sum_{\{k: v_k = x_k\}} F_l(k, f(v_1), f(v_2), \dots, f(v_N))$ De même que pour les autres opérateurs, la fonction de sélection doit être choisie et sa vitesse de convergence caractérisée :

$$F_l(i, f_1, f_2, \dots, f_N) = \frac{\exp(cf_i \ln(l))}{\sum_{r=1}^N \exp(cf_r \ln(l))} \quad (E.5)$$

Ce choix correspond bien à une probabilité de sélection avantageant les fortes adaptations au détriment des faibles, le réel positif c indexant cette fonction.

Le mécanisme de sélection opérant sur le processus de fond (X_k^∞) , correspond à la fonction de sélection F^∞ définie par :

$$F^\infty(k, f(x_1), f(x_2), \dots, f(x_N)) = \frac{1_{\bar{x}}(xk)}{\text{card}(\bar{x})}$$

C'est-à-dire, la loi uniforme sur l'ensemble $x = \{x_k \in \text{argmax} f(x)\}$ est la limite de La suite $(F_l)_{l \in \mathbb{N}}$ des fonctions de sélection.

$$(\forall x \in E^N)(\forall k) \lim_{l \rightarrow \infty} F_l(k, f(x_1), f(x_2), \dots, f(x_N)) = F^\infty(k, f(x_1), f(x_2), \dots, f(x_N)) \quad (E.6)$$

Les conditions (E.2), (E.4), et (E.6) nous permettent d'assurer que le mécanisme de transition de la chaîne (X_k^l) converge vers celui du processus de fond (X_k^∞) : $\forall (y, z \in E^N)$
 $\lim_{l \rightarrow \infty} P[X_{k+1}^l = z | X_k^l = y] = P[X_{k+1}^\infty = z | X_k^\infty = y]$

C'est également en ce sens que l'on interprète la chaîne (X_k^l) comme une perturbation de la chaîne (X_k^∞) .

Les vitesses de convergence intervenant dans chacun des opérateurs jouent un rôle important. La formulation proposée en (E.1), (E.3) et (E.5), permet un ajustement équitable de ces vitesses (elles sont logarithmiquement du même ordre) de sorte qu'aucun opérateur ne domine les autres dans la dynamique. Lorsque l tend vers l'infini, les conditions (E.2), (E.4), et (E.6) nous permettent d'assurer que le mécanisme de transition de la chaîne (X_k^l) converge vers celui du processus de fond (X_k^∞) , et on a :

$$(\forall y, z \in E^N) \lim_{l \rightarrow \infty} P[X_{k+1}^l = z | X_k^l = y] = P[X_{k+1}^\infty = z | X_k^\infty = y]$$

La chaîne (X_k^l) se comporte alors comme le ferait (X_k^∞) . La théorie de Freidlin-Wentzell nous donne les outils pour simplifier l'étude de ces processus à temps continu.

2.3.6 La théorie de Freidlin et Wentzell

Soit le système différentiel de \mathbf{R}^N satisfaisant les équations déterministes :

$$dx_t = b(x_t)dt \quad \text{et} \quad x_0 = x_{ini} \quad (E.7)$$

Sous de *bonnes* hypothèses, il existe une solution (trajectoire) unique, $x(t)$ à l'équation (E.7) et à la condition initiale associée. L'une des préoccupations immédiates est de savoir si cette solution va, ou non, tendre vers un équilibre (qui n'est pas forcément unique). Et si oui, quel en est l'ensemble de stabilité. L'équilibre est défini comme une fonction constante x^* telle que $x^* = \lim_{t \rightarrow \infty} x_t$, et l'ensemble de stabilité comme l'ensemble $K(x^*)$ des points de départ qui mènent à cet équilibre. On peut élargir cette notion, d'équilibre et de stabilité, par celles, très proches, d'attracteur et de bassin d'attraction.

Un attracteur du système est le voisinage compact K_i d'un point visité une infinité de fois, et le bassin d'attraction l'ensemble des points de départ qui mènent à cet attracteur. Nous supposons que \mathbb{R}^d possède un nombre fini d'attracteurs K_1, \dots, K_r .

La théorie de Freidlin-Wentzell étudie l'évolution du système (E.7) lorsqu'il subit des perturbations Browniennes, d'intensité ϵ . Le système déterministe (E.7) devient alors un système différentiel stochastique :

$$dX_t^\epsilon = b(X_t^\epsilon)dt + \epsilon dW_t \quad \text{et} \quad X_0^\epsilon = x_{ini} \quad (E.8)$$

Le processus $(X_t^\epsilon)_{t \in \mathbb{R}_+}$ est maintenant un processus stochastique perturbé par le mouvement Brownien $(w_t)_{t \in \mathbb{R}_+}$ et dépendant de ϵ . La situation change alors puisque les perturbations browniennes permettent au processus de s'échapper de n'importe quel bassin d'attraction, et en fait le processus les visite tous.

De plus, le processus est ergodique et admet une unique mesure de probabilité invariante,

i.e. $\forall B$ Borélien de \mathbb{R}^d

$\lim_{t \rightarrow \infty} P[X_t^\epsilon \in B | X_0^\epsilon = x_{ini}] = \mu^\epsilon(B)$ existe et $\mu^\epsilon(B)$ est la probabilité de présence du processus dans le Borélien B , lorsque le système a atteint son état d'équilibre. Cette probabilité μ^ϵ est invariante avec le point de départ x_{ini} .

Lorsque les perturbations cessent, le processus se comporte comme dans (E.7) et reste presque sûrement au voisinage $V(K_1 \cup \dots \cup K_r)$ des attracteurs, tandis que la probabilité

de présence dans n'importe quel Borélien A disjoint de $K_1 \cup \dots \cup K_r$ disparaît.

$$\lim_{\varepsilon \rightarrow \infty} \mu^\varepsilon(V(K_1 \cup \dots \cup K_r)) = 1, \text{ et } \lim_{\varepsilon \rightarrow \infty} \mu^\varepsilon(A) = 0$$

Le résultat principal de Freidlin et Wentzell repose sur l'équivalence du processus $(X_t^\varepsilon)_{t \in \mathbf{R}_+}$ à temps continu et espace d'états \mathbf{R}^d et du processus $(Z_n^\varepsilon)_{n \in \mathbf{N}}$ à temps discret et espace d'états fini $\{1, \dots, r\}$ décrivant les visites au $n^{\text{ième}}$ attracteur.

La construction précise de $(Z_n^\varepsilon)_{n \in \mathbf{N}}$, n'est pas détaillée ici mais nous en donnons un aperçu afin de mieux comprendre ce dernier processus.

Si x_{ini} est proche de l'attracteur K_h alors $Z_0^\varepsilon = h \in \{1, \dots, r\}$, puis le processus, sous l'influence de (w_t) , est attiré par K_s et $Z_1^\varepsilon = s$, etc.

La chaîne de Markov ainsi créée a pour espace d'états $\{1, \dots, r\}$, est irréductible, et possède une unique mesure de probabilité invariante ν^ε .

Théorème 2.3.2. *L'étude du comportement asymptotique de la mesure μ^ε est "équivalente" à l'étude du comportement asymptotique de la mesure ν^ε .*

Notons toutefois que ces probabilités de transition s'écrivent :

$$P[Z_n^\varepsilon = i | Z_n^\varepsilon = j] \underset{ln}{\sim} \exp\left(-\frac{V(i, j)}{\varepsilon^2}\right)$$

où $V(i, j) = \inf\{V(f), f(\cdot) \text{ continue } [0, 1] \rightarrow \mathbf{R}^d, f(0) \in K_i, f(1) \in K_j\}$

et $V(f) = \int_0^1 |f(t) - b(f(t))|^2 dt$ est une constante associée à f et caractérisant sa vitesse de convergence.

La quantité $V(i, j)$ ou coût de communication, mesure le coût de passage de l'attracteur K_i à l'attracteur K_j .

Les intensités de transitions de la chaîne $(Z_n^\varepsilon)_{n \in \mathbf{N}}$, nous ouvrent la voie pour déterminer la mesure invariante ν^ε .

Mesure invariante ν^ε

Les outils qui permettent de déterminer cette mesure invariante ont été développés, une nouvelle fois par Freidlin et Wentzell, nous aurons besoin de certains d'entre eux.

Définition 2.3.1. *Soit i un élément de $\{1, \dots, r\}$.*

Un i -graphe sur $\{1, \dots, r\}$ est un graphe g sur $\{1, \dots, r\}$ possédant les propriétés suivantes :

- $\forall j \neq i$, le graphe g contient une unique flèche issue de j ;
- Il existe un chemin dans g qui mène de j à i ;
- g ne contient pas de flèche issue de i .

Il s'agit donc d'un graphe sans cycles formé de chemins qui aboutissent en i . On note $G(i)$ l'ensemble des i -graphes sur $\{1, \dots, r\}$.

Définition 2.3.2. *La fonction d'énergie virtuelle W est la fonction de $\{1, \dots, r\}$ dans \mathbf{R}_+ définie par :*

$\forall i \in \{1, \dots, r\} \quad W(i) = \min \left\{ \sum_{(\alpha \rightarrow \beta) \in g} V(\alpha, \beta) \mid g \in G(i) \right\}$
 A cette fonction est associé l'ensemble W^* des minima globaux de W .

Finalement, la mesure invariante ν^ε est caractérisée par :

$$\forall i \in \{1, \dots, r\} \quad \nu^\varepsilon(i) \underset{\sim}{\sim} \exp \left(-\frac{w(i) - w(w)}{\varepsilon^2} \right)$$

où $W(W^*) = \min\{W(i) : i \in \{1, \dots, r\}\}$ et le signe $\underset{\sim}{\sim}$ signifie que les logarithmes des deux fonctions sont équivalentes.

Le comportement asymptotique de ν^ε (et par la même occasion de μ^ε) est donc connu : la mesure ν^ε se concentre sur les attracteurs dont l'indice est dans W^* et décroît vers zéro à la vitesse $\exp\left(\frac{-Cste}{\varepsilon^2}\right)$ pour les autres attracteurs. Il existe donc un sous-ensemble de W^* de l'ensemble des attracteurs sur lequel se concentre la mesure invariante du processus.

$$\lim_{\varepsilon \rightarrow 0} \lim_{t \rightarrow \infty} P[X_t^\varepsilon \in V(\cup_{i \in W^*} K_i) \mid X_0^\varepsilon = x_{ini}] = 1$$

Dynamique du processus

Cerf nous donne dans sa thèse une très claire interprétation de la hiérarchie des cycles qui caractérisent la dynamique du processus. Supposons que le processus soit initialement dans le bassin d'attraction de K_1 . Il le quitte au bout d'un temps fini. Parmi toutes les trajectoires de sortie, il en existe une plus "probable" que les autres, qui l'amène vers un nouvel attracteur ; par exemple K_2 puis, bientôt K_3 . L'ensemble des attracteurs étant par hypothèse fini, le processus finit par revisiter un attracteur formant un cycle d'ordre 1 sur lequel le processus tourne longtemps. Englobons maintenant ces trois attracteurs dans une boîte. Comme toujours, les perturbations browniennes finissent par pousser le processus hors de cette boîte, et ici encore, il existe une trajectoire de sortie canonique qui fait tomber le processus dans un nouveau bassin d'attraction, ou plus généralement, dans un autre cycle d'ordre 1.

Les cycles d'ordre 1 sont aussi en nombre fini, et le processus finit par revisiter un cycle d'ordre 1 : un cycle d'ordre 2 est alors formé, dans lequel le processus reste piégé très longtemps. En continuant de la sorte, il est possible de construire toute une hiérarchie de cycles qui épuise l'ensemble des attracteurs et fournit une image très précise de la dynamique asymptotique du processus. À chaque transition entre cycles est associée une constante qui caractérise la difficulté de la transition.

Enfin, lorsque ε décroît avec le temps ($\varepsilon = \varepsilon(t)$ est une fonction de t qui décroît vers 0 quand t tend vers l'infini), nous obtenons un processus de Markov inhomogène (le mécanisme de transition dépend du temps) ;

- Si $\varepsilon(t)$ décroît très lentement, de sorte qu'à chaque instant la loi de X_t soit proche de l'état d'équilibre associé au niveau de perturbation $\varepsilon(t)$, la situation ne change pas fondamentalement. La loi limite correspond à la limite de la suite des lois d'équilibre.
- Si au contraire $\varepsilon(t)$ décroît très rapidement, le processus risque de rester piégé dans certains sous-ensembles d'attracteurs : plus précisément, dans la hiérarchie des cycles, certaines transitions ne pourront être effectuées qu'un nombre fini de fois, alors que d'autres, plus *faciles*, seront réalisées une infinité de fois avec probabilité 1. La loi limite dépend alors fortement du point de départ.

La hiérarchie des cycles permet ainsi de décrire les dynamiques possibles de (X_t) en fonction de la vitesse de décroissance de $\varepsilon(t)$.

2.3.7 Résultats de convergence

Lorsque l croît vers l'infini, les perturbations affectant le processus (X_k^l) diminuent de sorte que cette chaîne se comporte, presque sûrement, comme la chaîne (X_k^∞) . Plus précisément, nous savons que les attracteurs de la chaîne (X_k^∞) sont les populations d'équidaptation S et les populations uniformes (attracteurs stables). La chaîne (X_k^l) va donc être attirée par ses attracteurs, en commençant par l'ensemble S .

La théorie de Freidlin et Wentzell nous permet de reporter cette étude sur celle de la chaîne des (Z_k^l) des visites successives de (X_k^l) à l'ensemble S . Nous poserons donc $Z_k^l = X_{T_k}^l$ où T_k est l'instant de la $k^{\text{ème}}$ visite de (X_k^l) dans S .

Les probabilités de transition de la chaîne (Z_k^l) , sont estimées à l'aide des opérateurs définis en (E.2.3) et selon le schéma développé ci-dessus. Les fonctions de coût de communication $V(i, j)$ et d'énergie virtuelle W sont définies et estimées.

Nous savons alors que la suite des mesures stationnaires de la chaîne (X_k^l) se concentre sur l'ensemble W^* des minima de W :

$$\forall x_{ini} \in E^N \lim_{l \rightarrow \infty} \lim_{k \rightarrow \infty} P[X_t^l \in W^* | X_0^l = x_{ini}] = 1$$

L'un des principaux résultats indique qu'il existe une taille de la population de (X_k^l) , (taille critique) telle que les maxima de f soient atteints asymptotiquement avec la probabilité 1.

Taille critique

Supposons fixés l'espace d'états E , la fonction d'adaptation f , les noyaux de transition de mutation a et de croisement b , ainsi que les constantes positives gérant les trois opérateurs a , b , et c .

Théorème 2.3.3. [27]

Il existe une valeur critique N^ , telle que lorsque la taille de la population de l'algorithme génétique dépasse N^* , l'ensemble f^* des maxima globaux de f , contient l'ensemble W^* .*

Cette taille critique N^ , dépend fortement de l'espace d'états E , de la fonction d'adaptation f , des noyaux de transition de mutation α et de croisement β , ainsi que des paramètres a , b , et c .*

Une borne grossière, mais lisible de N^ est :*

$$N^* \leq \frac{aR + c(R-1)\Delta}{\min(a, \frac{b}{2}, c\delta)}$$

où :

- R est le nombre minimal de transition permettant de joindre deux points arbitraires de E par mutation ;
- Δ et δ sont des paramètres d'échelle :
 - ★ $\Delta = \max\{|f(i) - f(j)| : i, j \in E\}$ paramètre mesurant les écarts maximaux de f ;

★ $\delta = \min\{|f(i) - f(j)| : i, j \in E, f(i) \neq f(j)\}$ mesurant les écarts minimaux de f .

Il est intéressant de relever que le résultat est obtenu sans faire intervenir l'opérateur de croisement, qui n'est donc pas indispensable. L'exploration par mutation et la sélection suffisent à assurer la convergence vers f^* ([109]).

Ce premier résultat nous indique que dès que $N \geq N^*$, la suite des mesures stationnaires de la chaîne (X_k^l) se concentre asymptotiquement sur f^* lorsque l tend vers l'infini. On peut dans une étape suivante faire évoluer l , et donc l'intensité des perturbations, en fonction de la génération. Nous obtenons alors une chaîne de Markov inhomogène $(X_k^l(k))$ dont le mécanisme de transition dépend alors de la génération k .

Vitesse de convergence

Le principal problème est de savoir si cette chaîne inhomogène peut avoir un comportement proche de celui de la chaîne homogène (X_k^l) , et si oui, sous quelles conditions. La vitesse de croissance de $l(k)$ vers l'infini, est bien évidemment, au centre du débat.

- Si $l(k)$ croît *lentement*, alors la loi de X_k sera proche de la loi stationnaire $\mu^\varepsilon(l(n))$ de niveau de perturbation $\varepsilon(l(n))$ associé à $l(n)$.
- Si $l(k)$ croît *rapidement*, alors X_k risque de rester piégé dans des bassins d'attraction ne correspondant pas aux maxima de f , l'intensité des perturbations devenant trop faible pour pouvoir s'en échapper.

La vitesse recherchée se situe entre ces deux extrêmes, permettant à X_k de s'échapper des *mauvais* bassins d'attraction (ne correspondant pas à des maxima de f) et de rester piégé dans le *bon* (celui des points de f^*).

La vitesse de convergence de la suite $l(k)$ est caractérisée par l'exposant de convergence, λ .

Définition 2.3.3. *L'exposant de convergence λ de la suite $l(k)$ est l'unique réel λ tel que :*

$$\sum_{k \in \mathbb{N}} l(k)^{-\theta} \begin{cases} \rightarrow & \text{converge pour } \theta > \lambda \\ \rightarrow & \text{diverge pour } \theta < \lambda \end{cases}$$

Deux conditions pour la colonisation de f^* sont également données par Cerf, l'une est nécessaire, l'autre est suffisante.

Théorème 2.3.4. *Condition nécessaire pour la colonisation de f^**

Pour que : $(\forall x_{ini} \in E^N) P[\exists K \forall k \geq K [X_{T_k}] \subset f^ | X_0 = x_{ini}] = 1$*

c'est-à-dire, pour que la chaîne $Z_k = X_{T_k}$ des visites successives des attracteurs soit piégée dans f^* après un nombre fini K de transitions, il est nécessaire que l'exposant de convergence l de la suite $l(k)$ appartienne à l'intervalle $]\phi, \psi[$.

Les constantes ϕ et ψ sont des caractéristiques du problème, l'intervalle $]\phi, \psi[$ est alors non vide pour N assez grand.

Théorème 2.3.5. *Condition suffisante pour la colonisation de f^**

Il existe deux constantes h et r telles que si l'exposant de convergence l de la suite $l(k)$ appartient à l'intervalle $]h, r[$, alors :

$$(\forall x_{ini} \in E^N) P[(\exists K) (\forall k \geq K) [X_{T_k}] \subset f^*, \bar{X}_k \subset f^* | X_0 = x_{ini}] = 1$$

ce qui signifie qu'après un nombre fini de transitions, nous avons presque sûrement, la situation suivante :

- la chaîne X_{T_k} est piégée dans f^* ,
- la population X_k contient toujours un ou des individus appartenant à f^* .

2.4 Conclusion

D'autres résultats existent dans la littérature. Par exemple, le résultat de convergence obtenu par Rudolph [94] en se basant seulement sur la positivité de l'opérateur de mutation (il existe une mutation qui lie deux points quelconque de l'espace avec une probabilité non nulle), condition qui a été relaxé par Agapie [3] en la remplaçant par le fait que la matrice de transition qui correspond à l'opérateur de mutation soit irréductible (existence d'une suite finie de mutations qui lie deux points quelconque de l'espace avec une probabilité non nulle) et ses éléments diagonaux soient positive.

Le plus important résultat est la convergence en probabilité vers l'optimum global de la variante d'algorithme génétique composée de l'opérateur de croisement avec un point, un opérateur de mutation tel quel est défini précédemment et la méthode de sélection définie par la formule (E.5).

Il y a une autre théorie qui se base sur la notion du paysage dont Weinberger [104] est à l'origine et qui fut développée par Stadler ([98] [97]), [46] et [33]. Elle permet de déterminer le niveau de difficulté d'une instance d'un problème pour l'algorithme génétique [63] en utilisant la fonction d'autocorrélation.

De nombreuses interrogations demeurent, cependant, concernant les relations réelles entre les différents paramètres caractérisant l'algorithme génétique et les choix pratiques de ceux-ci. Dans ce domaine, la pratique devance encore la théorie.

Deuxième partie

Le problème du voyageur de commerce (T.S.P.)

Introduction

L'algorithme génétique est considéré comme une boîte de Pandore. On lui donne des valeurs d'entrées pour résoudre un problème donné (finance, analyse de données, recherche opérationnelle, etc.) et il fournit des résultats sans que les utilisateurs n'aient à se soucier de son fonctionnement interne.

Vous savez, maintenant, que l'algorithme génétique contient plusieurs paramètres qu'il faut les ajuster pour qu'il donne de bons résultats. Choisir une méthode de sélection, un opérateur de croisement, un opérateur de mutation, les probabilités de croisement et de mutation et la méthode d'insertion vous permettra d'obtenir une variante de l'algorithme génétique.

Dans cette première partie, nous essayons de répondre à cette attente en utilisant comme difficulté le problème du voyageur de commerce qui est un problème NP complet. Dans la théorie de complexité, les problèmes NP-complets sont les problèmes les plus difficiles de NP. Trouver un algorithme qui résout n'importe quel problème NP-complet dans un temps polynômial, permettra à coup sûr de résoudre tous les problèmes NP rapidement. Dans le premier chapitre, nous présenterons le problème du voyageur de commerce ainsi que certaines méthodes de résolution, déterministes et approximatives.

Le deuxième chapitre est consacré d'abord à présenter huit opérateurs de croisement et cinq de mutation, et de comparer les résultats obtenus par chaque opérateur. L'objectif escompté est d'obtenir le duo, opérateur croisement et mutation, qui fonctionne le mieux avec leurs probabilités respectives.

Dans le troisième chapitre, nous nous intéresserons à la manière de sélectionner les individus pour un éventuel croisement et nous en présenterons six que nous comparons dans le but d'améliorer la convergence de notre algorithme.

Le quatrième chapitre est consacré à l'analyse et l'étude comparative de la manière de génération de la population initiale.

Dans le cinquième chapitre, nous présenterons notre variante de la résolution du problème du voyageur de commerce.

Chapitre 1

Problème du voyageur de commerce (T.S.P.)

Sommaire

1.1	Historique du problème du voyageur de commerce (T.S.P.)	33
1.2	Méthodes de résolutions du problème T.S.P.	35
1.3	Algorithmes déterministes	35
1.3.1	Méthode de séparation et d'évaluation (Branch and Bound)	36
1.3.2	Méthode des plans sécants (Cutting plane)	37
1.4	Algorithmes approximatifs	37
1.4.1	Le recuit simulé	37
1.4.2	L'algorithme de colonies de fourmis	39
1.4.3	La méthode recherche tabou	42
1.4.4	Algorithme de Lin et Kernighan	43

1.1 Historique du problème du voyageur de commerce (T.S.P.)

Le problème du voyageur de commerce, ou le problème du commis de commerce, que nous noterons par la suite T.S.P. (Travelling Salesman Problem), a pour objectif de chercher le parcours minimal reliant V points distincts donnés et passant une et une seule fois par chaque point. Le point peut représenter une ville, location, entrepôt,....

Autrement dit, un T.S.P. de taille V est défini par un ensemble de points $v = \{v_1, v_2, \dots, v_V\}$ pour lesquels on définit une fonction de distance métrique d . Une solution de T.S.P. correspond à une forme d'ordonnancement $v_\pi = (v_{\pi(1)}, v_{\pi(2)}, \dots, v_{\pi(V)})$ où π est une permutation sur l'ensemble $\{1, 2, \dots, V\}$. Chaque V -uplet représente une solution réalisable au problème T.S.P..

La fonction d'évaluation calcule l'adaptation de chaque V-uplet par la formule suivante :

$$f(v_\pi) = \left(\sum_{i=1}^{i=V-1} d(v_{\pi(i)}, v_{\pi(i+1)}) \right) + d(v_{\pi(V)}, v_{\pi(1)})$$

où $d(v_{\pi(i)}, v_{\pi(i+1)})$ est la distance, ou le coût de déplacement, entre deux points $v_{\pi(i)}$ et $v_{\pi(i+1)}$.

Dans la littérature, il y a deux types de problème du voyageur de commerce : T.S.P. symétrique et T.S.P. asymétrique. On dit que nous sommes dans le premier cas lorsque $d(v_i, v_j) = d(v_j, v_i)$ pour tout $i \neq j$, c'est-à-dire que le coût de déplacement de la ville i à la ville j est le même dans les deux sens. Dans le cas contraire, le problème est dit asymétrique. Le travail dans la suite s'inscrit dans le cas d'un T.S.P. symétrique puisque tout T.S.P. asymétrique de taille V peut être transformé en un problème T.S.P. symétrique de taille $2 * V$.

La formulation mathématique de T.S.P. peut se traduire par :

$$\min\{f(v_\pi), v_\pi = (v_{\pi(1)}, v_{\pi(2)}, \dots, v_{\pi(V)})\}$$

où π est une permutation de $\{1, 2, \dots, V\}$.

Les premières approches mathématiques exposées pour le problème du voyageur de commerce ont été traitées au 19^{ème} siècle par les mathématiciens Sir William Rowan Hamilton et Thomas Penyngton Kirkman ([20]). Hamilton a fait un jeu qui s'appelle le jeu icosian de Hamilton (Hamilton's Icosian game ([20])). Dans ce jeu, les joueurs devaient réaliser une tournée passant par 20 points en utilisant uniquement les connections prédéfinies. Pendant les années trente, Le T.S.P. a été traité plus en profondeur par Karl Menger à Harvard ([95]). Il est ensuite développé à Princeton par les mathématiciens Hassler Whitney et Merrill Flood ([95]). Une attention particulière est portée sur les connections par Menger et Whitney ainsi que sur la croissance du T.S.P..

En 1954, Dantzig a résolu un problème T.S.P. de 49 villes en utilisant la méthode du cutting-plane [35]. Camerini, Fratta et Maffioli ont trouvé la solution pour un problème de 100 villes en 1974 ([25]). Après 13 ans, Padberg et Rinaldi ont résolu un problème de 532 villes ([89]) puis un autre de 2392 villes ([90]). En 2001, Applegate, Bixby, Chvátal et Cook des universités de Rice et Princeton ont obtenu la solution optimale pour un problème de 15112 villes en Allemagne ([6]).

L'existence d'un algorithme déterministe exacte avec une complexité polynomiale reste inconnue. Un calcul rapide de la complexité montre qu'elle est en $O(V!)$ où V est le nombre de villes. En supposant que le temps pour évaluer un trajet quelque soit V est de $1 \mu s$, le tableau (TAB.1.1) montre l'explosion combinatoire du T.S.P..

Nb villes	Nb possibilités	Temps de calcul
5	12	12 μs
10	181440	0,18 ms
15	43 milliards	12 heures
20	60×10^{15}	1928 ans
25	310×10^{21}	9,8 milliards d'années

TAB. 1.1: Nombre de chemins possibles et temps de calcul en fonction du nombre de villes

Le T.S.P. appartient à la famille des problèmes NP-complet dont les méthodes de résolution peuvent s'appliquer à d'autres problèmes de mathématiques discrètes. Néanmoins, il a aussi des applications directes, notamment dans le transport et la logistique. Par exemple, trouver le chemin le plus court pour les bus de ramassage scolaire ou, dans l'industrie, trouver la plus courte distance que devra parcourir le bras mécanique d'une machine pour percer les trous d'un circuit imprimé (les trous représentent les villes).

1.2 Méthodes de résolutions du problème T.S.P.

Pour résoudre le T.S.P, on trouve dans la littérature les **algorithmes déterministes** (exacts) et les **algorithmes d'approximation** (heuristiques).

Les algorithmes déterministes permettent de trouver la solution optimale, mais leur complexité est d'ordre exponentiel. Parmi les algorithmes les plus utilisés, nous citons la méthode de séparation et d'évaluation et la méthode des plans sécants [35]. Ces algorithmes demandent beaucoup de places mémoires et ils sont très gourmands en temps de calcul. En plus, lorsque le problème est de grande dimension ces algorithmes deviennent inutilisables.

Les algorithmes d'approximation permettent de trouver en un temps raisonnable une solution approchée de la solution optimale. Ils sont utilisés généralement pour résoudre des problèmes concrets de grandes tailles. Outre que l'algorithme génétique, nous présenterons recuit simulé, colonie de fourmis, recherche tabou et l'algorithme de Lin et Kernighan.

1.3 Algorithmes déterministes

Le T.S.P. peut être modélisé en un problème de programmation linéaire en nombres entiers sous contraintes comme suit :

À chaque ville correspond un entier i , compris entre 1 et V . Pour chaque couple de villes (i,j) , on définit par c_{ij} le coût de passage de la ville i à la ville j et la variable binaire :

$$x_{ij} = \begin{cases} 1 & \text{si le voyageur passe de la ville } i \text{ à } j \text{ ou inversement,} \\ 0 & \text{sinon.} \end{cases}$$

Le T.S.P. s'écrit donc sous forme de problème de programmation linéaire en nombres entiers :

$$\min \sum_{i=2}^n \sum_{j=1}^{i-1} c_{ij} x_{ij}$$

sous les contraintes suivantes :

1. $\sum_j x_{ij} = 2, \forall i \in \mathcal{N} = \{1, 2, \dots, n\}$
2. $\sum_{i \in \mathcal{S}} \sum_{j \notin \mathcal{S}} x_{ij} \geq 2$ pour tout $\mathcal{S} \subset \mathcal{N}$

La première contrainte vérifie que pour toute ville i la somme des variables ayant pour extrémité i est égale à 2, ainsi nous vérifions que chaque ville est visitée une et une seule fois. La deuxième contrainte permet d'enlever les sous-cycles et de n'en laisser qu'un seul passant par toutes les villes. En effet, pour n'importe quel sous-ensemble \mathcal{S} de villes, le parcours est réalisé et sort de cet ensemble.

Cette modélisation permettra d'utiliser toutes les méthodes de résolution pour ce type de problèmes, notamment, la méthode de séparation et d'évaluation et la méthode de cutting-plane.

1.3.1 Méthode de séparation et d'évaluation (Branch and Bound)

La méthode de séparation et d'évaluation s'applique à la résolution des problèmes d'optimisation combinatoire avec un grand nombre de solutions envisageables. Nous pouvons associer cette méthode à une arborescence dont la racine contient l'ensemble de toutes les solutions réalisables (c.à.d. solutions satisfaisant les contraintes). Les autres sommets émanant de la racine sont régies par les méthodes de séparation, évaluation et stratégie de développement.

Principe de séparation

Ce principe consiste à séparer l'ensemble de solutions réalisables contenues dans un sommet de l'arborescence en sous-ensembles. L'union de ceux-ci est le sommet lui même.

Principe d'évaluation

Dans un problème de minimisation, la borne correspond à une valeur qu'on sait atteindre pour la fonction objective à l'aide d'une certaine solution réalisable, et qui est donc par définition un majorant du minimum. Évaluer un sommet c'est déterminer un minorant de l'ensemble des valeurs de l'objectif correspondant aux solutions contenues en ce sommet. Une évaluation est dite exacte si la valeur est atteinte par un élément de l'ensemble associé au sommet. La borne et l'évaluation permettent de ne pas développer un sommet S dans les deux cas suivants :

- lorsque l'évaluation de S est supérieure ou égale à la borne ;
- lorsque l'évaluation de S est exacte, et si cette évaluation est inférieure à la borne alors cette évaluation deviendra la borne par la suite.

Stratégie de développement

Elle désigne dans quel ordre doit-on appliquer le principe de séparation. Nous distinguons dans la littérature, en général, deux stratégies de développement : en profondeur d'abord et en largeur d'abord. La première consiste à descendre les branches jusqu'à ce qu'on trouve un sommet que l'on peut éliminer par le critère d'évaluation. Alors que la deuxième stratégie s'intéresse à traiter en largeur tous les sommets de même niveau de l'arborescence avant d'étudier les sommets du niveau suivant. Celle-ci est moins utilisée que la première stratégie à cause du problème d'encombrement de mémoire.

1.3.2 Méthode des plans sécants (Cutting plane)

C'est une alternative à la méthode de séparation et d'évaluation utilisée également pour résoudre des problèmes en nombres entiers. L'idée fondamentale est d'ajouter des contraintes adéquates appelées *coupes* à un problème linéaire jusqu'à ce que la solution faisable de base optimale prenne des valeurs en nombres entiers. Une coupe relativement à une solution partielle courante satisfait les critères suivants :

1. chaque solution faisable en nombres entiers est faisable pour la coupe, et
2. la solution partielle courante n'est pas faisable pour la coupe.

Il y a deux manières différentes de générer les coupes. La première, appelée coupes de Gomory, produit des coupes de n'importe quel tableau de la programmation linéaire. Ceci a l'avantage de résoudre n'importe quel problème mais son inconvénient est qu'elle est très lente. La deuxième approche est d'employer la structure du problème pour produire de très bonnes coupes. Elle a besoin d'une analyse de cas par cas, mais peut fournir des techniques de résolution très efficaces.

1.4 Algorithmes approximatifs

1.4.1 Le recuit simulé

Les thermodynamiciens ont remarqué qu'à partir d'un même liquide, on peut obtenir différents solides selon la vitesse de décroissance de la température. En effet,

- ★ Si la baisse de la température est brutale, elle produira à un minimum local d'énergie une structure amorphe, un verre par exemple.
- ★ Si la baisse de la température est progressive de façon à atteindre le minimum global d'énergie. On obtiendra par exemple le cristal.

La méthode de recuit simulé s'inspire de la méthode de Métropolis (1953, [74]) qui était utilisée pour modéliser les processus physiques. Elle a été proposée par S. Kirkpatrick et al. en 1983 ([66]). Ces derniers se sont inspirés de la mécanique statistique en utilisant en particulier la distribution de Boltzmann. Ainsi, la probabilité P pour qu'un système physique passe d'un niveau d'énergie E_1 à un niveau E_2 est donnée par :

$$P = e^{\frac{-(E_2 - E_1)}{kT}} \quad (1.1)$$

où k est la constante de Boltzmann, $k=1,3805 \cdot 10^{-23}$ J/K et T est la température.

L'équation (1.1) peut s'écrire sous la forme :

$$kT \times \log(P) = E_1 - E_2 \quad (1.2)$$

Cette dernière équation (1.2) montre que la probabilité d'observer une augmentation de l'énergie est d'autant plus grande que la température est élevée, donc au niveau du recuit simulé :

- ★ Une diminution de la fonction sera toujours acceptée ;
- ★ Une augmentation de la fonction sera acceptée avec une probabilité définie par l'équation (1.1).

Dans les applications numériques, le paramètre k prend dans un premier temps la valeur 1 et T est un paramètre formel qui joue le rôle de la température.

La méthode recuit simulé est illustrée dans l'algorithme suivant avec X_0 représentant le vecteur initial et une fonction voisin(X) qui calcule un vecteur aléatoire Y appartenant au voisinage de X .

Fonction recuit(X_0 :vecteur initial) : vecteur

$X \leftarrow X_0$

Tant que (Non condition d'arrêt) **faire**

$m \leftarrow 0$

Répéter

$Y \leftarrow \text{voisin}(X)$

$dF \leftarrow F(Y) - F(X)$

Si (Accepte(dF, T)) **Alors**

$X \leftarrow Y$

Fin Si

$m \leftarrow m + 1$

jusqu'à ce que ($m = N_T$ Nombre d'itérations maximum à la température T)

$T \leftarrow \text{Decroissance}(T)$

Fait

Fin

Algorithme 1: Le schéma général du recuit simulé

La condition d'arrêt peut être le nombre maximum d'itérations. La fonction "accepte" permet d'accepter la solution lorsque la fonction f décroît et avec une probabilité lorsque f croît. Ceci est illustré dans le pseudo-code suivant :

```

dF : vecteur initial;
T : température;
Si (dF ≤ 0) Alors
  | Retourner VRAI
Sinon
  A ← e-dF/T;
  Si (Alea(0,1) ≤ A)Alors
    | Retourner VRAI
  Sinon
    | Retourner FAUX
  Fin Si
Fin Si

```

Algorithme 2: La fonction accepte

Les résultats de recuit simulé dépendent essentiellement du choix des paramètres suivants : la température initiale, la manière de diminuer la température et le critère d'arrêt. On essaie lors de l'application de cette méthode de choisir initialement une température T très grande et de la faire baisser progressivement afin de ne pas tomber dans les optimums locaux.

1.4.2 L'algorithme de colonies de fourmis

Des biologistes ont observé que certains types de fourmis réelles sont capables de trouver le plus court chemin menant de la source de la nourriture vers leur nid ([56],[14] sans utiliser la vue en exploitant l'information du phéromone[60].

Ils ont recueilli les observations suivantes :

- les fourmis au départ recherche aléatoirement la nourriture
- les fourmis déposent des phéromones sur leur chemin ;
- elles suivent le chemin marqué par cette substance ;
- le chemin le plus marqué est préféré ;
- les phéromones s'évaporent avec le temps ;
- si on pose un obstacle coupant un chemin de phéromones, les fourmis le contournent par un côté au hasard.

Dès qu'une fourmi trouve la nourriture, à un instant donné, une plus grande quantité de phéromone sera déposée sur le chemin le plus court. Ceci est dû au fait que les fourmis ayant choisi ce chemin auront déposé leurs phéromones en premier. Celles qui traversent le passage dans l'autre sens y trouveront plus de phéromones à cet instant. L'algorithme

de colonies de fourmis a été utilisé pour la première fois pour trouver le plus court chemin.

En 1992, Marco Dorigo ([42]) s'est inspiré du comportement des fourmis pour inventer l'algorithme de colonies de fourmis pour le problème du T.S.P.. Il procède comme suit :

Au début toutes les m fourmis sont uniformément distribuées sur toutes les villes. Chaque fourmi k est un agent simple possédant les caractéristiques suivantes :

- $\tau_{ij}^k(t)$ est la quantité de phéromone déposée par la fourmi k sur l'arête (i, j) ;
- η_{ij} est un paramètre dit de *visibilité* qui est égal à l'inverse de la distance, c'est-à-dire $\eta_{ij} = \frac{1}{c_{ij}}$. On remarque que lorsque la distance c_{ij} décroît, le paramètre de visibilité croît ;
- J_i^k est l'ensemble de villes possibles quand la fourmi k est dans la ville i . Naturellement cet ensemble exclut les villes qui ont été déjà visitées.
- Elle choisit à partir de la ville i une ville j , qui n'a pas été encore visitée. Cette ville appartient à J_i^k , avec une probabilité P_{ij}^k qui est fonction de la distance entre les villes i et j et de la quantité de phéromones actuelle sur l'arête les reliant ;

$$P_{ij}^k(t) = \begin{cases} \frac{(\tau_{ij}^k(t))^\alpha (\eta_{ij})^\beta}{\sum_{l \in J_i^k} (\tau_{il}^k(t))^\alpha (\eta_{il})^\beta} & \text{si } j \in J_i^k \\ 0 & \text{sinon.} \end{cases}$$

i et j : des indices pour les villes
 k : indice pour les fourmis variant de 1 à m .
 $T^k(t)$: le trajet effectué par la fourmi k
 $L^k(t)$: la longueur du tour faite par la fourmi k à l'itération t

$t \leftarrow 1$

$\forall i \neq j \tau_{ij}(t) = \rho_0$

Tant que ($t \leq t_{max}$) **faire**

Pour k de 1 à m **faire**

 Choisir une ville au hasard i

Pour (chaque ville $j \in J_i^k$ la liste des villes restantes) **faire**

$$P_{ij}^k(t) = \begin{cases} \frac{(\tau_{ij}^k(t))^\alpha \times (\eta_{ij})^\beta}{\sum_{l \in J_i^k} (\tau_{il}^k(t))^\alpha (\eta_{il})^\beta} & \text{si } j \in J_i^k \\ 0 & \text{sinon.} \end{cases}$$

Fin Pour

 Déposer le phéromone sur le trajet $T^k(t)$:

$$\tau_{ij}^k(t) = \begin{cases} \frac{Q}{L^k(t)} & \text{si } (i, j) \in T^k(t) \\ 0 & \text{sinon.} \end{cases}$$

Fin Pour

 Mise à jour du phéromone :

$$\tau_{ij}(t+1) = (1 - \rho)\tau_{ij}(t) + \sum_{k=1}^{k=m} \tau_{ij}^k(t)$$

$t \leftarrow t + 1$

Fait

Algorithme 3: L'algorithme de colonies de fourmis pour résoudre le problème T.S.P.

Les deux paramètres α et β jouent un rôle très important et dépendent des choix préférentiels de l'utilisateur.

Par exemple, si $\alpha = 0$ et $\beta = 1$, l'utilisateur s'intéresse d'abord aux villes les plus proches de la ville i indépendamment de la quantité de phéromones déposée par les fourmis.

Alors que si $\alpha = 1$ et $\beta = 0$, l'utilisateur privilégie la quantité de phéromone déposée par les fourmis vis-à-vis de la distance entre les villes ;

- Pour forcer la fourmi k à faire une tournée légale, il faut qu'elle rejette les transitions aux villes déjà visitées jusqu'à ce qu'une tournée soit accomplie ;
- Quand elle accomplit une tournée, elle étend une substance appelée la phéromone sur chaque arête (i, j) visitée.

Lorsque toutes les fourmis ont terminé leurs tournées, il y a une mise à jour de l'intensité

de la quantité de phéromones sur chaque arête vérifiée par la formule suivante :

$$\tau_{ij}(t+1) = (1 - \rho)\tau_{ij}(t) + \sum_{k=1}^{k=m} \tau_{ij}^k(t)$$

L'algorithme 3 décrit l'algorithme de colonies

de fourmis de base pour résoudre le problème du voyageur de commerce.

L'efficacité de cet algorithme dépend de la quantité de phéromone déposée par une fourmi et la vitesse avec laquelle s'évapore.

1.4.3 La méthode recherche tabou

La méthode tabou [100] est une méthode de recherche locale. Son principe repose sur une méthode de déplacement sur l'espace des solutions, tout en cherchant constamment à améliorer la meilleure solution courante et en conservant en mémoire la liste des précédents déplacements et ainsi guider la recherche en dehors de zones précédemment parcourues. En général, on ne va pas garder tous les déplacements (trop coûteux en mémoire), mais on va seulement empêcher l'accès à certaines solutions pendant un certain nombre d'itérations. La méthode consiste à se déplacer d'une solution vers une autre par observation du voisinage de la solution de départ et à définir des transformations tabous que l'on garde en mémoire. Une transformation tabou est une transformation que l'on s'interdit d'appliquer à la solution courante. Soient :

- ★ S_0 , la solution initiale.
- ★ $x \mapsto f(x)$, la fonction à minimiser.
- ★ S^* , la meilleure solution jusqu'à présent.
- ★ $F^* = f(S^*)$, la valeur de la fonction objectif de la meilleure solution trouvée.

Une fois ces définitions fixées, il suffit de suivre le schéma de cet algorithme :

```

Fonction tabou( $S_0$  :solution initiale,  $L$  liste) : solution
   $S^* \leftarrow S_0$ ;
   $F^* \leftarrow f(S_0)$ ;
   $t \leftarrow 0$ ;
   $S_t \leftarrow S_0$ ;
  Tant que (le test d'arrêt n'est pas réalisé) faire
    Tant que (il existe une nouvelle solution  $S$  au voisinage de  $S_t$  qui  $\notin L$ )
      faire
        Fait
         $t \leftarrow t + 1$ ;
         $S_t \leftarrow S$ ;
        Si ( $f(S) < F^*$ ) Alors
           $S^* \leftarrow S$ ;
           $F^* \leftarrow F(S)$ ;
           $L \leftarrow L \cup S^*$ ;
        Fin Si
      Fait
    Fin
  Fin

```

Algorithme 4: La fonction de recherche tabou

Le critère d'arrêt de l'algorithme peut être un nombre d'itérations maximal, ou un temps à ne pas dépasser. Les tabous sont une manière de représenter la mémoire du cheminement effectué pour diriger l'exploration vers des régions non visitées.

La liste L de taille fixée N contenant les N dernières solutions rencontrées et on empêche la procédure d'y retourner. On gère cette liste comme une liste circulaire : on élimine le plus vieil élément de la liste tabou et on insère la nouvelle solution (cette manière de faire s'avère coûteuse en terme de quantité d'informations requises).

Il existe plusieurs variantes de la méthode de recherche tabou qui dépendent essentiellement du choix d'un voisinage et de la manière de gérer la liste tabou.

1.4.4 Algorithme de Lin et Kernighan

Un réseau complet de N sommets (où tous les sommets sont joignables deux à deux) admet $\frac{N \times (N-1)}{2}$ arêtes. On définit un système de voisinage qui se base sur λ échanges. Ainsi, deux circuits sont voisins lorsqu'on peut passer de l'un à l'autre en changeant λ arêtes. Bien sûr, le paramètre λ est définie à priori. On dit qu'un circuit hamiltonien est λ -optimum lorsqu'il est impossible de trouver un circuit plus court en remplaçant à chaque fois λ arêtes par d'autres. Il découle naturellement de cette définition que pour tous λ', λ avec $1 \leq \lambda' \leq \lambda$, un circuit λ -optimum est aussi λ' -optimum. Ainsi, si un circuit Hamiltonien pour un T.S.P. contenant V points est V -optimum alors il est solution

optimale du problème. Cette méthode de recherche locale a un temps de complexité de l'ordre de $O(N^\lambda)$. Malheureusement, le nombre d'opérations pour tester toutes les λ échanges croît rapidement à fur et à mesure que λ croît. Dans la littérature, le paramètre λ prend généralement les valeurs 2 et 3. En 1972, Christofides et *al.* [31] ont expérimenté les valeurs 4 et 5.

Fonction LKH() : T solution

```

1 : générer aléatoirement une solution initiale T ;
2 :  $i \leftarrow 1$  choisir  $t_1$  ;
3 : choisir  $x_1 = (t_1, t_2) \in T$  ;
4 : choisir  $y_1 = (t_2, t_3) \notin T$  tel que  $G_1 = g_1 = c(x_1) - c(y_1) > 0$  ;
   si ceci n'est pas possible aller à 12  $t \leftarrow 0$  ;
5 :  $i \leftarrow i + 1$ 
6 : choisir  $x_i = (t_{2i-1}, t_{2i}) \in T$  tel que
   si  $t_{2i}$  rejoint  $t_1$  alors on a une autre solution  $T'$  et
    $x_i \neq y_s \forall s < i$  Si ( $f(T') < f(T)$ ) Alors
       |  $T \leftarrow T'$  ; et aller à 2
       Fin Si
7 : choisir  $y_i = (t_{2i}, t_{2i+1}) \notin T$  tel que
    $G_i = g_1 + g_2 + \dots + g_i > 0$ 
    $y_i \neq y_s \forall s < i$  et  $x_{i+1}$  existe.
Si ( $y_i$  existe ) Alors
   | aller à 5
Fin Si
8 : Si (s'il existe  $y_2$  non encore essayé ) Alors
   |  $i=2$  et aller à 7
   Fin Si
9 : Si (s'il existe  $x_2$  non encore essayé ) Alors
   |  $i=2$  et aller à 6
   Fin Si
10 : Si (s'il existe  $y_1$  non encore essayé ) Alors
   |  $i=1$  et aller à 4
   Fin Si
11 : Si (s'il existe  $x_1$  non encore essayé ) Alors
   |  $i=1$  et aller à 3
   Fin Si
12 : Si (s'il existe  $t_1$  non encore essayé ) Alors
   | aller à 2
   Fin Si
13 : Stop ou aller à 1.

```

Fin

Algorithme 5: Une version simple de l'algorithme de Lin-Kernighan

Le point faible de l'algorithme λ -optimum est le choix difficile du paramètre λ à priori avant son commencement qui permettra de réaliser un bon compromis entre durée d'exécution acceptable et bonne qualité de solution. Pour enlever l'effet négatif de la fixation de λ au départ, Lin et Kernighan ont proposé d'adapter la taille λ de l'échange à au cours d'exécution de l'algorithme. Ils ont implémenté en 1971 ([70]) cet algorithme en se basant sur la généralisation du principe λ adaptative. Cet algorithme de Lin et Kernighan se base sur les voisinages de λ -optimum, avec deux particularités : d'une part, ces voisinages sont explorés (partiellement) pour des grands valeurs de λ , d'autre part l'exploration est suffisamment limitée pour garder une complexité pratique.

L'idée générale de l'algorithme consiste à définir deux ensembles d'arêtes $\{x_1, x_2, \dots, x_\lambda\}$ et $\{y_1, y_2, \dots, y_\lambda\}$. Sans perdre de généralité, on peut considérer que ces deux ensembles sont classés de telle sorte que pour tout i , x_i et y_i se terminent sur le même sommet. On peut donc construire séquentiellement une transformation locale qui remplace x_1 par y_1 , x_2 par y_2 , etc. Puisque la transformation améliore le tour, $G_\lambda = \sum_{i=1}^{i=\lambda} (c(x_i) - c(y_i)) > 0$. Le point-clé est donc de remarquer qu'on peut alors ordonner les i points de sorte que toutes les sommes partielles G_i soient strictement positives. L'algorithme 5 présente un algorithme basique de LKH. Keld Helsgaun ([58]) a implémenté une amélioration de cet algorithme. Il a notamment proposé d'utiliser la mesure de α -sensibilité pour choisir les candidats.

Chapitre 2

Algorithme génétique appliqué au problème du voyageur de commerce

Sommaire

2.1	Les représentations possibles du problème	48
2.1.1	La représentation adjacente	48
2.1.2	La représentation ordinale	48
2.1.3	Représentation chemin	50
2.2	Sélection	50
2.3	Opérateurs de croisement	50
2.3.1	Opérateur de croisement cyclique (cycle par Olivier et al. [84]) :	50
2.3.2	Opérateur de croisement de recombinaison des arcs (edrx par Whitley [105]) :	51
2.3.3	Opérateur de croisement de préservation maximale (MPX par Mühlenbein et al.[81] :)	51
2.3.4	Opérateur de croisement d'ordre 1 (order 1 par Davis et al.[38]) :	52
2.3.5	Opérateur de croisement d'ordre 2 (order 2 par Syswerda[99]) :	52
2.3.6	Opérateur de croisement de position (position par Syswerda [99]) :	53
2.3.7	Opérateur de croisement assorti partiel (pmx par Goldberg et al.[51]) :	53
2.3.8	Opérateur de croisement uniforme (uox) :	53
2.4	Opérateurs de mutation	54
2.4.1	Mutation twors :	54
2.4.2	Mutation de centre inverse (cim) :	54
2.4.3	Mutation inverse (im) :	54
2.4.4	Mutation throas :	55
2.4.5	Mutation thrors :	55
2.5	Méthode d'insertion	55

2.6	Résultats numériques	55
2.6.1	Environnement	55
2.6.2	Analyse	56
2.6.3	Comparaison entre les opérateurs de croisement	57
2.6.4	Comparaison entre les opérateurs de mutation	59
2.6.5	Comparaison entre les interactions des opérateurs de croisement et mutation	61
2.6.6	Discussion sur la taille de la population	61
2.7	Conclusion	64

2.1 Les représentations possibles du problème

Dans cette section, nous allons étudier différentes méthodes de représentation des données puis nous indiquerons laquelle a été retenue pour notre problème.

2.1.1 La représentation adjacente

La représentation adjacente [77] représente la tournée comme une liste de N villes commençant la ville 1. La ville j est listée à la position i si et seulement si le voyageur de commerce se dirige de la ville i à la ville j . Soit par exemple :

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

TAB. 2.1: Représentation adjacente de la tournée 1; 2; 4; 3; 8; 5; 9; 6; 7; 1.

Chaque tournée a seulement une représentation adjacente. Toutefois, quelques listes adjacentes représentent des parcours illégaux. La liste adjacente suivante

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

est une tournée contenant le cycle 1; 2; 4; 1. Mais le problème principal est que la représentation adjacente ne supporte pas les opérateurs classiques de croisement. Pour pallier ce problème; un mécanisme de réparation est nécessaire.

2.1.2 La représentation ordinale

La représentation ordinale ([77]) représente le parcours du voyageur de commerce comme une liste de N villes dont le $i^{\text{ème}}$ élément est un nombre compris entre 1 et $(N - i + 1)$. L'idée de cette représentation est basée sur la permutation identité $C =$

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

qui sert de référence pour les autres listes et permet de les décoder. Par exemple, considérons le parcours codé par la représentation ordinale $l =$

1	1	2	1	4	1	3	1	1
---	---	---	---	---	---	---	---	---

et interprété en suivant ce raisonnement :

Étape	la liste de référence C et l	ville choisie																				
1	<table border="1"> <tr><td>C={</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9}</td></tr> <tr><td>l={</td><td>1</td><td>1</td><td>2</td><td>1</td><td>4</td><td>1</td><td>3</td><td>1</td><td>1}</td></tr> </table>	C={	1	2	3	4	5	6	7	8	9}	l={	1	1	2	1	4	1	3	1	1}	<p>1 est le premier élément de C</p> <p>$C \leftarrow C - \{1\}$ et $l \leftarrow l - \{1\}$</p>
C={	1	2	3	4	5	6	7	8	9}													
l={	1	1	2	1	4	1	3	1	1}													
2	<table border="1"> <tr><td>C={</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9}</td></tr> <tr><td>l={</td><td>1</td><td>2</td><td>1</td><td>4</td><td>1</td><td>3</td><td>1</td><td>1}</td></tr> </table>	C={	2	3	4	5	6	7	8	9}	l={	1	2	1	4	1	3	1	1}	<p>2 est le premier élément de C</p> <p>$C \leftarrow C - \{2\}$ et $l \leftarrow l - \{1\}$</p>		
C={	2	3	4	5	6	7	8	9}														
l={	1	2	1	4	1	3	1	1}														
3	<table border="1"> <tr><td>C={</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9}</td></tr> <tr><td>l={</td><td>2</td><td>1</td><td>4</td><td>1</td><td>3</td><td>1</td><td>1}</td></tr> </table>	C={	3	4	5	6	7	8	9}	l={	2	1	4	1	3	1	1}	<p>4 est le deuxième élément de C</p> <p>$C \leftarrow C - \{4\}$ et $l \leftarrow l - \{2\}$</p>				
C={	3	4	5	6	7	8	9}															
l={	2	1	4	1	3	1	1}															
4	<table border="1"> <tr><td>C={</td><td>3</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9}</td></tr> <tr><td>l={</td><td>1</td><td>4</td><td>1</td><td>3</td><td>1</td><td>1}</td></tr> </table>	C={	3	5	6	7	8	9}	l={	1	4	1	3	1	1}	<p>3 est le premier élément de C</p> <p>$C \leftarrow C - \{3\}$ et $l \leftarrow l - \{1\}$</p>						
C={	3	5	6	7	8	9}																
l={	1	4	1	3	1	1}																
5	<table border="1"> <tr><td>C={</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9}</td></tr> <tr><td>l={</td><td>4</td><td>1</td><td>3</td><td>1</td><td>1}</td></tr> </table>	C={	5	6	7	8	9}	l={	4	1	3	1	1}	<p>8 est le quatrième élément de C</p> <p>$C \leftarrow C - \{8\}$ et $l \leftarrow l - \{4\}$</p>								
C={	5	6	7	8	9}																	
l={	4	1	3	1	1}																	
6	<table border="1"> <tr><td>C={</td><td>5</td><td>6</td><td>7</td><td>9}</td></tr> <tr><td>l={</td><td>1</td><td>3</td><td>1</td><td>1}</td></tr> </table>	C={	5	6	7	9}	l={	1	3	1	1}	<p>5 est le premier élément de C</p> <p>$C \leftarrow C - \{5\}$ et $l \leftarrow l - \{1\}$</p>										
C={	5	6	7	9}																		
l={	1	3	1	1}																		
7	<table border="1"> <tr><td>C={</td><td>6</td><td>7</td><td>9}</td></tr> <tr><td>l={</td><td>3</td><td>1</td><td>1}</td></tr> </table>	C={	6	7	9}	l={	3	1	1}	<p>9 est le troisième élément de C</p> <p>$C \leftarrow C - \{9\}$ et $l \leftarrow l - \{3\}$</p>												
C={	6	7	9}																			
l={	3	1	1}																			
8	<table border="1"> <tr><td>C={</td><td>6</td><td>7}</td></tr> <tr><td>l={</td><td>1</td><td>1}</td></tr> </table>	C={	6	7}	l={	1	1}	<p>6 est le premier élément de C</p> <p>$C \leftarrow C - \{6\}$ et $l \leftarrow l - \{1\}$</p>														
C={	6	7}																				
l={	1	1}																				
9	<table border="1"> <tr><td>C={</td><td>7}</td></tr> <tr><td>l={</td><td>1}</td></tr> </table>	C={	7}	l={	1}	<p>7 est le premier élément de C</p>																
C={	7}																					
l={	1}																					

l est la liste codée du parcours que le commis voyageur doit suivre et pour qu'il lui soit compréhensible, la liste est décodée à l'aide de la liste de référence C . Le tableau précédent montre le processus de décodage. Chaque ligne indique l'ordre de visite de la ville que la liste l fournit son numéro d'emplacement dans la liste de référence. Aussitôt que la ville à visiter est déterminée, les deux listes C et l sont mises à jour, on supprime le premier élément de l et l'emplacement indiqué par celui-ci dans C .

Par exemple, 1 est le premier nombre de la liste l , qui fait référence au premier élément 1 de C puis on l'enlève de C qui devient (23456789) et l devient (12141311). Ainsi, la première ville à visiter est 1, ainsi de suite, jusqu'à la fin de la liste l et nous trouvons que le parcours du voyageur de commerce correspond à 1 ; 2 ; 4 ; 3 ; 8 ; 5 ; 9 ; 6 ; 7 ; 1.

Cette représentation demande aussi un algorithme de conversion entre les listes et les parcours réels, ce qui augmente les temps de calculs [49]. De plus, les mauvais résultats expérimentaux indiqués dans [49] montrent que cette représentation n'est pas adaptée en collaboration avec un opérateur de croisement classique.

2.1.3 Représentation chemin

Un circuit est codé par un tableau des entiers qui représente le successeur et le prédécesseur de chaque ville.

6	2	4	1	5	3	7
---	---	---	---	---	---	---

TAB. 2.2: Codage du chromosome.

Ainsi, le tableau (TAB.2.2) représente le circuit suivant : le point de départ est la ville 6 qui est aussi la ville d'arrivée, en passant par les villes selon leur ordre d'apparition dans le tableau : $6 \rightarrow 2 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow 3 \rightarrow 7 \rightarrow 6$.

2.2 Sélection

La méthode utilisée est la sélection par la roulette. Elle consiste à associer à un chromosome i de la population une portion proportionnelle à $\frac{1}{N-1} \times (1 - \frac{f_i}{\sum_{j \in Population} f_j})$ où f_i est la valeur de la fonction objectif de l'individu i .

Ainsi, Les individus qui ont les valeurs faibles de la fonction d'évaluation peuvent avoir une forte chance d'être acceptés et soient assujettis à l'opération de croisement.

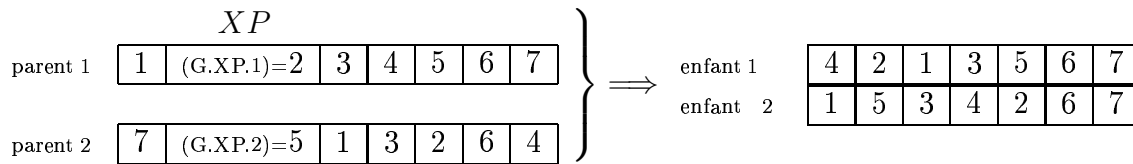
2.3 Opérateurs de croisement

Ils manipulent la structure des chromosomes de deux parents, qui sont choisis au hasard, afin de produire deux enfants. Ils permettent de découvrir l'espace des solutions. Dans la littérature, nous avons choisi huit opérateurs de croisement que nous allons expliquer leurs manières de procéder pour construire deux chromosomes (solutions) appelés *enfant 1 et enfant 2* à partir de deux chromosomes appelés *parent 1 et parent 2*.

2.3.1 Opérateur de croisement cyclique (cycle par Olivier et al. [84]) :

On note par (G.XP.1) le gène qui se trouve dans le parent 1 à la position XP et par (G.XP.2) le gène du parent 2 qui se situe à la position XP.

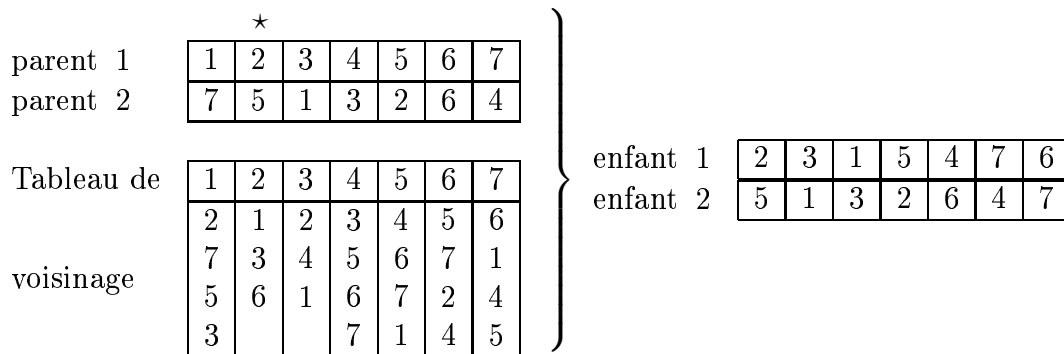
Une position (XP) est choisie au hasard dans parent 1, on y trouve le gène (G.XP.1) qu'on placera dans la position (XP) de l'enfant 1. Après, on placera les gènes du parent 2, à partir de la position (XP+1), dans l'enfant 1 jusqu'à ce qu'on trouve le gène (G.XP.1) dans le parent 2 dans une position (XP+k) alors (G.XP+k.1) est mis dans l'enfant 1 et on continue à remplir ce dernier à partir de son parent 1 en sautant les gènes qui ont été déjà introduits. Pour obtenir l'enfant 2, nous procédons de la même manière en commençant de la même position XP et en intervertissant les rôles du parent 1 et du parent 2.



TAB. 2.3: L'opérateur de croisement cyclique (cycle).

2.3.2 Opérateur de croisement de recombinaison des arcs (edrx par Whitley [105]) :

A partir de deux parents, on construit un tableau qui fait correspondre à chaque gène ses voisins adjacents. Par exemple, le gène 1 a pour voisins dans parent 1 les gènes 2 et 7, et dans le parent 2 les gènes 5 et 3.



TAB. 2.4: Opérateur de croisement de recombinaison des arcs (edrx).

Pour obtenir l'enfant 1, on choisit aléatoirement un gène parmi ceux du parent 1. Ce gène constituera le premier gène de l'enfant 1, son successeur sera sélectionné parmi ses voisins que l'on extrait du tableau de voisinage en respectant les règles suivantes :

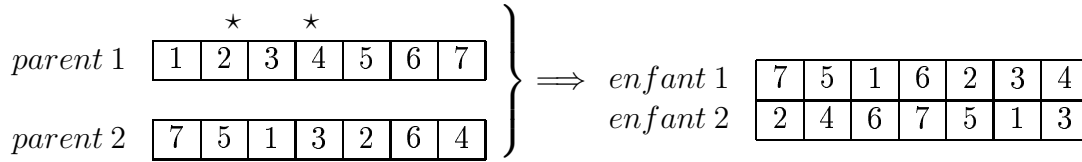
- Le gène qui sera élu est celui qui a le plus petit nombre de voisins distincts.
- Si on trouve plusieurs gènes qui vérifient la condition ci-dessus, alors on choisira aléatoirement parmi eux l'heureux élu.

Ayant le gène élu, on supprime aussitôt le gène qui le précède dans l'enfant 1 et on recommence jusqu'à ce que l'enfant 1 soit complètement construit.

2.3.3 Opérateur de croisement de préservation maximale (MPX par Mühlenbein et al.[81]) :

On choisit au hasard deux positions XP1 et XP2 et on cherche la position du gène (G.XP1.1) dans parent 2 que l'on note P1. Ensuite, on place la séquence des gènes du parent 1 qui se trouve entre XP1 et XP2 dans le enfant 1 à partir de la position P1. Le

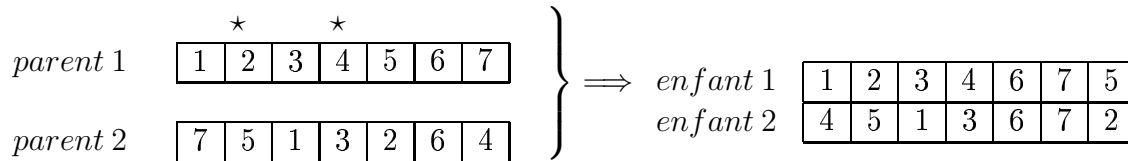
reste de l'enfant 1 est copié à partir du parent 2 en commençant par son premier gène en sautant celui qui a été déjà mis en place dans l'enfant 1.



TAB. 2.5: Opérateur de croisement de préservation maximale.

2.3.4 Opérateur de croisement d'ordre 1 (order 1 par Davis et al.[38]) :

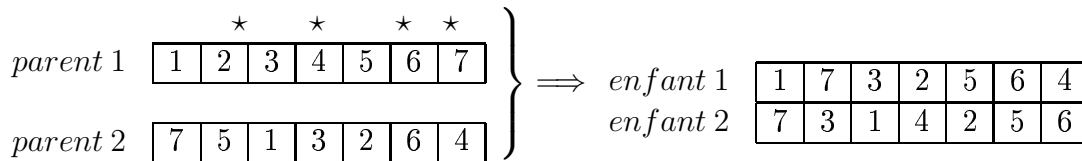
On choisit au hasard deux positions XP1 et XP2 du parent 1. La partie du milieu est copiée dans l'enfant 1. Le reste est rempli à partir du parent 2 en commençant par son gène XP2+1 et en sautant les éléments qui sont déjà présents dans l'enfant 1.



TAB. 2.6: Opérateur de croisement order1.

2.3.5 Opérateur de croisement d'ordre 2 (order 2 par Syswerda[99]) :

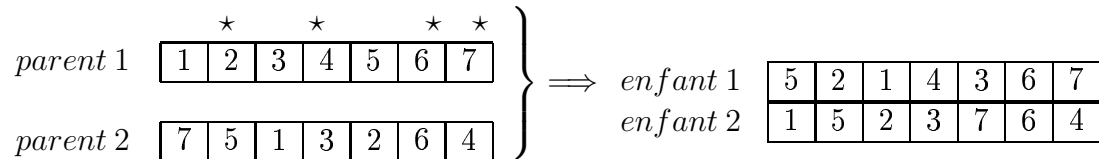
On remplit l'enfant 1 à partir du parent 1. On choisit quatre points au hasard que nous remplissons suivant leur ordre d'apparition dans parent 2. Pour remplir l'enfant 2, nous faisons la même chose mais en intervertissant les rôles de parent 1 et parent 2.



TAB. 2.7: Opérateur de croisement order2.

2.3.6 Opérateur de croisement de position (position par Syswerda [99]) :

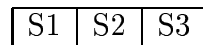
Pour construire l'enfant 1, on choisit quatre positions aléatoirement qu'on va recopier à partir du parent 1 et le reste de l'enfant 1 est copié à partir du parent 2 en commençant depuis le début et en sautant les gènes qui ont été déjà mises en place.



TAB. 2.8: Opérateur de croisement de position.

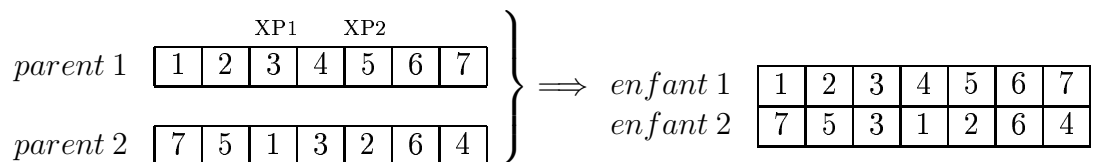
2.3.7 Opérateur de croisement assorti partiel (pmx par Goldberg et al.[51]) :

On choisit aléatoirement deux points de croisement XP1 et XP2 qui décomposent les deux parents en trois séquences chacun.



TAB. 2.9: La partition d'un parent.

les séquences S1 et S3 du parent 1 sont copiées dans l'enfant 1. La séquence S2 de l'enfant 1 est constituée par les gènes du parent 2 en commençant par le début de sa partie S2 et en sautant les gènes qui sont déjà mises en place.

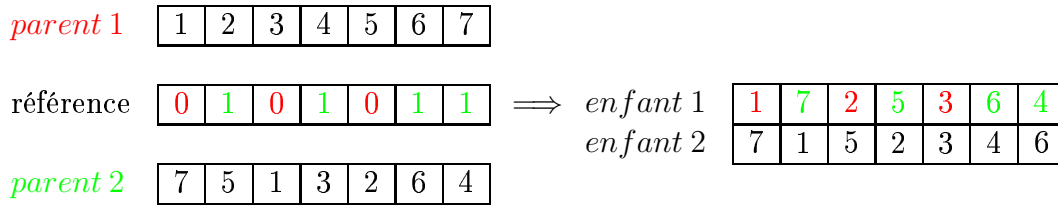


TAB. 2.10: Opérateur de croisement assorti partiel (pmx).

2.3.8 Opérateur de croisement uniforme (uox) :

L'enfant 1 est constitué en alternant aléatoirement entre les deux parents. En effet, nous générons selon la loi uniforme un tableau binaire (tableau de référence) de même longueur que les parents qui permet de choisir à chaque fois l'un des deux parents afin d'en prélever le gène qui sera introduit dans l'enfant. Ce gène sera supprimé des deux

parents pour s'interdire de créer un circuit non Hamiltonien et pour obtenir un enfant valide.



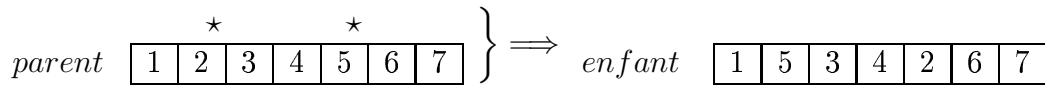
TAB. 2.11: Opérateur de croisement uox.

2.4 Opérateurs de mutation

Les opérateurs de mutation évitent d'établir des populations uniformes incapables d'évoluer. Ils permettent de modifier les valeurs des gènes de chromosomes. Nous définirons les cinq opérateurs de mutation suivants qui à l'encontre des opérateurs de croisement n'ont besoin que d'un seul parent pour produire un enfant unique.

2.4.1 Mutation twors :

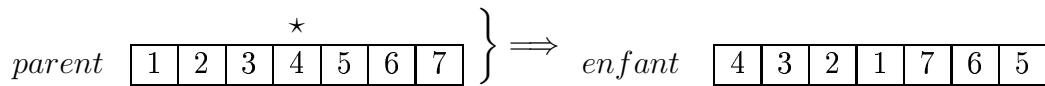
C'est l'échange de deux positions de gènes choisies aléatoirement.



TAB. 2.12: Opérateur de mutation twors (twors).

2.4.2 Mutation de centre inverse (cim) :

On choisit aléatoirement une position qui coupe le chromosome en deux séquences S_1 et S_2 . On prend la première séquence S_1 et on inverse l'ordre de ses gènes : le dernier devient premier, l'avant dernier devient second et ainsi de suite. on procède de la même manière sur S_2 .



TAB. 2.13: Opérateur de mutation centre inverse (icm).

2.4.3 Mutation inverse (im) :

On prend une séquence S limitée par deux positions $i < l$ choisies aléatoirement. L'ordre des gènes de cette séquence sera inversé de la même manière que dans le cadre de

S_1 considéré au paragraphe (2.4.2).

$$\text{parent } \left[\begin{array}{|c|c|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline \end{array} \right] \left. \begin{array}{l} \star \\ \star \end{array} \right\} \Rightarrow \text{enfant } \left[\begin{array}{|c|c|c|c|c|c|c|} \hline 1 & 5 & 4 & 3 & 2 & 6 & 7 \\ \hline \end{array} \right]$$

TAB. 2.14: Opérateur de mutation inverse (im).

2.4.4 Mutation throas :

On construit une séquence de trois gènes dont le premier est choisi au hasard et les deux autres ne sont que ses deux successeurs. et on procède de la manière suivante : le dernier gène devient premier de la séquence, le second passe en dernier et le premier devient le second.

$$\text{parent } \left[\begin{array}{|c|c|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline \end{array} \right] \left. \begin{array}{l} \star \longrightarrow \end{array} \right\} \Rightarrow \text{enfant } \left[\begin{array}{|c|c|c|c|c|c|c|} \hline 1 & 2 & 5 & 3 & 4 & 6 & 7 \\ \hline \end{array} \right]$$

TAB. 2.15: Opérateur de mutation throas (throas).

2.4.5 Mutation thrors :

On choisit trois gènes aléatoirement qui prennent les positions distinctes non nécessairement successives $i < j < l$ en procédant ainsi : le gène en position i passe en position j , celui qui est en position j passe en $l^{\text{ème}}$ position, puis le gène occupant la position l passe en $i^{\text{ème}}$ position.

$$\text{parent } \left[\begin{array}{|c|c|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline \end{array} \right] \left. \begin{array}{l} \star \\ \star \\ \star \end{array} \right\} \Rightarrow \text{enfant } \left[\begin{array}{|c|c|c|c|c|c|c|} \hline 1 & 6 & 3 & 2 & 5 & 4 & 7 \\ \hline \end{array} \right]$$

TAB. 2.16: Opérateur de mutation thrors (thrors).

2.5 Méthode d'insertion

Nous avons utilisé, la méthode d'insertion élitisme qui consiste à recopier le meilleur chromosome de l'ancienne population dans la nouvelle. Celle-ci est complétée par les solutions résultantes d'opérateurs de croisement et de mutation en veillant à ce que la taille de la population reste fixe de génération en génération.

2.6 Résultats numériques

2.6.1 Environnement

Les opérateurs de l'algorithme génétique et leurs différentes modalités, qui seront utilisées par la suite, sont regroupés dans ce tableau :

Opérateur de Croisement	Order1; Order2; position; cyclique; pmx; uox; MPX; edrx
Opérateur de Mutation	cim; throas; im; Thrors; twors
Probabilité de croisement	1; 0.8; 0.6; 0.4; 0.3; 0.2; 0.1; 0
Probabilité de mutation	0; 0.1; 0.2; 0.4; 0.5; 0.6; 0.8; 0.9

TAB. 2.17: Les différents niveaux des différents opérateurs.

Nous travaillerons sur les instances suivantes du problème de voyageur de commerce :

Instances	Taille	Valeur Optimale connue f_{opt}
Berlin52	52	7542
Eil101	101	629
Kroa200	200	29368

TAB. 2.18: Les instances utilisées [103].

Ces problèmes appartiennent à la famille des problèmes symétriques de T.S.P. Chacun est constitué de coordonnées de points dans un repère Euclidien. Nous ne changeons qu'un seul paramètre à la fois, fixons les autres et exécutons trente fois l'algorithme génétique. La programmation a été faite en langage C sur une machine PC Pentium de CPU 1.99Ghz ayant pour système d'exploitation Linux SUSE.

2.6.2 Analyse

Au début, nous avons fixé la taille de la population pour l'algorithme génétique à 104 pour les différentes expériences effectuées sur les trois problèmes de T.S.P.. Afin de déceler les paramètres ; que ce soit l'opérateur de croisement ou l'opérateur de mutation avec leurs probabilités respectives, qui sortent du lot et permettent d'obtenir les meilleurs résultats. Le critère qui permet de comparer entre ces réalisations était un problème en soi. Nous avons tout d'abord, utilisé l'analyse de la variance (ANOVA) pour voir s'il y avait une différence entre les moyennes de chaque groupe constitué de trentes exécutions. Après, Nous avons appliqué le test de Tukey pour départager les moyennes lorsqu'il y a rejet de l'hypothèse H_0 (correspondante au fait qu'il n'y a pas de différence entre les moyennes). Ensuite, nous avons travaillé avec les critères suivants :

- La somme d'écart moyen $SEM = \sum_{i=1}^{30} \frac{(f_i - f_{opt})}{30}$
- L'écart minimal

$$EM = \underset{i=1, \dots, 30}{Min} (f_i - f_{opt}).$$

Utiliser ces deux critères, pour comparer entre l'influence de chaque composante des différents compartiments de l'algorithme génétique, est devenu un problème d'optimisation multi-objectifs. En fin de compte, nous avons pris le critère de la racine carrée de l'erreur quadratique moyenne qu'on notera **RMSE** (Root Mean Square Error) comme

critère de comparaison :

$$RMSE = \sqrt{\sum_{i=1}^{30} \frac{(f_i - f_{opt})^2}{30}}. \quad (2.1)$$

Ce critère peut s'écrire aussi sous la forme suivante :

$$\sqrt{\sum_{i=1}^{30} \frac{(f_i - f_{mean})^2}{30} + (f_{mean} - f_{opt})^2}. \quad (2.2)$$

On voit évidemment que minimiser RMSE (2.2) c'est à la fois minimiser la variance des fitness et l'écart de la moyenne des fitness par rapport à la valeur optimale. Ce qui nous a conforté dans notre choix de ce critère.

2.6.3 Comparaison entre les opérateurs de croisement

Pour comparer les opérateurs de croisement entre eux en absence des opérateurs de mutations, nous avons pris les réalisations ayant la probabilité de croisement égale à 1 et la probabilité de mutation à 0. Les résultats obtenus par rapport à l'instance de Berlin 52 sont illustrés dans la figure (FIG.2.1).

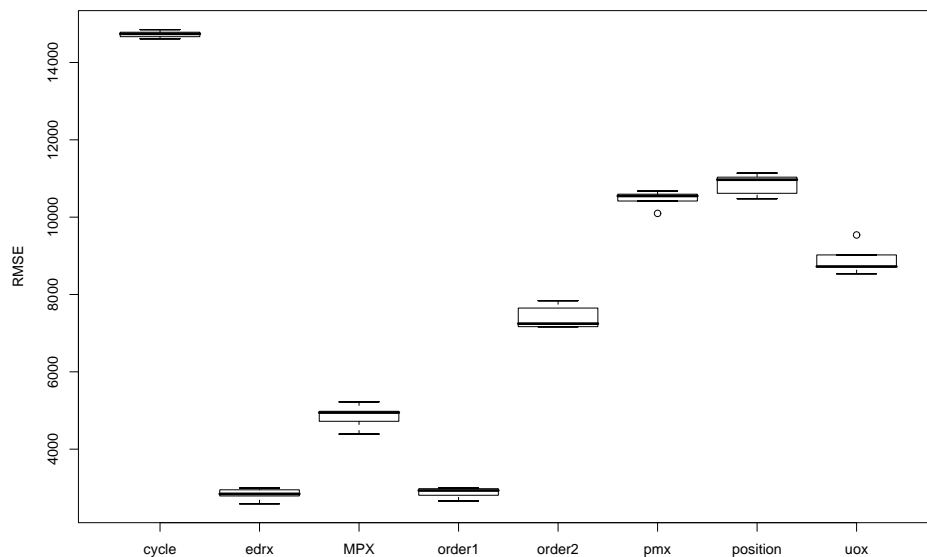


FIG. 2.1: Les boîtes à moustaches de RMSE en fonction des variations des opérateurs de croisement -Berlin52.

Une boîte à moustache se compose d'une boîte rectangulaire au milieu, dont les deux extrémités sont les quartiles (en bas 25% et en haut 75%). Ces extrémités se prolongent

par des traits terminés par des segments orthogonaux représentant les déciles (en bas 10% et en haut 90%). On représente aussi la médiane (correspondant à 50%) par un trait dans la boîte, et parfois les valeurs extrêmes (minimum en bas et maximum en haut) par des points. D'après la figure FIG.2.1, le meilleur opérateur de croisement est *edrx* suivi par *order1*, *MPX*, *order2*, *uox*, *pmx*, *position*, et en dernier l'opérateur *cyclique*. Par rapport

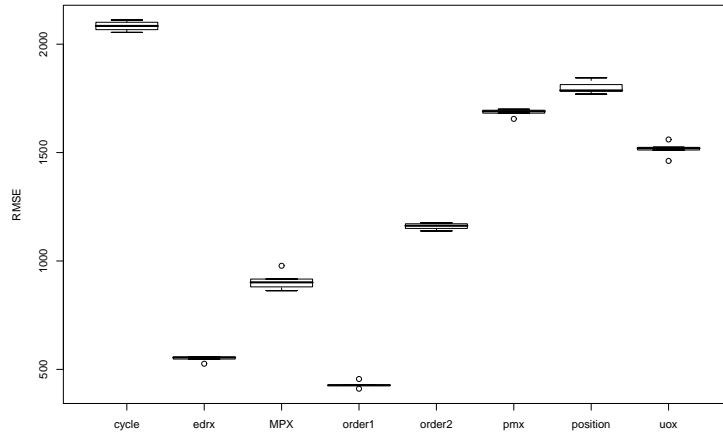


FIG. 2.2: Les boites à moustaches de RMSE en fonction des variations des opérateurs de croisement- eil101.

à l'instance eil101, Nous trouvons que l'opérateur *cyclique* est le moins efficace. Mais, le meilleur opérateur est devenu *order1* suivi par *edrx*. Les autres opérateurs ont gardés leurs places respectives.

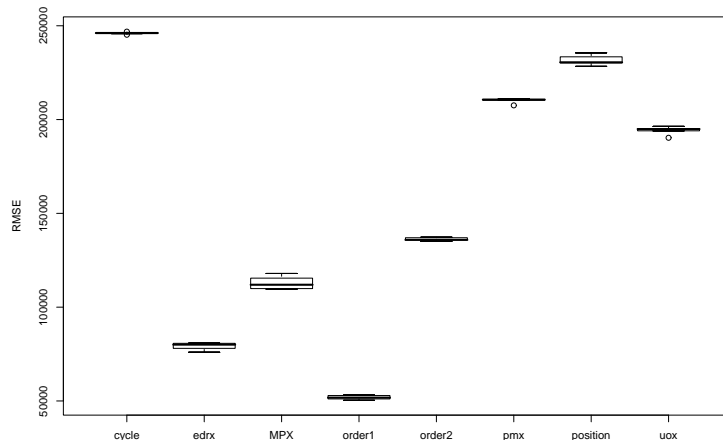


FIG. 2.3: Les boites à moustaches de RMSE en fonction des variations des opérateurs de croisement- kroA200.

Nous trouvons pour kroA200 le même ordre des opérateurs de croisement rencontré lors des résultats de eil101. A partir des trois instances, **order1** a un léger avantage sur **edrx**, celui-ci est talonné par **MPX** qui est pisté à son tour par **order2**. Ces opérateurs sont poursuivis par **uox**, **pmx**, **position**. En dernière position, on trouve l'opérateur **cyclique**.

En utilisant uniquement l'opérateur de croisement au sein de l'algorithme génétique, il vaut mieux utiliser l'opérateur **order1**. Il est plus stable que *edrx* et obtient les plus petites valeurs de la fonction objective. Donc, au niveau de l'exploration de l'ensemble des solutions, l'opérateur **order1** est le meilleur.

2.6.4 Comparaison entre les opérateurs de mutation

Pour comparer uniquement l'influence des opérateurs de mutation en absence de croisement, nous fixons la probabilité de croisement à 0 et la probabilité de mutation à 0.9.

Nous avons trouvé un ordre d'efficacité constant concernant ces opérateurs de mutations pour les trois instances de T.S.P. (FIG.2.4). Le meilleur opérateur de mutation est **im**, suivi par *cim*, qui dépasse de peu *twors*. Ces opérateurs sont talonnés par *thrors* et en dernière position on retrouve l'opérateur de mutation *throas*.

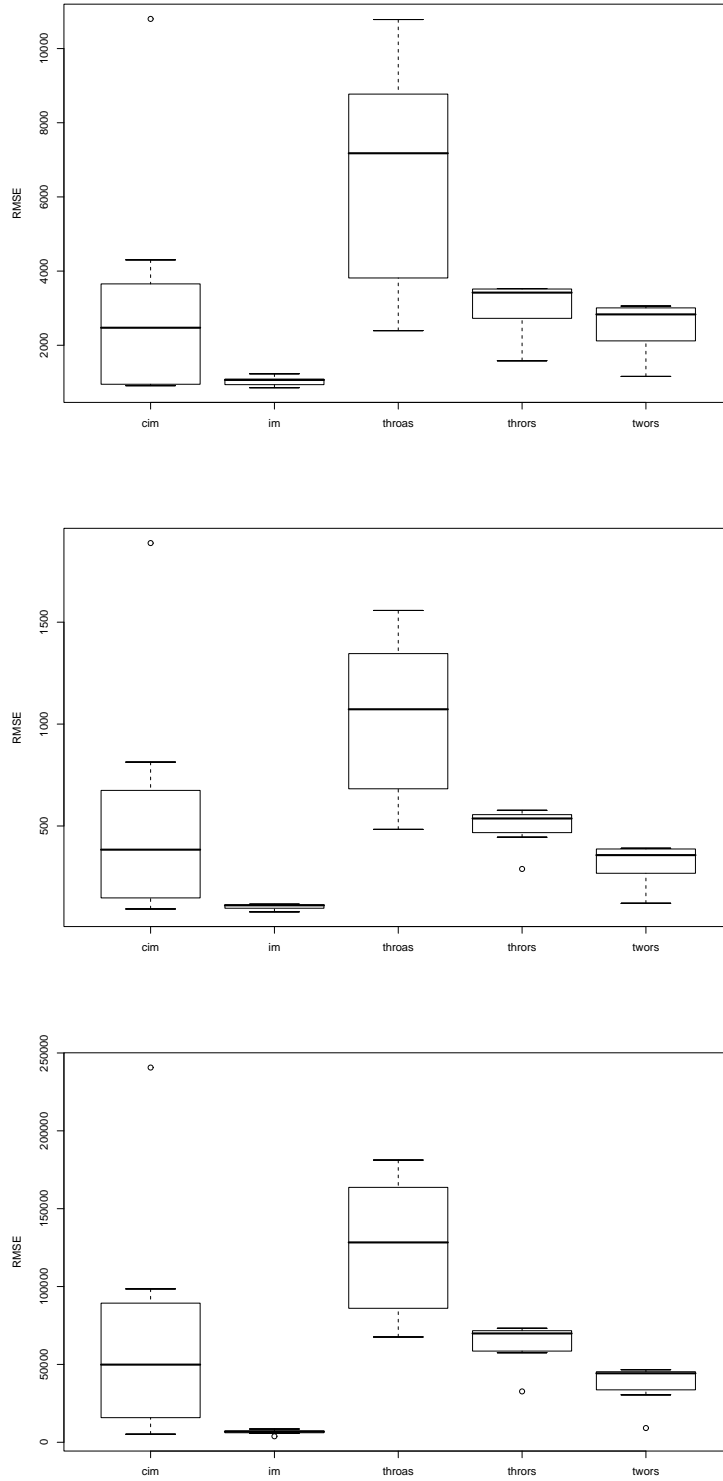


FIG. 2.4: Les boîtes à moustaches de RMSE en fonction des variations des opérateurs de mutations pour les instances Berlin52 (en haut), eil101 (au milieu) et kroA200 (en bas).

2.6.5 Comparaison entre les interactions des opérateurs de croisement et mutation

Parmi toutes les exécutions, nous avons sélectionné les dix premières valeurs de RMSE pour chaque instance dans le tableau suivant :

Ordre	Cross.	P _x	Mut.	P _m	RMSE
1	edrx	0.4	im	0.9	834.74
2	edrx	0	im	0.8	843.86
3	MPX	0	cim	0.5	844.68
4	MPX	0.2	im	0.9	845.59
5	edrx	0.7	im	0.9	846.82
6	edrx	0.4	im	0.8	850.31
7	edrx	0.6	im	0.9	854.76
8	edrx	0	im	0.9	857.01
9	edrx	0.2	im	0.9	857.37
10	edrx	1	im	0.9	867.89

berlin52

Ordre	Cross.	P _x	Mut.	P _m	RMSE
1	edrx	0	im	0.9	78.93
2	edrx	0	im	0.8	81.57
3	edrx	1	im	0.9	82.09
4	edrx	0.4	im	0.9	82.15
5	edrx	0.7	im	0.9	83.42
6	edrx	0.6	im	0.9	83.76
7	edrx	0.2	im	0.9	83.92
8	edrx	0.2	im	0.9	84.28
9	edrx	0.8	im	0.9	85.37
10	edrx	0.2	im	0.8	86.23

eil101

Ordre	Cross.	P _x	Mut.	P _m	RMSE
1	edrx	1	im	0.9	3682.68
2	edrx	0	im	0.9	3836.76
3	edrx	0.4	im	0.9	3919.81
4	edrx	0.2	im	0.9	4035.31
5	edrx	0.7	im	0.9	4090.82
6	edrx	0.8	im	0.9	4147.89
7	edrx	0.2	im	0.8	4203.11
8	edrx	0.6	im	0.9	4286.19
9	edrx	0	im	0.8	4357.87
10	edrx	1	im	0.8	4417.76

kroA200

TAB. 2.19: Les dix meilleurs modalités des opérateurs

Nous observons que l'opérateur **im** avec sa probabilité de mutation égale à **0.9** et l'opérateur de croisement **edrx** obtiennent les plus petites valeurs de RMSE. Par contre, la probabilité de croisement varie entre trois valeurs 0, 0.4 et 1 corresspondantes respectivement aux instances *eil101*, *berlin52* et *kroA200*.

2.6.6 Discussion sur la taille de la population

Jusqu'à maintenant, nous avons utilisé une taille fixée de la population et trouvé trois stratégies :

- edrx, $P_x = 1$, im et $P_m = 0.9$ que nous allons la nommer St1 ;
- edrx, $P_x = 0.4$, im et $P_m = 0.9$ nommée St2 ;

– St3 est composée de **edrx**, $P_x = 0$, **im** et $P_m = 0.9$.

Nous faisons varier la taille de la population $N \in \{V, 2 * V, 3 * V, \dots, 10 * V\}$, où V est la taille du problème, afin de comparer et de départager les trois stratégies et en même temps d'obtenir une taille optimale. À partir de la figure (FIG.2.5) de berlin52, nous observons que la stratégie St3 donne la plus petite valeur de RMSE pour une taille de population égale à $10 \times V$, suivie par la stratégie S1 pour la même taille. Cette dernière stratégie, donne les plus petites valeurs de RMSE pour le problème eil101 à partir de la taille $8 \times V$ et pour kroA200 à partir de $6 \times V$.

La stratégie St1, correspond à **edrx** comme opérateur de croisement appliqué à tous les éléments de la population ($P_x = 1$) et comme opérateur de mutation **im** avec $P_m = 0.9$, est la meilleure stratégie avec une taille de population égale à $8 \times V$.

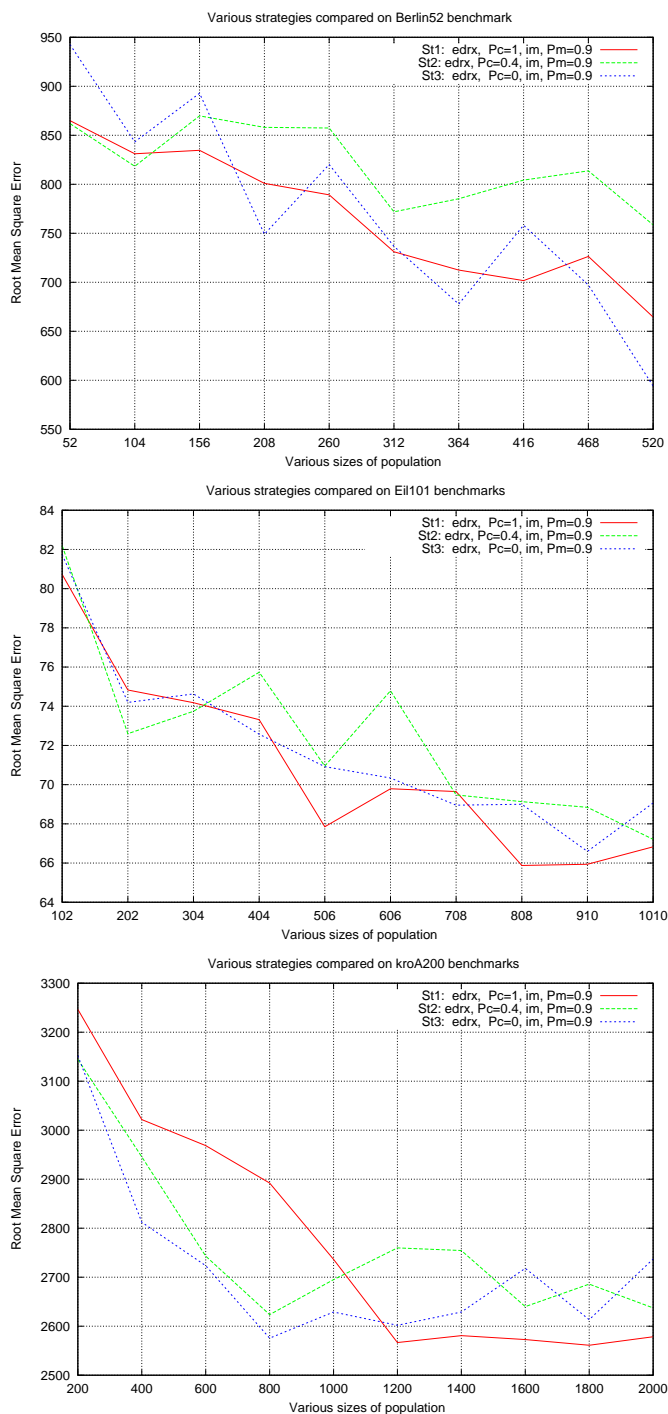


FIG. 2.5: La variation du RMSE en fonction du changement de la taille de la population pour les instances berlin52, eil101 et kroA200.

2.7 Conclusion

Nous avons, dans ce chapitre, utilisé l'algorithme génétique générationnel illustré dans la figure (FIG.1.1) avec la méthode de sélection la roulette et la méthode d'insertion basé sur l'élitisme. Nous avons employé huit opérateurs de croisement et cinq opérateurs de mutation trouvés dans la littérature et nous nous sommes servis de huit valeurs de probabilités de croisement et autant de mutation.

Nous avons trouvé les résultats suivants à partir des données recueillies sur les trois instances :

- avec une taille fixe de la population et en regardant l'influence de croisement en absence de mutation, l'opérateur **order1** est meilleur. L'opérateur cyclique arrive en dernière position.
- en considérant la mutation sans presque de croisement, nous décelons que le meilleur opérateur de mutation est **im**.
- en traitant les opérateurs de croisement et mutation ensemble avec leurs probabilités, nous avons obtenu ces trois stratégies :
 - St1={edrx, $P_x = 1$, im et $P_m = 0.9$ };
 - St2={edrx, $P_x = 0.4$, im et $P_m = 0.9$ };
 - St3 ={edrx, $P_x = 0$, im et $P_m = 0.9$ }.
- en examinant l'influence de la variation de la taille de population N , nous remarquons que lorsque N croît le critère RMSE décroît. Mais à partir de $6 \times V$, les valeurs de RMSE de chaque stratégies restent presque stables en moyenne.

Chapitre 3

Comparaison entre les méthodes de sélection appliquées dans l'AG

Sommaire

3.1	Introduction	65
3.2	Composantes de l'algorithme	65
3.3	Méthode de sélection	66
3.4	Résultats	68

3.1 Introduction

Le problème qui nous intéresse, est de connaître l'influence du choix de la méthode de sélection sur la convergence de l'algorithme génétique générationnel. Nous utiliserons six méthodes de sélection et deux instances correspondantes au problème du voyageur de commerce symétrique. Nous présenterons dans un premier temps l'algorithme génétique générationnel ainsi que les méthodes de sélection utilisées. Nous adaptons ces méthodes au problème de minimisation (T.S.P.) et nous étudions chacune d'elles afin de déceler leurs comportements et leurs influences sur la convergence de la variante de l'algorithme génétique. À la fin, nous comparons ces méthodes et nous présenterons celle qui se met en exergue et qui nous paraît la meilleure pour la résolution du T.S.P..

3.2 Composantes de l'algorithme

Nous utiliserons l' algorithme génétique générationnel caractérisé par :

- La population est constituée de $2 \times V$ solutions générées aléatoirement.
- Chaque chromosome représente un circuit Hamiltonien, solution du problème de T.S.P. de taille V .
- Opérateur de croisement (Edge recombination crossover - EDRX-) appliqué avec $P_x = 1$.
- Opérateur de mutation (Inverse mutation -im-) appliqué avec $P_m = 0.9$.

- La méthode d'insertion est l'élitisme.
 - Nous utiliserons deux instances de T.S.P. [103] :
 - Berlin 52 son optimum f_{opt} égal à 7542
 - Eil101 d'optimum égal à 629
- l'algorithme s'arrête lorsque le nombre d'itération maximum (NMax = 500) est atteint .

Comme critère de comparaison, nous restons fidèle au critère du racine de l'erreur quadratique moyenne (RMSE) afin de comparer l'influence des différentes méthodes de sélection et les départager.

3.3 Méthode de sélection

Nous adaptons les six méthodes de sélection au problème de minimisation :

Méthode du tournoi : C'est une méthode basée sur le fait de prendre la meilleure solution dans un ensemble de cardinal égale à $[t \times N]$ dont les éléments sont prélevés aléatoirement de la population actuelle.

```

Pop(i) population à l'itération i de taille N;
Pop(i)={I1, I2, ..., IN} ;
t ∈ ]0,1] paramètre prédéfini
Pour i de 1 à N faire
    | Ii le meilleur individu choisi parmi [t * N] choisi au hasard dans Pop(i).
Fin Pour
Return { I'1, I'2, ..., I'N } ;
    
```

Où $[x]$ désigne la partie entière de la variable x.

Méthode du troncation : Tout d'abord, on ordonne la population actuelle selon l'ordre décroissant de la valeur de la fonction objectif. Ensuite un individu situé dans la population à partir d'un certain rang, est choisi aléatoirement.

```

Pop(i);
t ∈ [0,1] paramètre fixé à priori;

Pop(i) est ordonné de telle manière que le mauvais chromosome prend la
première position;
Pour i de 1 à N faire
    r = random ∈ {[ (1-t)*N], ..., N};
    I'_i = I_r;
Fin Pour
Return { I'_1, I'_2, ..., I'_N };

```

Méthode de la roulette : Elle consiste à associer à chaque chromosome i la probabilité :

$$P_i = \frac{(1 - \frac{f_i}{\sum_{j \in Pop} f_j})}{N - 1}$$

où f_i est la valeur de la fonction objectif pour l'individu i .

Méthode d'ordre linéaire (linear ranking) : Dans cette méthode, on ordonne la population selon l'ordre décroissant de la fonction objectif. Cette méthode dépend d'un paramètre fixé, noté η^- . Il permet de définir la probabilité de prélever de la population le plus mauvais individu. Elle est égale à $\frac{\eta^-}{N}$.

```

Pop(i);
 $\eta^- \in [0, 1]$  paramètre fixé à priori;

Pop(i) est ordonné tel que le mauvais chromosome prend la première position;
 $S_1 = 0$ ;
Pour i de 1 à N faire
  |  $P_i = \eta^- + 2 * \frac{(1-\eta^-)*(i-1)}{(N-1)}$ ;
Fin Pour
Pour i de 2 à N faire
  |  $S_i = S_{i-1} + P_{i-1}$ ;
Fin Pour
Pour i de 1 à N faire
  |  $r = random \in [0, S_N[$ ;
  |  $I'_i = I_i$  tel que  $S_i \leq r < S_{i+1}$ ;
Fin Pour
Return  $I'_1, I'_2, \dots, I'_N$ ;

```

Méthode d'ordre exponentiel : Cette méthode est caractérisée par le paramètre de contrôle $t \in]0,1[$. La méthode procède de la même manière que la méthode ordre linéaire avec :

$$P_i = \frac{t^{(N-i)}}{\sum_{j=1}^N t^{(N-j)}}$$

Méthode de Boltzmann : Cette méthode procède de la même manière que la méthode d'ordre exponentiel en y remplaçant le paramètre de contrôle t par e dans P_i .

3.4 Résultats

Pour chaque instance, nous générons aléatoirement au départ une population de taille $N=2*V$ qui sera la population initiale pour les 30 exécutions effectuées pour chaque méthode de sélection.

Pour les méthodes ordre exponentiel, ordre linéaire et troncation, le paramètre de contrôle prend respectivement les valeurs 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 et 1.

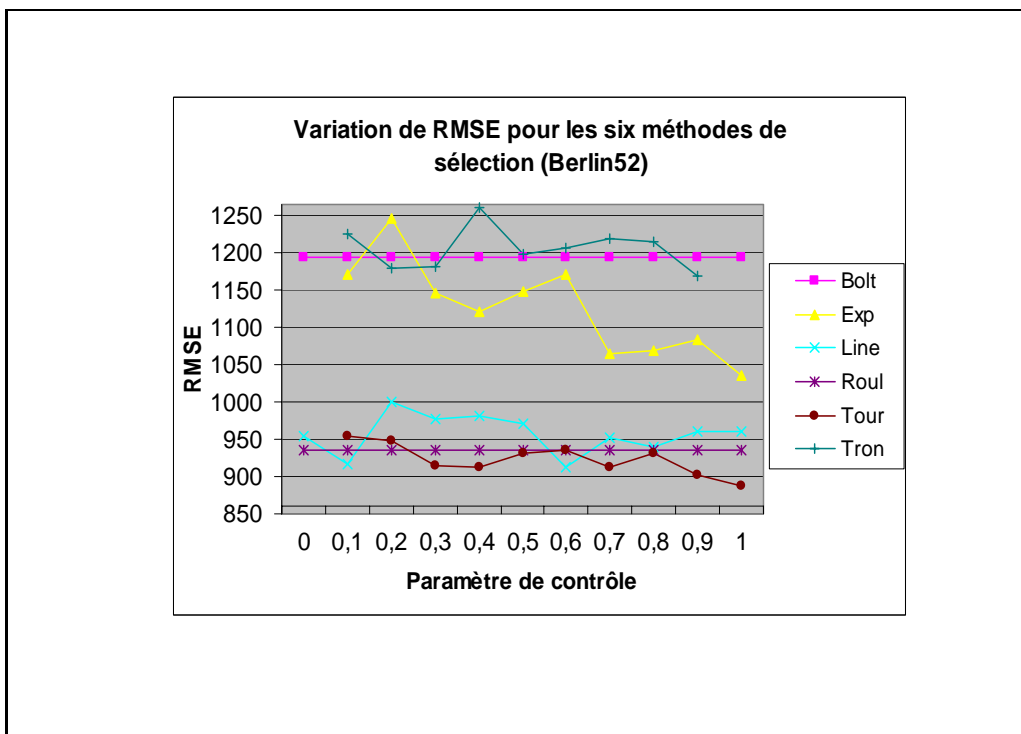


FIG. 3.1: Comparaison entre les méthodes de sélection en utilisant RMSE -Berlin52.

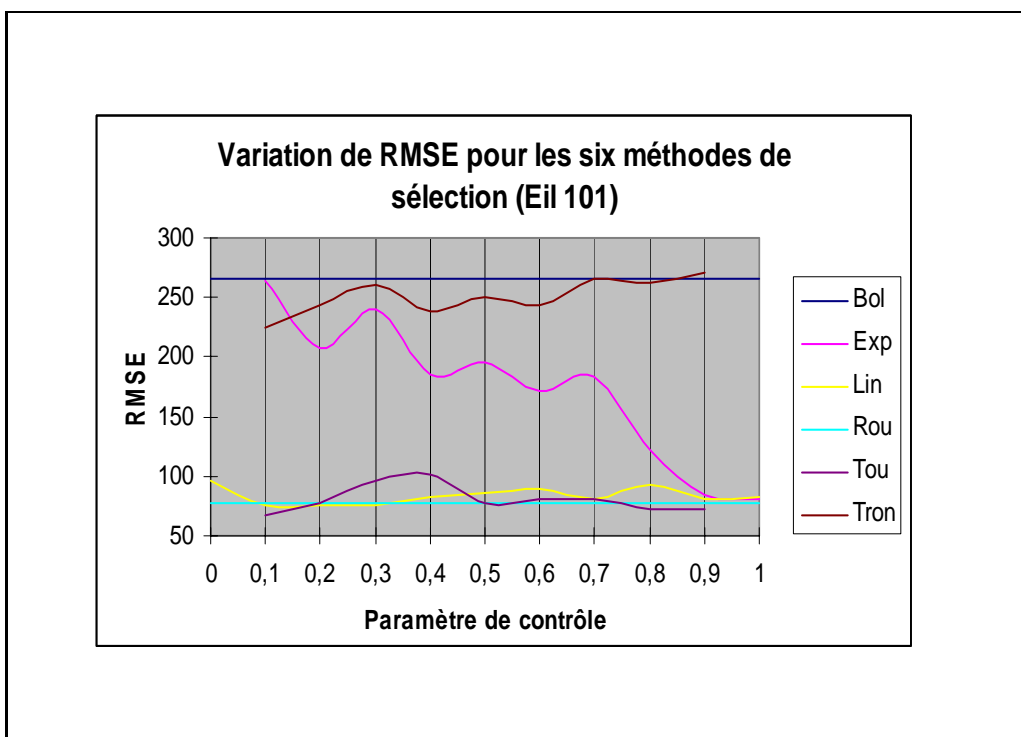


FIG. 3.2: Comparaison entre les méthodes de sélection en utilisant RMSE -Eil101.

En se focalisant sur la figure 3.1, nous remarquons qu'il y a deux groupes :

- Le 1^{er} regroupe les méthodes de Boltzmann et de troncation qui donnent des valeurs très grandes de RMSE.
- Le 2^{ème} se constitue des méthodes de la roulette, ordre linéaire et tournoi qui obtiennent des valeurs petites de RMSE.
- La méthode d'ordre exponentiel se place au début avec le 1^{er} groupe et quand son paramètre de contrôle s'approche de 1, ses valeurs de RMSE s'approche du 2^{ème} groupe.
- Les mêmes remarques se confirment en appliquant les méthodes de sélection sur le problème de Eil 101 (3.2).

La méthode de sélection du tournoi avec $t = 0.8$ ou 0.9 sort du lot suivie par la méthode "ordre linéaire" avec $\eta^- = 0.6$. Toutefois, il faut signaler que la méthode de sélection de la roulette a donné des résultats satisfaisants par rapport aux précédentes méthodes.

Chapitre 4

Génération de la population initiale concernant le problème T.S.P.

Sommaire

4.1	Introduction	71
4.2	Implémentation de l'algorithme génétique	72
4.2.1	Génération de la population initiale	72
4.3	Résultats numériques	75
4.3.1	Instance Berlin 52	75
4.3.2	Instance Eil 101	78
4.3.3	Instance de Kroa200	80
4.3.4	Instance a280	83
4.4	Conclusion	85

4.1 Introduction

Dans ce chapitre, nous étudierons l'influence de la manière de générer la population initiale sur la convergence de l'algorithme génétique. Nous exposerons six méthodes de génération de la population que nous les appliquerons sur quatre instances de T.S.P. À la fin, nous dresserons les résultats des comparaisons effectuées.

4.2 Implémentation de l'algorithme génétique

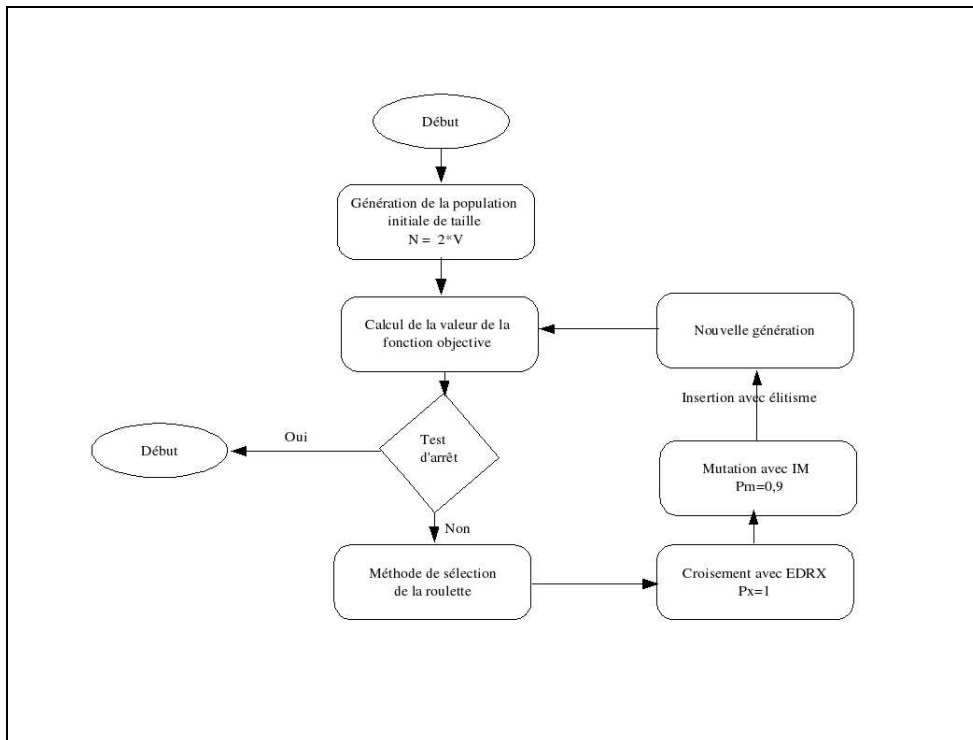


FIG. 4.1: L'organigramme de notre algorithme génétique.

Notre algorithme, présenté dans la figure (FIG.4.1), débute par générer une population initiale de N individus ($N = 2 \times V$). Pour lesquels, nous calculons leurs valeurs de la fonction objectif et nous sélectionnons les individus les mieux adaptés en se basant sur le principe de la roulette. Les individus, sujets de croisement par l'opérateur de croisement edrx (paragraphe 2.3.2), sont choisis selon une probabilité $P_x = 1$. Leurs résultats peuvent être mutés par l'opérateur de mutation inverse (paragraphe 2.4.3) avec une probabilité de mutation $P_m = 0.9$. Les individus issus de ces opérateurs génétiques seront insérés dans la nouvelle population et nous évaluons les valeurs de la fonction objectif de chaque individu. Après, un test d'arrêt sera effectué, qui consiste à examiner si la variance des valeurs de la fonction d'évaluation est nulle. Si ce test est vérifié alors l'algorithme s'arrête avec une solution optimale, sinon on réitère le processus.

Nous employerons la représentation chemin (paragraphe 2.1.3) pour le codage de nos chromosomes, méthode de sélection par la roulette (paragraphe 2.2) et pour la méthode d'insertion l'élitisme (paragraphe 2.5).

4.2.1 Génération de la population initiale

La population initiale conditionne fortement la rapidité de l'algorithme. Pour cela, nous avons appliqué plusieurs méthodes de génération de la population initiale :

Méthode 1 : Génération aléatoire de la population initiale. Nous générons chaque individu de la manière suivante :

```

Fonction Méthode 1(Pop : Tableau de V colonnes et 2*V lignes) : Pop
| individu : tableau d'entiers de dimension V ;
| Pour j de 1 à V faire
| | individu[j] ← (-1)
| Fin Pour
| Pour j de 1 à V faire
| | rechercher aléatoirement un indice pour lequel individu[indice]=-1 ;
| | individu[indice] ← j ;
| Fin Pour
| copier individu dans Pop[1];
| Pour j de 1 à 2*V-1 faire
| | Muter individu ;
| | copier individu dans Pop[j] ;
| | copier Pop[1] dans individu ;
| Fin Pour
Fin

```

Algorithme 6: Méthode 1 pour la génération de la population initiale

Méthode 2 : Génération du premier individu au hasard, celui-ci sera muté $(2 \times V - 1)$ fois avec un opérateur de mutation.

Méthode 3 : Nous utilisons au début un algorithme génétique qui se base sur une génération aléatoire de la population initiale. L'individu optimal obtenu et ses $(2 \times V - 1)$ mutations constituent la population initiale.

Méthode 4 : Génération du premier individu en utilisant un mécanisme heuristique. Le successeur de la première ville est celui qui se trouve à une distance plus petite par rapport aux autres. Ce mécanisme permet de construire un parcours qui prend en compte que la distance entre deux villes successives soit la plus petite possible. C'est à dire que le successeur d'une ville au milieu du circuit est le plus proche en terme de distance par rapport aux villes non encore visitées. Ce parcours malheureusement n'est pas la solution optimale mais c'est une borne supérieure. Après, nous utilisons un opérateur de mutation sur le parcours obtenu afin de générer $(2 \times V - 1)$ autres individus qui constitueront la population initiale.

```

V= nombre de villes;
tab1 : tableau des réels de dimension V;
individu : tableau des entiers de dimension V;
min : variable réel;
l,i,idx,k : variables entiers;

i=1;
idx=1;
individu[idx] ← i;
Tant que ( idx ≤ V ) faire
    Pour k de 1 à V faire
        | tab1[k] ← distance entre la ville i et la ville k
    Fin Pour
    Pour k de 1 à idx faire
        | m ← individu[k];
        | tab1[m] ← (-1);
    Fin Pour
    recherche de i l'indice de la valeur minimale du tab1 parmi ses valeurs
    ≠ -1;
    idx ← idx+1;
    individu[idx] ← i;
Fait

```

Algorithme 7: Mécanisme heuristique

Méthode 5 : Une génération aléatoire d'un individu solution du problème du T.S.P., celui-ci est le premier élément (individu1) de la population initiale. Pour compléter la population, nous utilisons le processus suivant :

```

V : nombre de villes;
individu1 : tableau des entiers de dimension V ;
individu2 : tableau des entiers de dimension V ;
copier individu1 dans individu2;
f1 ← la longueur du circuit représentée par individu2;
f3 ← 0.0;
compteur ← 0;
Tant que ((f3 < 0.02 )et(compteur<= 50)) faire
    copier individu2 dans individu1;
    Muter individu1;
    f2 ← la longueur du circuit représentée par individu1;
    f3 ← (f1 - f2)/f1;
    compteur ← compteur +1;
Fait

```

Algorithme 8: Processus de création de nouveaux individus de la population initiale par la méthode 5

Nous essayons, au plus 50 fois, à faire diminuer la valeur de la fonction objectif de 2% en utilisant l'opérateur de mutation im. Si le processus échoue à trouver un tel individu, le dernier individu de la boucle est introduit dans la population initiale jusqu'à avoir $(2 \times V)$ éléments.

Méthode 6 : Cette méthode procède de la même manière que la méthode 5 sauf que le premier individu est créé par le mécanisme heuristique utilisé dans la méthode 4, qui prend en compte la distance la plus courte entre deux villes successives.

Dans la suite, nous allons comparer l'influence de chacune de ces six méthodes de génération de la population initiale sur notre algorithme.

4.3 Résultats numériques

Nous allons exécuter le programme pour chaque instance une dizaine de fois. Nous travaillerons sur les quatre instances suivantes, prélevées du site d'internet TSPLIB [103] :

1. berlin52 de 52 locations dans Berlin dont la borne supérieure est égale à 7542.
2. eil101 de 101 villes, de borne supérieure égale à 629.
3. kroA200 de 200 villes, de borne supérieure égale à 29368.
4. a280 de 280 emplacements ayant pour borne supérieure est égale à 2579.

4.3.1 Instance Berlin 52

Les résultats obtenus dans le tableau 4.1 vont nous aider à faire une étude comparative entre les six méthodes précédentes selon les valeurs de la fonction objectif.

	<i>Min</i>	médiane	<i>moyenne</i>	<i>max</i>	écart type
Méthode1	7879	8324	8376	8786	300.802
Méthode2	8024	8562	8472	8805	283.4594
Méthode3	7879	8406	8299	8560	247.7045
Méthode4	7805	8235	8212	8390	163.5615
Méthode5	8265	8395	8454	8781	169.0157
Méthode6	8011	8319	8274	8463	154.6315

TAB. 4.1: Les variations des valeurs de la fonction objectif par rapport aux différentes méthodes.

Nous remarquons que la méthode 4 permet d’obtenir la plus petite valeur avec un écart type de 163.56. Elle est suivie par la méthode 3 avec une valeur minimale de 7879 équivalente à la méthode 1 mais au niveau d’écart type la méthode 3 est meilleure. Après, nous trouvons en quatrième position la méthode 6 suivie par la méthode 2 et en dernière place la méthode 5 de génération de la population initiale. Ces remarques sont bien visualisées dans la figure suivante.

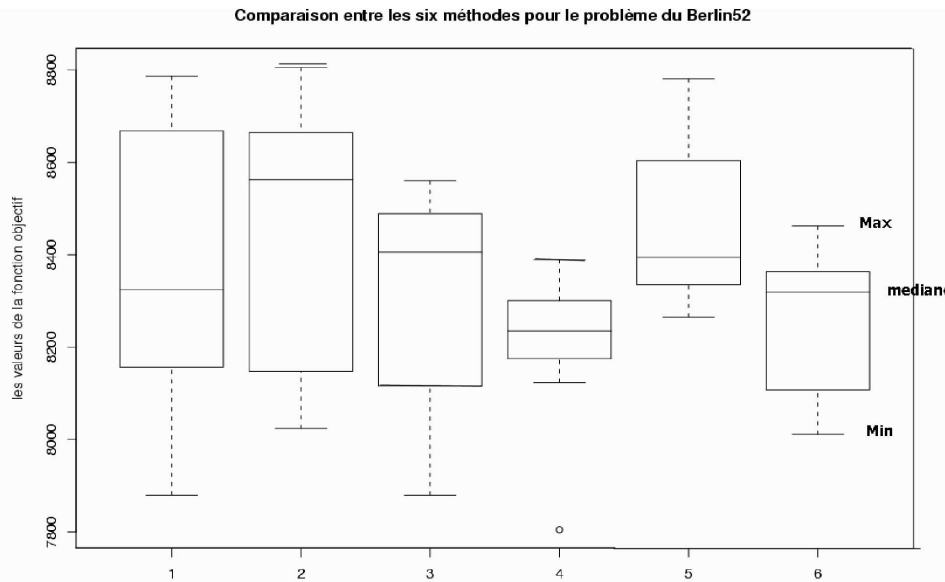


FIG. 4.2: Les boîtes à moustaches des valeurs de la fonction objectif en fonction de six méthodes.

Ainsi, nous remarquons que la méthode 4 a une boîte à moustache très petite qui reflète le fait que cette méthode trouve des valeurs très concentrées et que son troisième quartile

est inférieur aux médianes des autres méthodes c.à.d. que sur quatre réalisations, une seule peut donner une valeur plus mauvaise que celle de deux mauvaises réalisations de chacune d'autres méthodes.

Et si nous comparons par rapport au médiane de ces six méthodes, nous trouvons dans l'ordre d'efficacité : méthode 4 ; méthode 6 ; méthode1 ; méthode5 ; méthode3 ; méthode2.

Et par rapport à la valeur du troisième quartile 75% :

méthode 4 ; méthode 6 ; méthode3 ; méthode5 ; méthode2 ; méthode1.

En prenant en compte le temps d'exécution comme moyen de comparaison, nous obtiendrons le tableau suivant :

Méthode	<i>min</i>	<i>moyenne</i>	<i>max</i>
	des temps d'exécutions		
1	0.0	0.6	1.0
2	0.0	0.6	1.0
3	0.0	0.7	1.0
4	0.0	0.4	1.0
5	1.0	1.2	2.0
6	0.0	0.3	1.0

TAB. 4.2: Les variations des temps d'exécution par rapport aux différents méthodes.

Nous remarquons que la méthode 5 est la plus lente avec une durée moyenne plus grande que les autres d'une seconde. Par contre, la méthode 6 est plus rapide que toutes les autres, suivie par la méthode 4.

Les résultats concernant le nombre d'itérations sont enregistrés dans le tableau (TAB.4.3).

Méthode	<i>Min</i>	<i>moyenne</i>	<i>max</i>
	des nombres d'itérations		
1	426	542	693
2	396	488	672
3	305	4795	10020
4	203	264	332
5	321	435	568
6	135	233	322

TAB. 4.3: Les variations de nombre d'itérations par rapport aux différentes méthodes.

Ainsi, la méthode 6 s'exécute en un nombre d'itérations plus petit que les autres méthodes.

4.3.2 Instance Eil 101

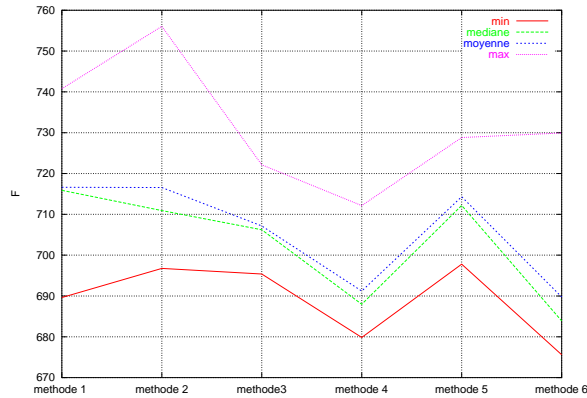


FIG. 4.3: Comparaison des six méthodes en fonction des valeurs de la fonction objectif pour eil101.

La méthode 6 donne la plus petite valeur de la fonction objectif, suivie par la quatrième méthode pour cette instance. En prenant en compte la valeur moyenne de la fonction objectif, cet ordre est préservé mais par rapport à la valeur maximum des différentes exécutions la méthode 4 est meilleure et nous trouvons que les deux méthodes 3 et 5 s'introduisent devant la méthode 6.

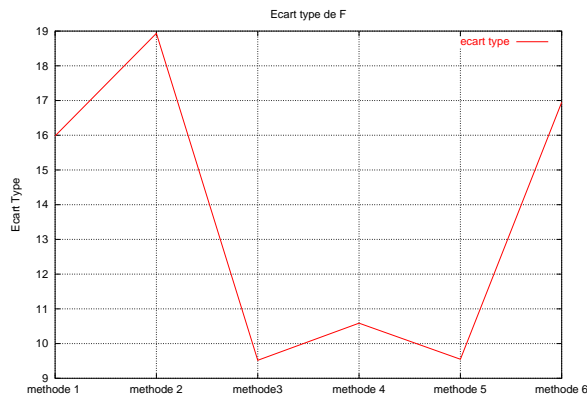


FIG. 4.4: Écart type des valeurs de la fonction objectif pour les six méthodes pour eil101.

Par rapport à l'écart type des 10 exécutions pour chacune des méthodes, nous trouvons en première position la méthode 3 suivie du 5. Après c'est la méthode 4 qui figure suivie des méthodes 1 et 6. La méthode 3 demande en moyenne un nombre exorbitant d'itérations par rapport aux autres. Par contre, les méthodes 4 et 6 ont des nombres d'itérations ne dépassant pas en moyenne les 600 itérations.

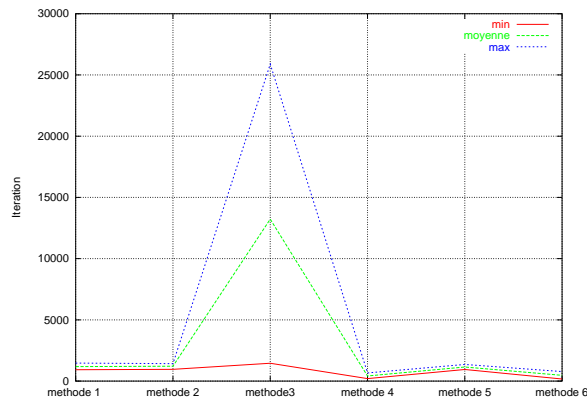


FIG. 4.5: Nombre d'itérations des six méthodes pour eil 101.

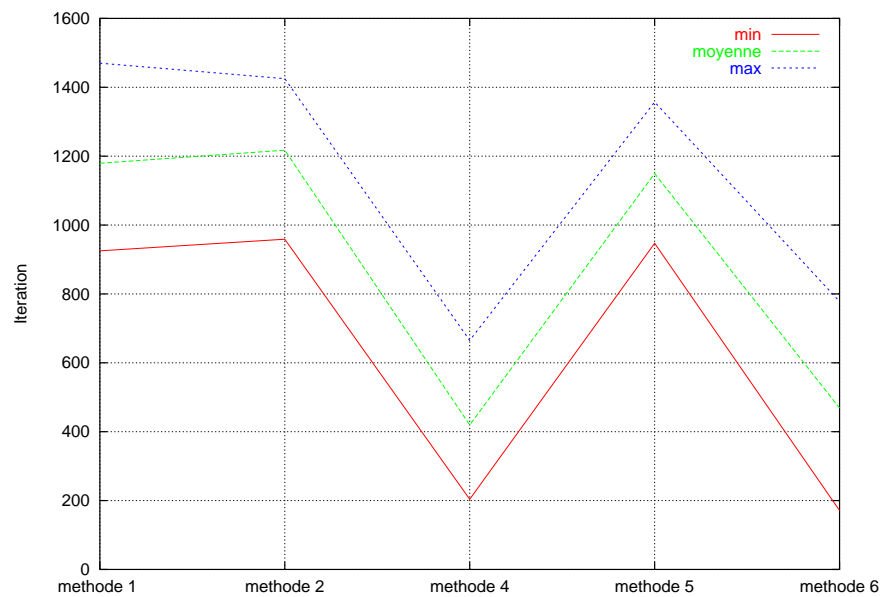


FIG. 4.6: Nombre d'itérations sans la méthode 3 pour eil101.

Nous remarquons que la méthode 3 dans (Fig.4.5) s'exécute en un nombre exorbitant d'itérations dépassant en moyenne 10000. En se référant à la figure (Fig.4.6), la méthode 4 demande en moyenne le plus petit nombre d'itérations, suivie par la sixième, la cinquième, la première et la deuxième méthode.

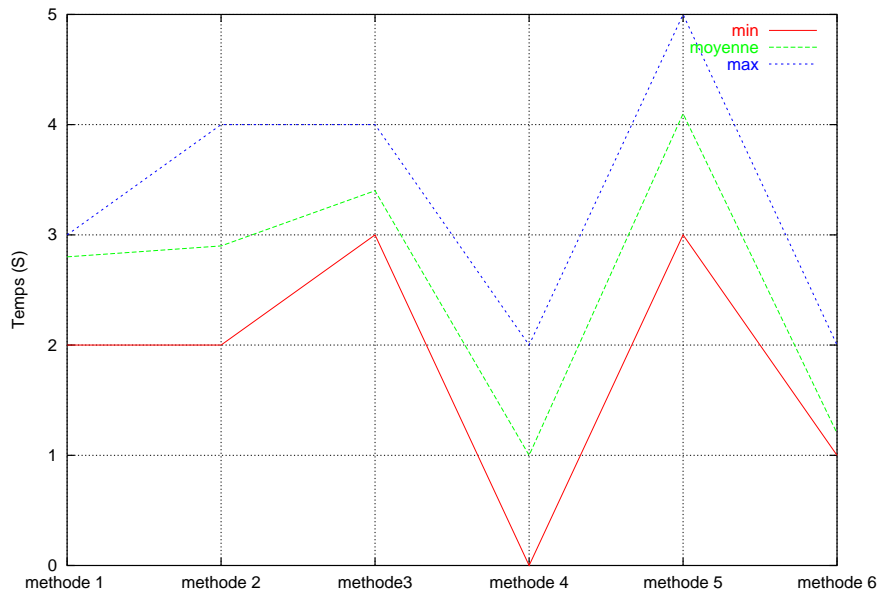


FIG. 4.7: Temps d'exécution des six méthodes pour eil101.

En prenant en compte la durée moyenne d'exécution des six méthodes, la quatrième est meilleure suivie par la sixième. En effet, la méthode 4 s'effectue en une durée moyenne égale à une seconde.

4.3.3 Instance de Kroa200

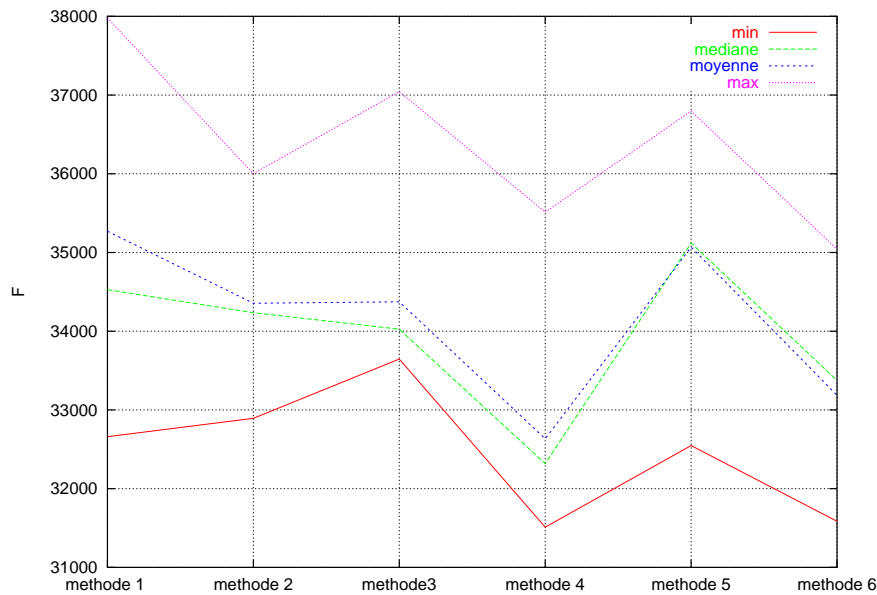


FIG. 4.8: Comparaison des six méthodes en fonction des valeurs de la fonction objectif pour Kroa200.

En moyen, la méthode 4 obtient des valeurs inférieures à celles de la méthode 6 (Fig.4.8). Ensuite, nous trouvons la méthode 2 suivie par la méthode 3, la méthode 5 et en dernière position la méthode 1. Les méthodes 2, 3, 5, et 1 arrivent en moyen après les méthodes 4 et 5.

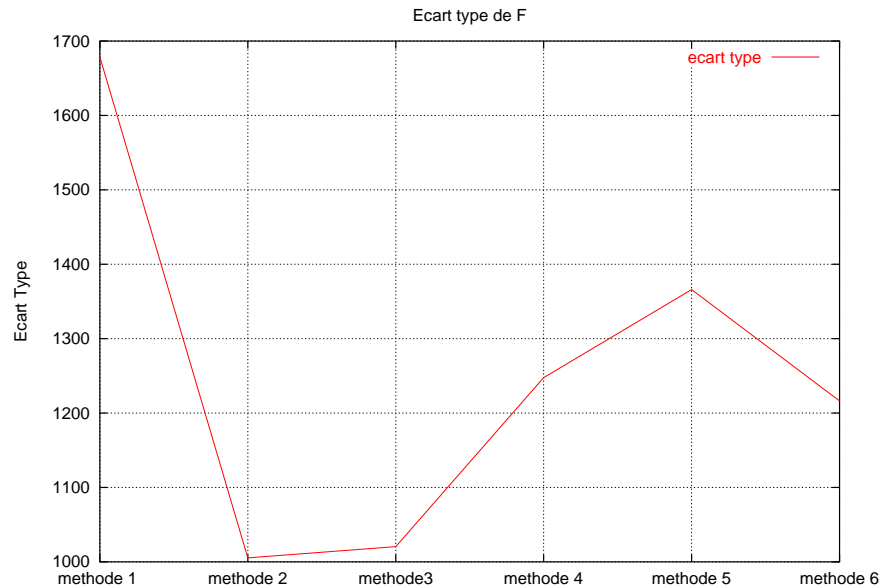


FIG. 4.9: Écart type des valeurs de la fonction objectif du Kroa200 pour les six méthodes.

L'écart type des valeurs de la fonction objectif des dix exécutions de la méthode 1 dépasse la barre de 1600 et c'est le plus grand par rapport aux différentes méthodes (Fig.4.9). Alors que la méthode 4 donne un écart type plus petit que celui obtenu par la méthode 6. Les dix exécutions de la Méthode 2 donnent les résultats les plus stables avec le plus petit écart type.

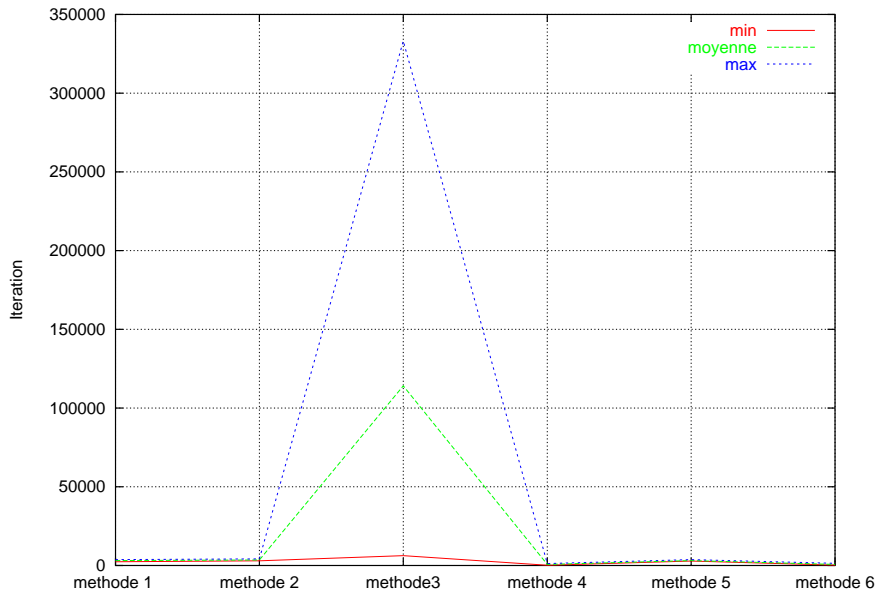


FIG. 4.10: Comparaison des six méthodes en fonction de nombre d'itérations pour Kroa200.

En se référant au moyenne de nombre d'itérations (Fig.4.10), la méthode 4 s'exécute en un plus grand nombre d'itérations dépassant les 10 000 et aussi une durée moyenne plus grande que les autres méthodes (Fig.4.11).

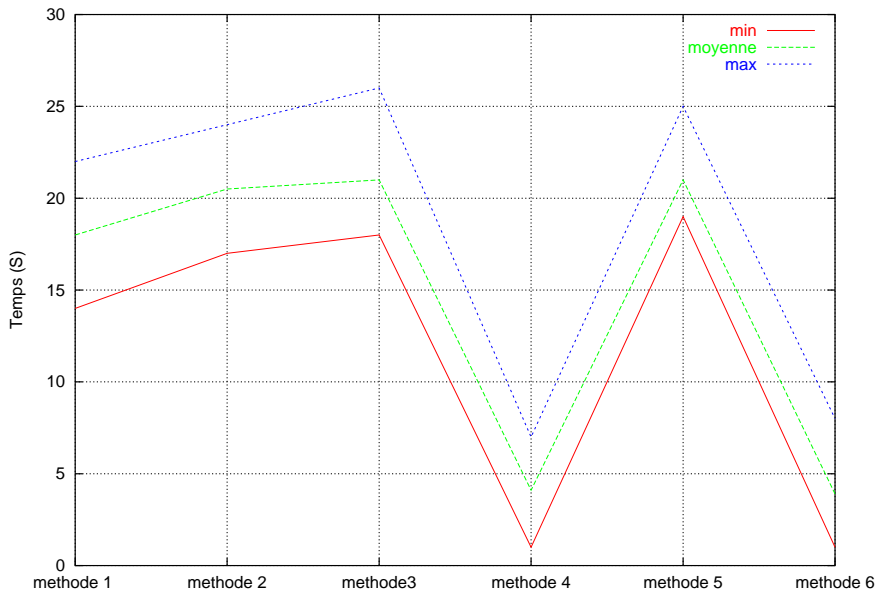


FIG. 4.11: Temps d'exécution des six méthodes pour Kroa200.

En absence des deux méthodes 4 et 6, la méthode 1 dure un temps moyen relativement petit par rapport à la deuxième, la troisième et la cinquième méthode.

4.3.4 Instance a280

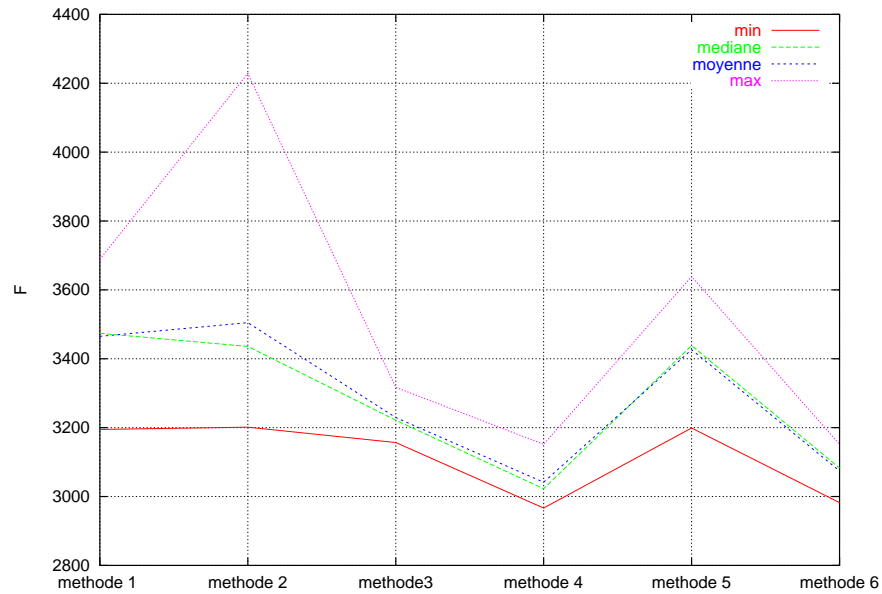


FIG. 4.12: Comparaison des six méthodes en fonction des valeurs de la fonction objectif de l'instance a280.

La méthode 4 permet d'obtenir les plus petites valeurs de la fonction objectif, suivie par la sixième méthode. Après, nous trouvons les méthodes 3, 5 et 1 qui surclassent la méthode 2. Le gain apporté par la méthode 4 par rapport à la méthode 3 dépasse 6.7%.

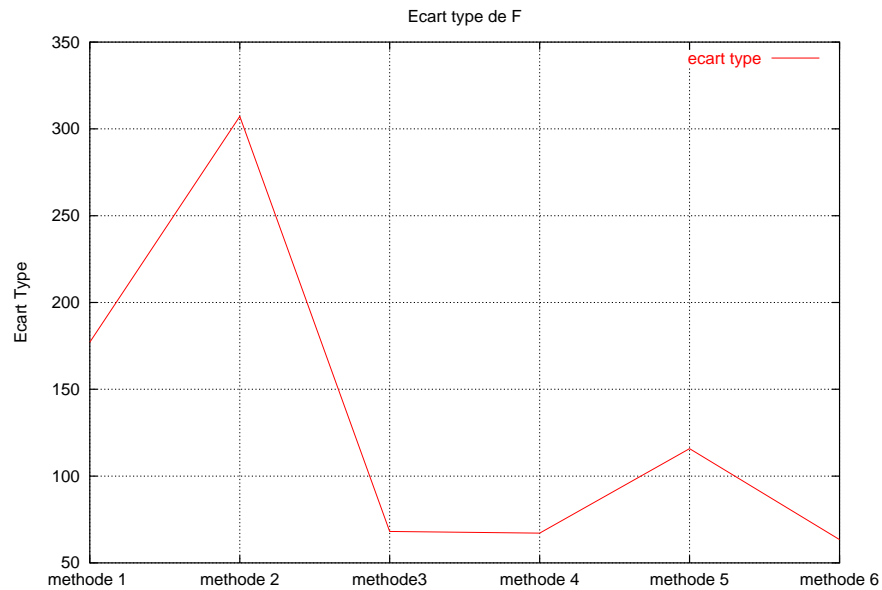


FIG. 4.13: Écart type des valeurs de la fonction objectif de a280 pour les six méthodes.

L'écart type de dix exécutions de chacune des méthodes 3, 4 et 6 sont presque égales et inférieures aux écart types de la méthode 5, 1 et méthode 2. Pour les problèmes de tailles supérieures à 200, la méthode 3 est aussi stable que les méthodes 4 et 6 (Fig.4.13).

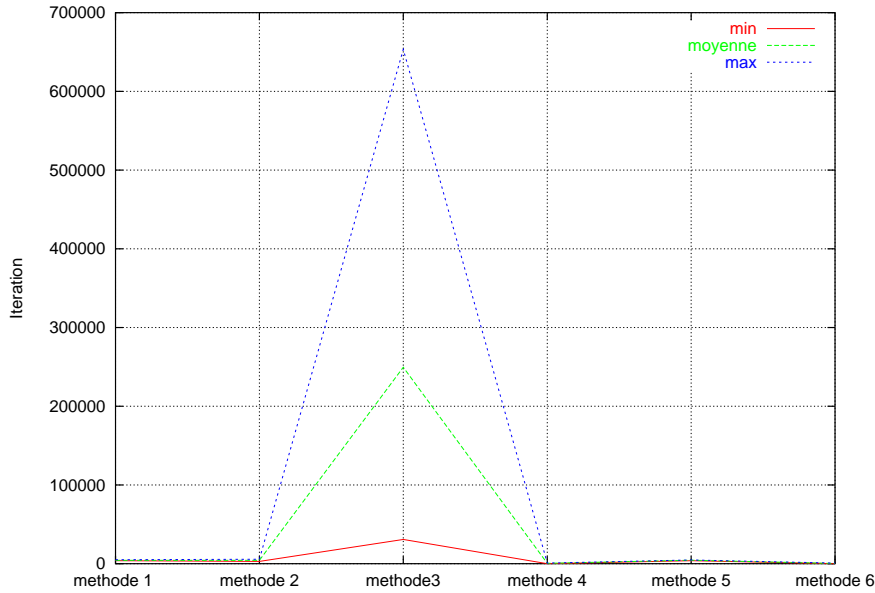


FIG. 4.14: Comparaison des six méthodes en fonction de nombre d'itérations pour a280.

Pour le problème de a280, la méthode 3 s'exécute en un nombre d'itérations dépassant les 25 000 (Fig.4.14) mais dure en moyenne moins longtemps que la méthode 2 (Fig.4.15).

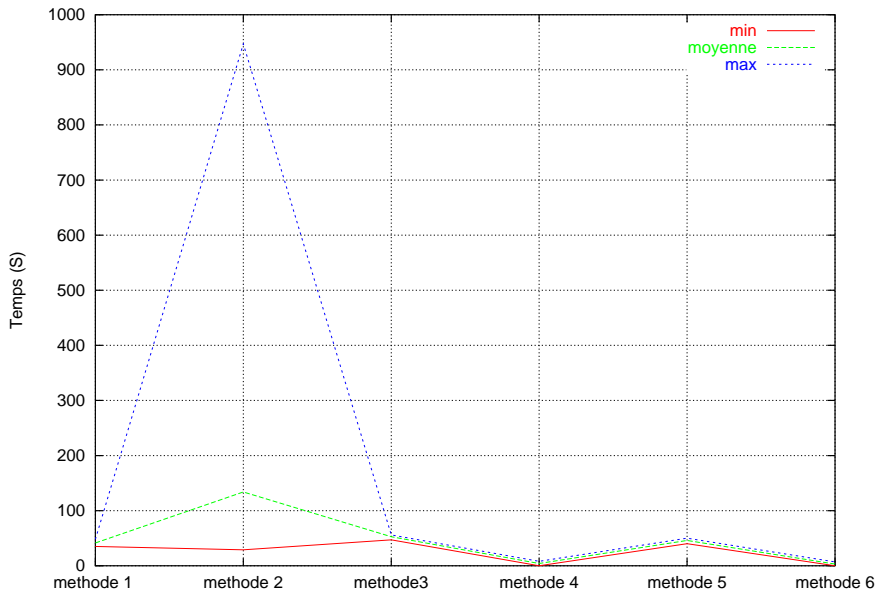


FIG. 4.15: Temps d'exécution des six méthodes pour a280.

4.4 Conclusion

En générant la population initiale par la méthode 4, nous observons ce qui suit :

- Les distributions des valeurs de la fonction objectif sont asymétriques par rapport à leurs médianes ; les parties les plus étendues des boîtes à moustaches correspondent aux valeurs supérieures à la médiane pour les problèmes Eil 101, Kroa200 et a280. Alors que la distribution pour Berlin52 est symétrique par rapport à la médiane avec le plus petit écart type par rapport aux autres méthodes.
- Les plus petites valeurs de la fonction objectif sont obtenues par cette méthode pour les problèmes berlin52, kroa200 et a280.
- Les plus petites durées moyennes et le nombre moyenne d'itérations sont obtenus avec cette méthode pour les problèmes eil101, kroa200 et a280.

Nous obtenons en générant la population initiale par la méthode 6, les résultats suivants :

- Les distributions des valeurs de la fonction objectif sont asymétriques pour les problèmes berlin52 et a280 avec des écarts types très petits et les parties les plus étendues correspondent aux valeurs inférieures aux médianes. La distribution est symétrique pour kroa200 et asymétrique avec un écart type très grand pour le problème de eil101 ; très étendue pour les valeurs supérieures à la médiane.
- La plus petite valeur de la fonction objectif pour le problème de eil101 et la deuxième plus petite valeur pour les problèmes de kroa200 et a280.
- Les résultats, en fonction du nombre moyenne d'itérations et de la durée moyenne d'exécution de l'algorithme génétique en se basant sur cette méthode de génération de la population initiale, la classent en deuxième position par rapport aux problèmes eil101, kroa200 et a280 et première pour berlin52.

La génération de la population initiale par la méthode 3 permet d'obtenir les résultats suivants :

- Les distributions sont asymétriques dont les parties les plus étendues correspondent aux valeurs de la fonction objectif plus grandes que les médianes pour les problèmes eil101, kroa200, et a280. Alors que sa distribution est asymétrique pour le problème de berlin52 avec la partie la plus large correspond aux valeurs inférieures à la médiane.
- La deuxième plus petite valeur ex aequo avec celle de la méthode 1 pour le problème berlin52 et troisième pour le problème de a280.
- Par rapport aux durées moyennes d'exécutions, et le nombre moyenne d'itérations. Les résultats la classent avant dernière et dernière position respectivement.

La génération de la population par la méthode 1, nous donne les résultats suivants :

- La distribution est asymétrique par rapport à la médiane pour le problème berlin52, étendue par rapport aux valeurs de la fonction objectif supérieure au médiane. Alors qu'elles sont symétriques pour par rapport aux médianes pour les autres problèmes.
- La troisième plus petite valeur de la fonction objectif pour le problème eil101 et quatrième pour kroa200 et a280.
- Elle se classe en troisième position par rapport aux durées moyennes d'exécution

et quatrième en nombre moyenne d'itérations pour les problèmes eil101, krao200et a280.

De ce qui précède, et en prenant en compte tous les paramètres, nous remarquons que la méthode 4 est la meilleure, suivie par la méthode 6 tant qu'il y a moyen d'utiliser le mécanisme heuristique de plus courte distance entre deux voisins dans la génération de la population initiale. Si nous n'utilisons pas ce mécanisme, la méthode 3 s'avère plus concurrente à la méthode 1 qui se base sur la génération aléatoire de la population initiale.

Chapitre 5

Notre variante de l'algorithme génétique

Sommaire

5.1	Introduction	87
5.2	Codage de chromosome	87
5.3	Génération de la population initiale	88
5.4	Opérateurs de croisement	89
5.4.1	Opérateur de croisement de recombinaison des arcs (Whit- ley 1989)	89
5.4.2	Notre opérateur de croisement Cedrx	90
5.5	Opérateurs de mutation	90
5.6	Implémentation de l'algorithme génétique	90
5.7	Résultats numériques	92

5.1 Introduction

Ce chapitre profite des avantages trouvés dans les chapitres précédents. Notamment, les études comparatives entre les méthodes de génération de la population initiale, les opérateurs de croisement et de mutation.

5.2 Codage de chromosome

Nous avons codé un circuit par un tableau des entiers qui représente le successeur et le prédécesseur de chaque ville en laissant fixe le premier élément pour tous les chromosomes. Par défaut, le premier gène de chaque chromosome sera fixé à 1.

1	2	3
---	---	---

TAB. 5.1: Codage du chromosome.

Ainsi, le tableau TAB.5.1 représente les circuits suivants :

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \text{ ou } 2 \rightarrow 3 \rightarrow 1 \rightarrow 2 \text{ ou bien } 3 \rightarrow 1 \rightarrow 2 \rightarrow 3.$$

La fixation du premier élément permet d'avoir un codage qui représente une application bijective entre l'ensemble de solutions et l'ensemble de recherche et aussi de minimiser le nombre de redondances dans la même population.

5.3 Génération de la population initiale

La taille de la population a été fixée à $2*V$ où V est la taille du problème T.S.P.. Nous utilisons deux méthodes pour générer deux parties de la population initiale, chacune est de taille V .

La première méthode, expliquée dans l'algorithme 7, est la méthode heuristique qui se base sur la recherche du successeur à chaque fois parmi les voisins les plus proches non encore visités. La deuxième méthode crée V individus par l'algorithme 5.3. Pour trouver un successeur, nous choisissons celui le plus proche parmi deux prélevés aléatoirement qui ne sont pas encore parcourus.

```

V= nombre de villes ;
tab1 : tableau des réels de dimension V ;
individu : tableau des entiers de dimension V ;
d1, d2 : variable réel ;
l,m,i,idx,k : variables entiers ;

i=1 ;
idx=1 ;
individu[idx] ← 1 ;
tab1[1]=-1 ;
Tant que ( idx ≤ V ) faire
    m ← individu[idx] ;
    tab1[m] ← ( -1 ) ;
    recherche deux gènes l et k tel que
    tab1[m] ≠ -1 et tab1[l] ≠ -1
    calculer d1 distance entre m et l ;
    calculer d2 distance entre m et k ;
    Si ( d1 < d2 ) Alors
        idx ← idx+1 ;
        individu[idx] ← m ;
    Sinon
        idx ← idx+1 ;
        individu[idx] ← l ;
    Fin Si
Fait

```

5.4 Opérateurs de croisement

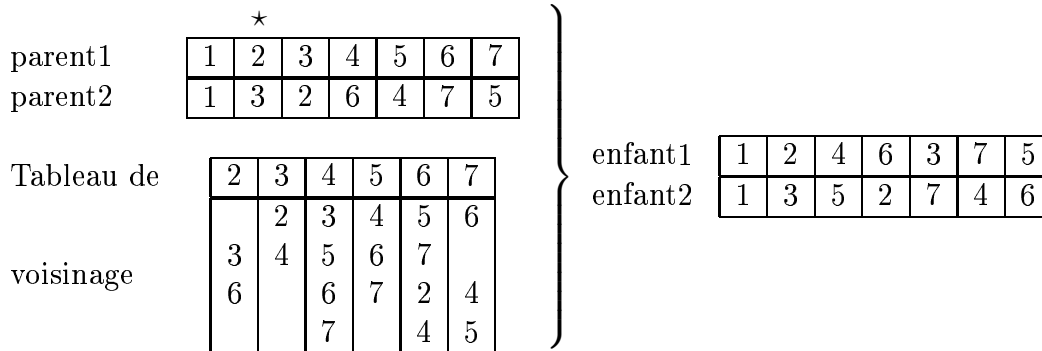
Ils manipulent la structure des chromosomes de deux parents, choisis au hasard, afin de produire deux enfants. Il permet de découvrir l'espace des solutions. Nous utilisons en même temps deux opérateurs de croisement.

5.4.1 Opérateur de croisement de recombinaison des arcs (Whitley 1989)

C'est l'opérateur qui s'est mis en exergue lorsque nous avons comparé les opérateurs de croisement (2.3.2). Son utilisation permet de préserver les arêtes les plus fréquentées dans les parents.

5.4.2 Notre opérateur de croisement Cedrx

L'application forcée et répétitive d'opérateur de croisement *edrx* permet à un certain moment que l'algorithme génétique stagne et converge dans la plupart du temps vers un minimum local. Pour remédier ce phénomène, nous avons créé cet opérateur qui casse les arêtes les plus fréquentées chez les parents. Dans le cas où les voisins d'un gène X, avec les gènes constituant l'enfant, regroupent l'ensemble de tout les gènes. Alors pour choisir le successeur de X, nous sélectionnons aléatoirement un gène parmi ceux qui n'ont pas été placés dans l'enfant.



TAB. 5.2: Notre opérateur de croisement (Cedrx).

5.5 Opérateurs de mutation

Il évite d'établir des populations uniformes incapables d'évoluer. Il consiste à modifier les valeurs des gènes de chromosomes. Nous utilisons les opérateurs de mutation suivants qui à l'encontre des opérateurs de croisement n'ont besoin que d'un seul parent pour produire un enfant unique :

- Mutation inverse (im), qui a montré son efficacité parmi les opérateurs de mutation ;
- Mutation twors, qui a donné les meilleurs résultats en le combinant avec l'opérateur im.

5.6 Implémentation de l'algorithme génétique

Au début, la population initiale, utilisée par notre algorithme génétique, est constituée de $N = 2 * V$ individus. Chaque moitié est fabriquée par une méthode à part ; la première construit des individus en se basant sur la méthode du plus proche voisin alors que la deuxième moitié est composée par des individus produits par notre méthode 5.3. Nous calculons pour chaque individu sa valeur de la fonction objectif afin de les ordonner en ordre croissant. Ainsi, le meilleur individu aura le numéro 0 et le numéro (N-1) sera attribué au pire individu.

indice1 = 1% des meilleurs individus seront recopiés dans la nouvelle population,

$indice2 = 55\%$ de la nouvelle population est le résultat de croisement de deux chromosomes choisis au hasard : le premier de 1% , et le deuxième parmi les éléments ayant un indice entre 1% et 55% de l'ancienne population.

Deux tiers des croisements s'effectuent avec l'opérateur de croisement $edrx$ et un tiers avec l'opérateur de croisement $Cedrx$. Le reste de la nouvelle population est constitué par la mutation d'un individu choisi au hasard dans l'ancienne population par l'opérateur de mutation im .

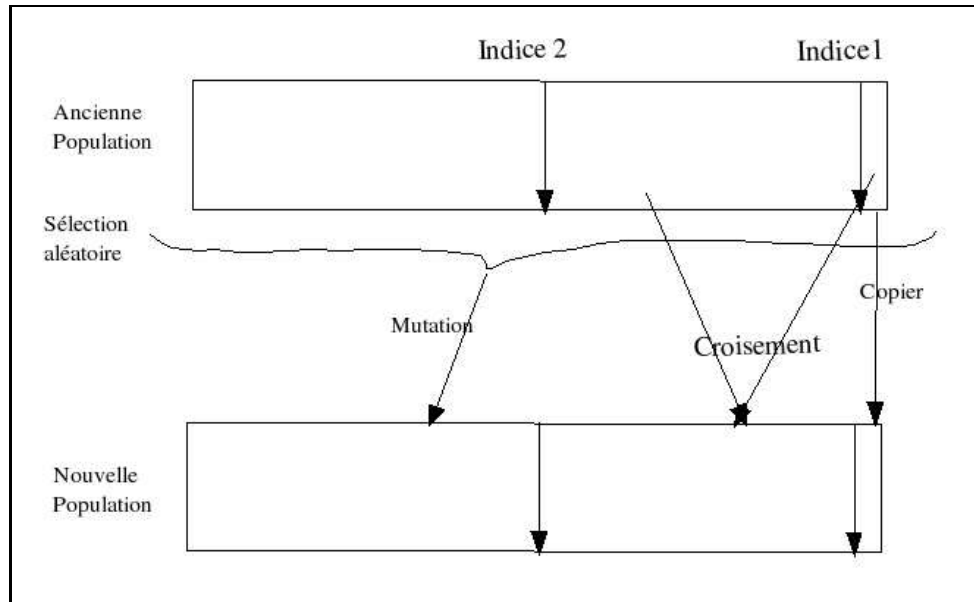


FIG. 5.1: La boucle principale de notre algorithme génétique lorsque la valeur de la fonction objectif du meilleur individu varie et ne reste pas fixe plus que 10 itérations.

Cette itération est répétée tant que le nombre maximum ($NMax = 2000$) n'a pas été dépassé et la valeur de la fonction objectif du meilleur individu est restée constante pendant au plus dix itérations.

Si la valeur de la fonction objectif du meilleur individu est restée constante pendant dix itérations, la nouvelle population est générée cette fois de la manière suivante : Chaque individu de l'ancienne population subira une recherche locale renforcée par le biais de N mutations dont le trois quart par l'opérateur de mutation im et un quart par l'opérateur de mutation $twors$. Le meilleur individu obtenu par cette recherche sera inséré dans la nouvelle population. La solution de cet algorithme est le meilleur individu de la population obtenue à $NMAX$ itérations.

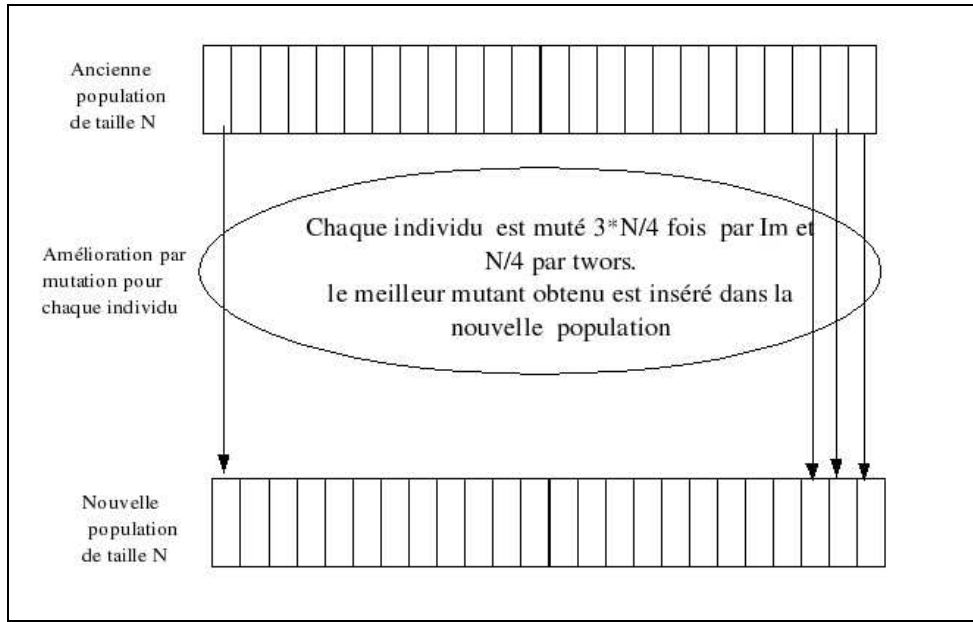


FIG. 5.2: La boucle de notre algorithme génétique lorsque la valeur de la fonction objectif est restée constante pendant 10 itérations.

5.7 Résultats numériques

Le schéma de notre variante de l'algorithme génétique, que nous avons proposé ci-dessus, a permis d'obtenir en l'appliquant sur des instances de [103] les résultats suivants :

Problème	Dimension	Borne supérieur des autres	Notre borne supérieur
berlin52	52	7542	7544
st70	70	860	860.83
gr96	96	512	512.76
kroa100	100	22500.241	22783.9
eil101	101	629	630.4
a280	279	2579	2579

TAB. 5.3: Les valeurs trouvées par notre algorithme génétique pour des problèmes de [103].

Ces résultats montrent l'utilité et les avantages de ce que nous avons préconisé. D'abord, l'utilisation du couple $edrx$ et $Cedrx$ en même temps avec une fréquence différente sur l'ensemble de la population sélectionnée comme opérateurs de croisement dans la boucle principale. L'opérateur $edrx$ permet de conserver les arêtes qui se répètent dans les parents et $Cedrx$ permet de construire des enfants différents des parents en essayant de casser, de supprimer leurs arêtes. Ensuite, il y a la boucle secondaire qui essaye de sortir du minimum local duquel la boucle principale n'a pas pu en sortir pendant dix

itérations en utilisant des mutations renforcées sur chaque élément de la population. Les bons résultats étaient au détriment du temps de calcul qui a augmenté considérablement. En prenant l'exemple de berlin52 avec l'algorithme génétique générationnel qui résulte des trois derniers chapitres, l'algorithme s'exécute en une durée moyenne de 0.4 secondes avec une borne supérieure du problème T.S.P. égale à 7804 alors que dans notre variante nous trouvons la borne supérieure de 7544 en 20 secondes.

Discussion

En utilisant uniquement un opérateur de croisement au sein de l'algorithme génétique, il vaut mieux utiliser l'opérateur **order1**. Il est plus stable que *edrx* et obtient les plus petites valeurs de la fonction objective. Donc, au niveau de l'exploration de l'ensemble des solutions, l'opérateur **order1** est le meilleur.

Nous avons remarqué que si nous nous intéressons uniquement aux opérateurs de mutation, nous avons trouvé l'ordre suivant pour les trois instances de T.S.P.. Le meilleur opérateur de mutation est **im**, suivi par *cim*, qui dépasse de peu *twors*. Ces opérateurs sont talonnés par *thrors* et en dernière position, on retrouve l'opérateur de mutation *throas*.

Après, nous avons pris en considération les deux opérateurs génétiques de croisement et de mutation. Et nous avons trouvé en utilisant une taille de la population fixe trois stratégies qui se mettent en exergue :

- *edrx*, $P_x = 1$, *im* et $P_m = 0.9$ que nous allons la nommer St1 ;
- *edrx*, $P_x = 0.4$, *im* et $P_m = 0.9$ nommée St2 ;
- St3 est composée de *edrx*, $P_x = 0$, *im* et $P_m = 0.9$.

De ce qui précède, l'opérateur de croisement, la probabilité et l'opérateur de mutation restent fixes dans les différentes stratégies. Alors que la probabilité de croisement se balade entre 0, 0.4 ou 1. Autrement dit, lorsque nous mutons avec *im* avec $P_m = 0.9$ et nous avons l'intention d'utiliser *edrx*, l'algorithme génétique donne de bons résultats par rapport aux autres paramètres et de croiser à 1 ou à presque moitié 0.4 ou 0 ne change presque rien.

Pour départager les probabilités de croisement, nous varions la taille de la population $N \in \{V, 2 \times V, 3 \times V, \dots, 10 \times V\}$, où V est la taille du problème.

Évidemment, lorsque la taille de la population croît, la valeur optimale obtenue par l'algorithme génétique s'améliore et nous avons trouvé qu'à partir de $6 \times V$, les valeurs de RMSE de chaque stratégie restent presque stables en moyenne. Les plus petites valeurs de RMSE, par rapport aux instances que nous avons étudié, sont obtenues pour *edrx*, $P_x = 1$, *im* et $P_m = 0.9$ avec $N = 8 \times V$. Les valeurs de probabilités de croisement et de mutation que nous avons retenu sont loin des valeurs existantes dans la littérature ($P_x \sim 0.6 : 0.8$ et $P_m \sim 10^{-3} : 10^{-2}$). Ces dernières valeurs ont été administrées à l'algorithme génétique mais ils fournissaient des résultats médiocres. Maintenant que nous avons fixé l'opérateur de croisement à *edrx* appliqué avec $P_x = 1$ et l'opérateur de mutation à *im* avec $P_m = 0.9$, nous allons nous intéresser à l'influence des méthodes de sélection sur la convergence de

cette variante de l'algorithme génétique. Pour cela, nous avons utilisé six méthodes de sélection :

- Méthode du tournoi ;
- Méthode du troncation ;
- Méthode de la roulette ;
- Méthode d'ordre exponentiel ;
- Méthode de Boltzmann ;
- Méthode d'ordre linéaire.

La méthode de sélection du tournoi avec $t = 0.8$ ou 0.9 sort du lot, suivie par la méthode "ordre linéaire" avec $\eta^- = 0.6$. Le paramètre de contrôle de la méthode de sélection du tournoi indique qu'il vaut mieux prendre un échantillon de taille proportionnelle à 0.9 de la taille de la population et d'en prendre le meilleur qui n'est pas forcément le meilleur individu de la population. Alors que le paramètre de contrôle de la méthode "ordre linéaire" indique qu'il vaut mieux octroyer une grande probabilité de sélection au mauvais individu. Cette politique va en sens tout à fait contraire de la précédente méthode. Mais ce qu'il faut retenir c'est que l'algorithme génétique travaille avec un "paquet" qui contient les mauvais et les meilleurs individus et il faut utiliser une méthode de sélection qui favorise les deux extrêmes et leur collaboration.

Toutefois, il faut signaler que la méthode de sélection par la roulette a donné des résultats satisfaisants par rapport aux autres méthodes.

Le point initial d'un algorithme déterministe joue un rôle primordiale dans la rapidité de sa convergence. Dans cette optique, nous avons comparé l'influence de la manière de générer la population initiale sur la convergence de l'algorithme génétique. Nous en avons défini six méthodes.

Nous avons remarqué que la méthode 4, consistant à générer le premier individu en choisissant le successeur parmi les voisins les plus proches non encore utilisés et après le muter pour compléter toute la population, est meilleure suivie par la méthode 6, basée sur le même mécanisme heuristique avec sa mutation en acceptant l'individu amélioré. La méthode 3 consistait à utiliser deux fois l'algorithme génétique ; le premier se base sur une génération de la population initiale aléatoire et le deuxième sur une population qui se composait du meilleur individu obtenu par le premier algorithme génétique et ses mutations. Elle s'avère plus concurrente, lorsque nous n'utilisons pas le mécanisme du plus proche voisin, à la méthode 1 qui se base sur la génération aléatoire de la population initiale. Au dernier chapitre, nous avons créé notre variante qui profite de ces remarques afin de résoudre le problème T.S.P.. Au niveau du codage, nous avons fixé le premier gène du chromosome. En ce qui concerne la population initiale, nous avons utilisé le plus proches voisin. La population a été divisée en trois parties ; la première contient les 1% meilleurs individus, les 55% moins meilleurs que les précédents composent la deuxième partie alors que les plus faibles constituent la troisième partie. Pour croiser deux individus choisis au hasard l'un de la première partie et l'autre de la deuxième partie, nous nous sommes servis de deux opérateurs de croisement *edrx* et du notre *Cedrx*. Un individu est sélectionné au hasard pour être muté avec *im*.

Lorsque la valeur de la fonction objectif du meilleur individu ne change pas pendant dix

itérations. Nous procédons autrement pour la génération de la population suivante en renforçant la mutation pour chaque individu de l'ancienne population afin de sortir de ce minimum local. C'est vrai que le temps de calcul a augmenté mais ceci au profit de bon résultat.

tel-00126292, version 2 - 8 Mar 2007

Troisième partie

Applications de l'algorithme génétique

Introduction

Après une présentation générale de l'algorithme génétique dans la première partie et son utilisation pour résoudre le problème T.S.P. dans la deuxième partie, nous nous intéressons dans cette partie à la résolution de problèmes concrets d'ordonnancement en tirant profit des résultats que nous avons obtenu.

Cette troisième partie est composée de trois chapitres.

Le premier est consacré au problème Job Shop, noté JSP, qui appartient à la famille des problèmes NP-complet. Le problème de Job Shop de type (J, M) est un atelier qui contient M machines servant à produire J tâches. Chaque tâche est composée d'un nombre fini d'opérations qui peuvent être exécutées dans un ordre prescrit. Chaque opération ne peut être exécutée que sur une seule machine et chaque machine ne peut réaliser qu'une seule opération à la fois.

L'objectif est de trouver un ordonnancement réalisable qui minimise la durée totale de la production de toutes les tâches (la minimisation du "Makespan"). C'est la différence entre la date de fin de la dernière opération et la date du début de la première opération de de l'ensemble des tâches.

Nous ferons un tour d'horizon de ce qui a été réalisé pour sa résolution en utilisant l'algorithme génétique. À la fin, nous présenterons notre variante de l'algorithme génétique appliqué à ce type de problème et ses résultats.

Nous nous intéressons dans le deuxième chapitre au problème concret d'atterrissage d'avions (ALP). Il consiste à ordonner une séquence d'atterrissage des avions sur une piste de l'aéroport, qui peut être plus performante que celle utilisée par les contrôleurs aériens. Il s'agit de trouver une séquence qui minimise le temps total d'atterrissage des appareils, en respectant les fenêtres (intervalles d'atterrissages) et les temps de séparation avec les avions précédents. Nous présenterons notre algorithme pour la résolution de ALP et ses applications sur des données réelles.

Le troisième chapitre est consacré au problème concret d'ordonnancement d'une chaîne de production. Nous commençons par la définition du problème ainsi que ses contraintes de l'atelier de peinture et de l'atelier de montage. Et comme c'est un problème multicritère, nous exposons deux stratégies de résolution ainsi que les algorithmes génétiques correspondants avec une étude comparative entre les deux algorithmes réalisés.

Nous utilisons une machine avec CPU 2593.547 Mhz avec un système d'exploitation Linux Mandrake et le langage C comme langage de de programmation.

Chapitre 1

Le problème de Job Shop (JSP)

Sommaire

1.1	Rappel	103
1.2	Les générateurs de solution d’ordonnancement	105
1.3	Règles de priorité	107
1.4	Formulation du problème JSP	108
1.5	Codage d’une solution de JSP	108
1.6	Implémentation de l’algorithme génétique	110
1.6.1	Représentation	110
1.6.2	Génération de la population initiale	110
1.6.3	Calcul du Makespan	113
1.6.4	Sélection	113
1.6.5	Opérateur de croisement	113
1.6.6	Opérateur de mutation	113
1.6.7	Mécanisme de réparation des individus générés	115
1.6.8	Méthode d’insertion	115
1.7	Résultats numériques	115
1.8	Conclusion	116

1.1 Rappel

Un problème d’ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d’enchaînement) et de contraintes portant sur la disponibilité des ressources requises.

En production (manufacturière, biens, service), on peut le présenter comme un problème où il faut réaliser le déclenchement et le contrôle de l’avancement d’un ensemble de commandes à travers les différents centres composant le système.

Un ordonnancement constitue une solution au problème d’ordonnancement. Il est défini par le planning d’exécution des opérations (« ordre » et « calendrier ») et d’allocation

des ressources et vise à satisfaire un ou plusieurs objectifs.

Une **opération** est une entité élémentaire localisée dans le temps par une **date de début** et/ou de fin, dont la réalisation nécessite une durée, et qui consomme un moyen selon une certaine intensité.

Selon les problèmes, les opérations peuvent être exécutées par morceaux (*préemptifs*), ou doivent être exécutées sans interruption (*non préemptifs*). Lorsque les opérations ne sont soumises à aucune contrainte de cohérence, elles sont dites indépendantes.

Plusieurs opérations peuvent constituer une **tâche** et plusieurs tâches peuvent définir un processus.

La **ressource** est un moyen technique ou humain destiné à être utilisé pour la réalisation d'une opération et disponible en quantité limitée.

Plusieurs types de ressources sont à distinguer. Une ressource est *renouvelable* si après avoir été allouée à une ou plusieurs opérations, elle est à nouveau disponible en même quantité (les hommes, les machines, l'équipement en général) ; la quantité de ressource utilisable à chaque instant est limitée. Dans le cas contraire, elle est *consommable* (matières premières, budget) ; la consommation globale au cours du temps est limitée. Une ressource est doublement contrainte lorsque son utilisation instantanée et sa consommation globale sont toutes deux limitées (l'argent en est un bon exemple).

Qu'elle soit renouvelable ou consommable, la disponibilité d'une ressource peut varier au cours du temps. Sa courbe de disponibilité est en général connue a priori, sauf dans les cas où elle dépend du placement de certaines tâches génératrices.

On distingue par ailleurs principalement dans le cas de ressources renouvelables les ressources *disjonctives* qui ne peuvent exécuter qu'une opération à la fois (machine-outil, robot manipulateur) et les ressources cumulatives qui peuvent être utilisées par plusieurs tâches simultanément mais en nombre limité (équipe d'ouvriers, poste de travail).

Les **contraintes** expriment des restrictions sur les valeurs que peuvent prendre simultanément les variables de décision. On distingue :

- contraintes temporelles comme :
 - les contraintes de temps alloué, issues généralement d'impératifs de gestion et relatives aux dates limites des opérations (délais de livraisons, disponibilité des approvisionnements) ou à la durée totale d'un projet ;
 - les contraintes de cohérence technologique, ou contraintes de gammes, qui décrivent des relations d'ordre relatif entre les différentes opérations ;
- contraintes de ressources comme :
 - les contraintes d'utilisation de ressources qui expriment la nature et la quantité des moyens utilisés par les opérations, ainsi que les caractéristiques d'utilisation de ces moyens ;
 - les contraintes de disponibilité des ressources qui précisent la nature et la quantité des moyens disponibles au cours du temps.

Dans la résolution d'un problème d'ordonnement, il y a plusieurs critères d'optimisation construits sur la base d'indicateurs de performance. On cherchera donc à optimiser ces critères :

- le temps total d'exécution ;
- le temps moyen d'achèvement d'un ensemble de tâches ;
- différents retards (maximum, moyen, somme, nombre, etc.) ou avances par rapport aux dates limites fixées.

1.2 Les générateurs de solution d'ordonnement

Un générateur d'ordonnement est une procédure qui à partir d'un ensemble d'informations lui servant de guide va construire un ordonnancement réalisable. On distingue trois grandes familles d'ordonnement :

Ordonnement semi-actif les opérations sur les différentes machines sont calées à gauche et on ne peut pas changer leurs emplacements à gauche et améliorer le critère d'optimisation.

Ordonnement actif c'est un ordonnancement semi-actif dont il est impossible d'avancer une opération sur une machine sans obligatoirement en reculer une autre ou il n'existe pas d'intervalle où la machine est libre et qui soit suffisamment grand pour qu'une opération soit avancée tout en respectant les contraintes de précédences des opérations d'une tâche. Le générateur d'ordonnement actif décrit par [9] fonctionne de la manière suivante :

Initialiser la date de début de chaque machine à 0 ;
 T : ensemble de toutes les opérations de toutes les tâches ;
Tant que ($T \neq \emptyset$) **faire**
 en considérant les opérations de T dont toutes les opérations qui précèdent dans le même travail sont déjà placées
 Trouver une opération que si on la place au plus tôt sur sa machine d'exécution M^* , aura la plus petite date de fin C^* ;
 Parmi les opérations utilisant la machine M^* et dont toutes les opérations qui précèdent dans la même tâche sont déjà placées : se limiter aux opérations dont la date de début peut être strictement inférieure à C^* et parmi ces opérations, choisir la plus prioritaire (selon les règles de choix intégrées) noté O^* ;
 Placer l'opération O^* sur la machine M^* et l'enlever de T ;
Fait

Algorithme 9: Générateur d'ordonnement actif

Ordonnancement sans délai C'est un ordonnancement actif dans lequel on ne laisse jamais une machine inoccupée lorsque au moins une opération qui l'utilise est disponible pour y être exécutée. Un générateur d'ordonnancement sans délai fonctionne de la manière suivante :

```

M : nombre de machines ;
t : tableau de dimension M ;
t(k) = 0  $\forall$  machine k ;
T : ensemble de toutes les opérations de toutes les tâches ;
Tant que (T  $\neq$   $\emptyset$ ) faire
    chercher une machine  $k^*$  disponible au plus tôt  $\min\{t(k)\}$  ;
    chercher l'ensemble R des opérations qui peuvent être ordonnancées sur
     $k^*$  à l'instant  $t(k^*)$  ;
    Si (R =  $\emptyset$ ) Alors
        |  $t(k^*) \leftarrow t(k^*) + 1$  ;
    Sinon
        | choisir  $O^*$  l'opération la plus prioritaire de R ;
        | placer l'opération  $O^*$  sur la machine  $k^*$  ;
        | corriger la date de disponibilité de la machine  $k^*$  ;
        | enlever  $O^*$  de T ;
    Fin Si
Fait

```

Algorithme 10: Générateur d'ordonnancement sans délai

Baker [9] a démontré que pour les critères d'optimisation réguliers, en l'occurrence le Makespan, les ordonnancements semi-actifs et les ordonnancements actifs sont des sous-ensembles de solutions dominantes. C'est à dire qu'ils contiennent au moins une solution optimale pour le critère considéré. Alors qu'une solution optimale peut ne pas appartenir aux ordonnancements sans délai.

1.3 Règles de priorité

Dans l'ordonnancement de production, il y a plusieurs règles de priorités qui permettent de favoriser et de choisir une opération parmi les autres afin de construire des solutions. Nous en citerons :

Premier arrivé, premier servi : Les opérations sont traitées dans l'ordre de leurs arrivées dans la file d'attente ;

Dernier arrivé, premier servi : Les opérations sont traitées dans l'ordre inverse de leurs arrivées dans la file d'attente ;

La durée d'exécution la plus courte Les opérations sont traitées dans l'ordre croissant de leurs durées opératoires ;

La marge de temps : Les opérations sont traitées dans l'ordre croissant de leur marge de temps. Ainsi, les opérations avec le moins de marge sont prioritaires ;

Le ratio critique : Les opérations sont traitées dans l'ordre croissant de leur ratio critique. Le ratio critique est obtenu par la formule suivante :

$$\text{ratio critique} = \frac{\text{temps restant avant exigibilité}}{\text{somme des temps opératoires restants}}.$$

1.4 Formulation du problème JSP

Le Job Shop de type (J, M) est un problème d'ordonnancement d'ateliers. Il contient du fait de sa structure deux problèmes classiques de l'optimisation combinatoire :

- Le problème d'affectation ;
- Le problème d'ordonnancement.

Les données de ce problème sont les suivantes :

- Un ensemble de M machines.
- Un ensemble de J tâches dont chacune est composée d'une gamme opératoire de $O_{i,j}$ ($i = 1, \dots, J$ et $j = 1, \dots, M$) d'opérations fixées.
- Une opération $O_{i,j}$ représentant la $j^{\text{ème}}$ opération de la tâche i . Cette opération nécessite un temps opératoire $P_{i,j}$.
- L'opération $O_{i,j}$ nécessite pour être réalisée une machine k appartenant à $\{1, 2, \dots, M\}$.

Ces données sont soumises aux contraintes suivantes :

- Les machines sont indépendantes les unes des autres ;
- Une machine ne peut exécuter qu'une seule opération à un instant donné ;
- Chaque machine est disponible pendant toute la période de l'ordonnancement ;
- Une opération en cours d'exécution ne peut pas être interrompue ;
- Les tâches sont indépendantes les unes des autres.

Soit C_{max} la durée totale de l'exécution de toutes les opérations qui composent les J tâches. Le but de l'ordonnancement est de minimiser la fonction C_{max} .

1.5 Codage d'une solution de JSP

Il y a plusieurs types de codage qui permettent de représenter les solutions d'un problème d'ordonnancement de type Job Shop. un codage permet de connaître la localisation temporelle des opérations sur les machines. Yamada [107] a proposé un codage qui consistait à donner la date de fin d'exécution $F_{i,j}$ de toute opération $O_{i,j}$. Pour coder une solution, il suffit d'utiliser des vecteurs d'entiers strictement positifs de $J * M$ éléments. Un vecteur d'entiers positifs ne constitue pas une solution réalisable car les entiers doivent vérifier à la fois les contraintes de précedence liées aux gammes et les contraintes liées à la succession des opérations sur la même machine. Une seconde expression de ce type de codage

est d'utiliser à la place de la date de fin d'exécution la date de son début. On remarque dans le cas des problèmes simples à une machine, que donner les dates de début ou de fin d'exécution des opérations est équivalent à donner l'ordre dans lequel les opérations sont exécutées sur la machine. Ce codage appelé codage de permutation et proposé pour la première fois comme codage symbolique par Davis [38]. Son avantage c'est la possibilité d'utiliser tous les opérateurs de croisement et de mutation dans l'algorithme génétique pour résoudre le problème T.S.P.. Il y a un autre codage très souvent utilisé en ordonnancement qui donne à chaque machine les opérations qui lui sont affectées. Les codages que nous venons de présenter permettent de représenter les solutions du problème de Job Shop mais rencontrent la difficulté de connaître la réalisabilité de la représentation. Ce dernier point est observable après croisement ou mutation. Face à ce problème, nous trouvons deux manières possibles :

- concevoir des codages qui correspondent toujours à des solutions réalisables ainsi que les opérateurs de croisement et de mutation associés. Dans ce cas, nous parlerons de **codage direct** puisqu'il y a une correspondance biunivoque entre l'ensemble des chromosomes codés et l'ensemble de solutions.
- créer une procédure de transformation qui permet à partir du génotype (informations dans le chromosome) d'obtenir le phénotype (solution obtenue à partir du génotype). Le codage est dit **codage indirect** puisque plusieurs génotypes peuvent donner le même phénotype.

La représentation des solutions infaisables pour le problème de Job Shop peut être évitée par une légère modification du schéma de permutation. Bierwirth et al. [19] ont introduit une stratégie de représentation qui modifie la permutation en utilisant un index qui montre l'ordre d'apparition d'une opération donnée en décodant.

Nous avons regroupé dans ce tableau les différents types de codage de la solution de JSP trouvés dans la littérature :

Codage	auteur(s)
Preference list-based representation	Davis [37], Falkenauer <i>et al.</i> [44], Croce <i>et al.</i> [34], Kobayashi <i>et al.</i> [67]
Job pair relation-based representation	Nakano <i>et al.</i> [82], Paredis [91]
Problem specific representation	Bagchi <i>et al.</i> [8]
Completion time-based representation	Yamada <i>et al.</i> [107]
Disjunctive graph-based representation	Tamaki <i>et al.</i> [101]
Job-based representation	Holsapple <i>et al.</i> [61], Byung <i>et al.</i> [24]
Operation-based representation	Fan <i>et al.</i> [45], Gen <i>et al.</i> [48]
Random key representation	Bean [10], Norman <i>et al.</i> [83], Gonçalves <i>et al.</i> [55]
Priority rule-based and machine-based	Dorndorf <i>et al.</i> [41]

TAB. 1.1: Les différents codages d'une solution de JSP dans la littérature

1.6 Implémentation de l'algorithme génétique

Le problème du Job Shop c'est un problème d'optimisation NP-difficile. Pour le résoudre, nous appliquons un algorithme génétique "générationnel" ([22],[23]). Nous utilisons un codage réel pour la représentation des chromosomes, le principe de la roulette pour la sélection. L'originalité de notre variante réside par le choix de trois opérateurs efficaces pour le croisement et un opérateur pour la mutation.

Notre algorithme commence par générer une population initiale de N individus, pour lesquels, nous calculons la valeur de la fonction objectif et nous sélectionnons les individus les mieux adaptés en appliquant le principe de la roulette. Les individus, sujets de croisement par trois opérateurs de croisement. Leurs résultats sont mutés par un opérateur de mutation avec une probabilité de mutation P_m .

Les individus issus de ces opérateurs génétiques seront insérés dans la nouvelle population, en utilisant la méthode d'insertion élitisme. À la fin, un test d'arrêt sera effectué. Il consiste à voir si le nombre maximum d'itérations a été dépassé. Si ce test est vérifié alors l'algorithme s'arrête avec une solution optimale, sinon on répète le processus sur la nouvelle génération.

1.6.1 Représentation

Considérons l'exemple d'un job shop composé de 3 machines et 3 tâches suivant :

Tâche i	$O_{ij} \Rightarrow$ Machine (durée opératoire)		
tâche 0	$O_{00} \Rightarrow 2$ (3)	$O_{01} \Rightarrow 1$ (6)	$O_{02} \Rightarrow 0$ (2)
tâche 1	$O_{10} \Rightarrow 0$ (2)	$O_{11} \Rightarrow 1$ (5)	$O_{12} \Rightarrow 2$ (2)
tâche 2	$O_{20} \Rightarrow 1$ (1)	$O_{21} \Rightarrow 0$ (2)	$O_{22} \Rightarrow 2$ (1)

Tout d'abord, le nombre total des opérations de cet exemple est 9.

Nous octroyons à chaque opération un numéro entre 0 et 8 en respectant l'ordre des opérations pour chaque tâche. Ainsi, les trois opérations de la tâche 0 auront comme code 0, 1 et 2. Pour la tâche 1, les opérations sont numérotées 3, 4 et 5. Et enfin, les opérations 6, 7 et 8, dans cet ordre, constituent la tâche 2.

Un chromosome est codé par un tableau d'entiers de 0 à 8 qui respecte l'ordre entre les opérations de la même tâche. Ainsi 0 correspond à l'opération O_{00} , 5 correspond à O_{11}

Le tableau [0, 3, 1, 4, 6, 7, 2, 5, 8] correspond à un ordonnancement faisable des opérations sur les machines. Ce n'est pas le cas pour [0, 3, 1, 5, 4, 2, 6, 7, 8]. En effet, l'opération 5 précède l'opération 4 ce qui ne respecte pas la gamme opératoire de la tâche 1.

1.6.2 Génération de la population initiale

Un chromosome est construit en utilisant le générateur d'ordonnancement actif (Algorithme 9) en se basant sur la règle de priorité de la plus courte durée d'exécution. Il est

inséré dans la population initiale, complétée par des chromosomes générés aléatoirement. Un chromosome est constitué de $M \cdot J$ gènes correspondants aux opérations. A chaque fois, nous choisissons au hasard une tâche dont une opération, qui n'était pas encore marquée, sera placée dans le chromosome. Cette opération sera distinguée et marquée et si la même tâche est choisie après alors son successeur sera placé dans le chromosome et il sera marqué. Si toutes les opérations d'une tâche ont été placées dans le chromosome alors cette tâche sera marquée afin qu'elle ne soit plus choisie par la suite. La procédure qui génère un chromosome est explicitée ci-dessous :

```

entiers j, k, l, idx, tmp, compteur;
entiers J, M;
Tab, chrom : Deux tableaux d'entiers de taille J * M;
job : Tableau d'entiers de taille J;

Pour j de 0 à (J * M)-1 faire
    Tab[j] ← j;
Fin Pour
Pour j de 0 à J faire
    job[j] ← j;
Fin Pour
j ← 0;
Tant que ( ( j < J * M ) ) faire
    tic : l ← nombre aléatoire entre 0 et (J - 1);
    Si ((job[l]==-1)) Alors
        aller à tic;
    Fin Si
    compteur ← 0;
    Pour idx de 1 * M à ((l+1) * M)-1 faire
        Si (( Tab[idx]!≠ -1 )) Alors
            chrom[j] ← Tab[idx];
            Tab[idx] ← -1;
            j ← j+1;
            sortir du boucle pour idx;
        Sinon
            compteur ← compteur +1;
            Si ((compteur == M)) Alors
                job[l] ← -1;
                aller à tic;
            Fin Si
        Fin Si
    Fin Pour
Fait

```

1.6.3 Calcul du Makespan

Le makespan est la durée opératoire totale entre la date de fin de la dernière opération de l'ordonnancement et la date du début de sa première opération.

Pour le calculer, il faut connaître la date du début de chaque opération sur sa machine de production.

Nous définissons respectivement P_{ij} , T_{ij} et F_{ij} , la durée opératoire, la date du début et la date de fin de chaque opération O_{ij} .

P_{ij} est une donnée du problème; F_{ij} est donné par la formule suivante : $F_{ij} = T_{ij} + P_{ij}$; T_{ij} est défini par la formule suivante : $T_{ij} = \text{Max}(F_{i(j-1)}, T_{hl} + P_{hl})$

avec O_{hl} est l'opération précédente de O_{ij} sur la machine M_{ij} et $F_{i(j-1)}$ est la date de fin de l'opération, qui précède O_{ij} , dans la tâche i .

Les cas suivants se présentent :

- Si O_{ij} est la première opération de la tâche i alors $F_{i(j-1)} = 0$.
- Si O_{ij} est la première opération à exécuter sur la machine M_{ij} alors $F_{hl} = T_{hl} + P_{hl}$ est remplacé par 0.

Le makespan est le maximum des dates de la fin des dernières opérations pour chaque tâche :

$$C_{max} = \text{Max}\{F_{hl}\}.$$

1.6.4 Sélection

La méthode utilisée est la sélection par la roulette. Elle consiste à associer à un chromosome i de la population une portion proportionnelle à $(1 - f_i) / (\sum f_j)$ où f_i est la valeur de la fonction objectif pour l'individu i . Ainsi, les individus qui ont les petites valeurs de la fonction d'évaluation peuvent avoir une forte chance d'être acceptés et être candidats à l'opération de croisement.

1.6.5 Opérateur de croisement

Nous allons utiliser à la fois trois opérateurs de croisement :

order1 : opérateur utilisé deux fois sur cinq.

edrx : opérateur utilisé deux fois sur cinq.

Cedrx : opérateur utilisé une fois sur cinq.

Ce choix a été guidé par notre souci d'agrandir l'ensemble exploré dans le domaine des solutions.

1.6.6 Opérateur de mutation

L'opérateur de mutation évite d'établir des populations uniformes incapables d'évoluer. Il consiste à modifier les valeurs des gènes de chromosomes à partir de ceux d'un seul parent.

Si un chromosome est choisi pour la mutation, nous déterminons la machine sur laquelle F_{ij} est égale à son C_{max} . Sur cette machine, nous cherchons la dernière opération et

sa tâche correspondante. Nous choisissons aléatoirement une autre tâche distincte et nous échangeons les emplacements des opérations de ces deux tâches dans le chromosome en respectant la gamme opératoire de chaque tâche. Cet opérateur de mutation est explicité dans l'algorithme suivant :

```

entiers machcritiq, tmp,job, job1, k1, k2;
entiers indice1, indice2;
entiers dernierjob, dernierop;
entiers i, j;

machcritiq = machine critique du chromosome sur laquelle le makespan est réalisé;
dernierop = recherche de la dernière opération qui opère sur la machine machcritiq;
dernierjob = dernierop /M;

Répéter
  | job ← nombre aléatoire entre 0 et(J -1);
jusqu'à ce que (job ≠ dernierjob))
Pour i de 1 à (M ) faire
  | k1 ← job * i;
  | k2= dernierjob*i;
  | Pour j de 0 à (J*M -1) faire
  | | Si ((k1 == chrom[j])) Alors
  | | | indice1 ← j;
  | | |
  | | | Fin Si
  | | | Si ( (k2 == chrom[j])) Alors
  | | | | indice2 ← j;
  | | | |
  | | | | Fin Si
  | | Fin Pour
  | | tmp ← chrom[indice1];
  | | chrom[indice1] ← chrom[indice2];
  | | chrom[indice2] ← tmp;
  | Fin Pour

```

En fait, cet opérateur permute l'emplacement des opérations de deux tâches différentes; l'une est choisie aléatoirement et l'autre contient la dernière opération dont sa date de fin n'est que le makespan.

1.6.7 Mécanisme de réparation des individus générés

Les chromosomes obtenus par les opérateurs génétiques de croisement ne sont pas toujours faisables. Pour enlever cette anomalie, nous utilisons un mécanisme qui veille sur le respect de la gamme opératoire de chaque tâche et qui permet de corriger ce chromosome et le rendre réalisable. (c.à.d. qui respecte l'ordonnancement des opérations de chaque tâche).

```

Chrom : tableau d'entiers de taille M * J ;
entiers : tmp, i, j, k ;
tab : tableau d'entiers de taille J * M ;

Pour i de 0 à J * M faire
    tmp ← chrom[i] ;
    tab[i] ← la tâche à laquelle appartient l'opération tmp ;
Fin Pour
Pour i de 0 à (J-1) faire
    k ← i * M ;
    Pour j de 0 à (J*M -1) faire
        Si ((tab[j] == i)) Alors
            chrom[j] ← k ;
            k ← k+1 ;
        Fin Si
    Fin Pour
Fin Pour

```

1.6.8 Méthode d'insertion

Nous avons utilisé, la méthode d'insertion élitisme qui consiste à recopier le meilleur chromosome de l'ancienne population dans la nouvelle. Celle-ci est complétée par chaque individu généré par les opérateurs génétiques que nous l'insérons au fur et à mesure jusqu'à son remplissage. Nous veillons à ce que la taille de la population reste fixe de génération en génération.

1.7 Résultats numériques

Nous travaillons sur les instances de Lawrance [62], et nous comparons nos résultats avec celles d'Ombuki *et al.* [86].

Notre algorithme se base sur :

- La taille de population de 2000,

- Les trois opérateurs de croisement sont order1, edrx et Cedrx.
- L'opérateur de mutation avec $P_m = 0.5$,
- Nombre maximum d'itérations = 600.

Nous récapitulons les résultats, dans le tableau suivant :

Instances [62]	Nbr. tâches	Nbr. Machines	M.S.C.	sGA	gkGA	LSGA [2]	Notre GA
La01	10	5	666	667	677	666	666
La05	15	5	593	593	593	593	593
La06	15	5	926	926	926	926	926
La07	15	5	890	891	890	890	890
La08	15	5	863	863	863	863	863
La09	15	5	951	951	951	951	951
La10	15	5	958	958	958	958	958
La11	20	5	1222	1222	1241	1222	1222
La12	20	5	1039	1039	1039	1039	1039
La13	20	5	1150	1150	1150	1150	1150
La14	20	5	1292	1292	1292	1292	1292
La15	20	5	1207	1256	1302	1207	1207

TAB. 1.2: Comparaison de nos résultats avec celle d'Ombuki par rapport aux instances de Lawrance

- La colonne M.S.C. donne les meilleures valeurs connues pour ces instances ;
- La colonne sGA donne les valeurs trouvées par Ombuki *et al.* [85] en utilisant un algorithme génétique basé uniquement sur les mutations ;
- La colonne gkGA fait référence aux valeurs trouvées par Ombuki *et al.* [85] ;
- La colonne LSGA fait référence aux nouvelles valeurs trouvées par Ombuki *et al.* [86].

Les résultats trouvés sont à pied d'égalité avec LSGA et meilleurs que les autres algorithmes "sGA" et "gkGA".

1.8 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle variante d'algorithme génétique. Elle est caractérisée par :

- La manière de coder les chromosomes (la représentation des individus) qui a contribué efficacement à la diminution de la valeur de la fonction objectif d'une part et à respecter l'ordonnancement des opérations de chaque tâche d'autre part ;
- La population initiale est composée d'un individu généré heuristiquement et d'autres aléatoirement ;
- Le choix de l'opérateur de mutation a permis de sortir des minimums locaux et d'obtenir des meilleurs résultats ;
- Le choix des valeurs de P_m et P_x ($P_m = 0.5$ et $P_x = 1$) a fait l'équilibre entre la mutation et le croisement ;

- L'utilisation de trois opérateurs de croisement à la fois a permis de mieux explorer le domaine de solutions :
 - order1** : opérateur utilisé deux fois sur cinq.
 - edrx** : opérateur utilisé deux fois sur cinq.
 - Cedrx** : opérateur utilisé une fois sur cinq.

Les expérimentations numériques ont montré l'efficacité et la robustesse de notre algorithme. En effet, dans la plupart des cas, il donne les meilleures valeurs de la fonction objectif déjà publiées.

Dans certains cas, la diminution de la fonction objectif était très lente ce qui nous a obligé à augmenter le nombre maximum d'itérations, que nous avons fixé après à 600, pour permettre la convergence de notre algorithme vers la solution optimale.

Chapitre 2

Le problème d’atterrissage des avions

Sommaire

2.1	Introduction	119
2.2	Le problème d’atterrissage des avions	120
2.3	État de l’art	120
2.4	Algorithme génétique	122
2.4.1	Définition des paramètres	122
2.4.2	Génération de la population initiale	122
2.4.3	Calcul des heures d’atterrissage des avions	122
2.4.4	Opérateur de croisement	125
2.4.5	Opérateur de mutation	125
2.4.6	Organigramme de notre algorithme génétique	126
2.5	Résultats numériques	126
2.6	Algorithmes de population heuristique	127
2.6.1	Algorithme génétique (AG)	128
2.6.2	Méthode de recherche Scatter (SS)	130
2.6.3	Algorithme Bionomique (BA)	131
2.7	Notre algorithme de population	133
2.8	Résultats	133
2.9	Conclusion	134

2.1 Introduction

Le problème d’atterrissage des avions (ALP) consiste à ordonner une séquence d’atterrissage des avions sur un aéroport, qui peut être plus performante que celle utilisée par les contrôleurs aériens. Il s’agit de trouver une séquence qui minimise le temps total d’atterrissage des appareils, en respectant les fenêtres (intervalles d’atterrissage) et les temps de séparation avec les avions précédents. Au départ, nous essayons de résoudre ce problème en utilisant l’algorithme génétique avec trois différentes fonctions objectifs que nous allons expliciter. Après, nous proposons au algorithme de population pour la

résolution du problème ALP. L'objectif est de fournir la meilleure façon de l'optimiser en utilisant l'algorithme génétique.

2.2 Le problème d'atterrissage des avions

Chaque avion $i \in I = \{1, 2, \dots, P\}$ a une heure prévue d'arrivée T_i , autour de laquelle doit se situer l'heure réelle d'arrivée X_i , dans une fourchette fixée par une heure d'atterrissage au plus tôt E_i et une heure d'atterrissage au plus tard L_i . L'arrivée d'un avion i en avance (resp. en retard) par rapport à son heure prévue T_i le pénalise occasionnant un coût Av_i (resp. Ap_i) par minute. Un intervalle de temps minimal Sep_{ij} de sécurité doit séparer deux atterrissages de deux avions i et j .

L'objectif est de déterminer les heures d'atterrissage des avions X_i ($i=1,2,\dots,P$) qui permettent de minimiser la pénalité totale, et la rendre nulle si c'est possible, tout en respectant les fenêtres et les intervalles de séparation.

Pour les P avions qui attendent leurs tours pour atterrir,

$$E_i \leq X_i \leq L_i \quad \forall i \in I \quad (2.1)$$

C.à.d.

$$X_i = E_i + y_i * (L_i - E_i) \quad \text{où } y_i \in [0, 1] \quad (2.2)$$

Pour tout avion i précédant l'avion j sur une piste $r \in \{1, 2, \dots, R\}$, où R est le nombre de pistes, on a :

$$X_i + SEP_{ij} \leq X_j \quad (2.3)$$

Sous ces trois contraintes, nous minimisons F_X la fonction d'évaluation de X :

$$\min F_X = \min \sum_{i \in I} (Av_i \times \max\{0, T_i - X_i\} + Ap_i \times \max\{0, X_i - T_i\}) \quad (2.4)$$

2.3 État de l'art

Beasley et al. ([11]) présentent une formulation des (ALP) sous forme d'un programme linéaire en variables mixtes. Ils proposent six contraintes additionnelles afin de réduire l'espace de solutions. Le problème est alors résolu avec un algorithme heuristique de façon optimale en employant la recherche par arborescence pour des instances d'OR-library ([4]) impliquant jusqu'à 50 avions.

Ernst et al. ([43]) présentent un algorithme de simplexe spécial qui évalue très rapidement le temps d'atterrissage des avions, en se basant sur l'information d'ordre partiel. Cette méthode est intégrée au sein d'une méthode de recherche heuristique dans l'espace de solutions comme la méthode branch-and-bound pour les (ALP) avec une ou plusieurs pistes. Des étapes préalables sont employées en détectant les fenêtres de temps serrées et l'ordre partiel obtenus à partir du problème ou de sa borne supérieure. Cet algorithme a fait ses preuves pour des instances prélevées d'OR-library ([4]) impliquant jusqu'à 50

avions et 4 pistes.

Jung et al. ([65]) proposent un algorithme heuristique basé sur la segmentation du temps. L'horizon de temps est divisé en segments de temps qui déterminent des sous-problèmes de (ALP). Chaque sous-problème est formulé comme problème linéaire en nombres entiers comme chez Beasley et al. ([11]) et résolu de façon optimale. Des résultats sont présentés pour des instances d'OR-library ([4]) et pour d'autres générées aléatoirement allant jusqu'à 75 avions et 4 pistes.

Cheng et al. ([30]) développent quatre formulations différentes de recherche génétique pour le problème (ALP) avec multiples pistes. Trois de ces schémas emploient un algorithme génétique tandis que le quatrième schéma utilise la programmation génétique. Des résultats présentés impliquant 12 avions et 3 pistes.

Pinol et al. ([13]) ont appliqué d'abord la méthode *scatter search* et *bionomic algorithm* pour les (ALP) avec multiples pistes. La population initiale se compose entre autres de trois individus basés sur l'ordre croissant de l'heure au plus tôt, heure préférentielle et de l'heure au plus tard de l'ensemble d'avions. La formule (2.4) définit la fonction objectif tandis que la valeur d'inaptitude est mesurée par la violation des contraintes de temps de séparation. Dans la méthode *scatter search*, la méthode de sélection des parents utilisée est la méthode de sélection tournoi. Au début on octroie à chaque avion un poids aléatoire et après on introduit pour un nouvel individu généré par croisement qui constitue une combinaison linéaire de proportions correspondantes aux parents. En outre, un test de duplication est employé pour maintenir un niveau de diversité dans la population. Alors un programme linéaire simple sur l'ordre d'atterrissage avec l'ordre fixé est appliqué pour améliorer le nouvel individu. Après, le nouvel enfant est inséré à la place du plus mauvais individu dans la population en gardant la constance de la taille de la population. Pinol et al. ([13]) ont considéré la fonction objectif linéaire et non linéaire et ils ont obtenu des résultats pour des instances allant jusqu'à 500 avions et 5 pistes.

Bianco et al. ([18]) ont également adopté une approche basée sur une formulation en programmation dynamique et deux algorithmes heuristiques. Des résultats numériques sont présentés pour un certain nombre de problèmes aléatoirement produits et aussi bien pour les problèmes de OR-library ([4]).

Parmi les algorithmes heuristiques, il y a celui de Beasley et al. ([11]) connu par sa rapidité à fournir les solutions optimales. Cependant, la qualité de solutions n'est pas stable. Pour les instances testées, la plus mauvaise solution est autour 77% loin de l'optimum. La segmentation du temps fournit des solutions bien meilleures. Pour les mêmes instances, l'écart d'optimalité est moins de 6.5%. D'autres chercheurs ont utilisé des méthodes heuristiques notamment celle de la population qui est très efficace et a été employée pour résoudre des problèmes de plus grandes tailles impliquant jusqu'à 500 avions. Tous les algorithmes exacts ([11] et [43]) emploient la méthode branch-and-bound pour trouver la solution optimale en utilisant la programmation en nombres entiers pour les (ALP). Cependant, en employant cette méthode, la durée de résolution croît exponentiellement avec la taille de problème. Par conséquent, ces algorithmes ne peuvent pas résoudre dans un temps acceptable les problèmes de très grande échelle de façon optimale.

2.4 Algorithme génétique

2.4.1 Définition des paramètres

Nous utilisons le codage du chromosome suivant :

9	3	1	4	6	7	2	5	8
X_9	X_3	X_1	X_4	X_6	X_7	X_2	X_5	X_8

TAB. 2.1: Le codage du chromosome

Un individu (une solution) est codé par un tableau de deux lignes et de P colonnes. La première ligne détermine le séquençement d'atterrissage d'avions : l'avion 9 atterrit avant l'avion 3 et ainsi de suite. La deuxième ligne contient l'heure effective d'atterrissage. Ainsi, l'avion 9 atterrit à l'heure X_9, \dots , l'avion 8 atterrit à l'heure X_8 .

2.4.2 Génération de la population initiale

Tout d'abord, nous commençons par générer trois individus selon l'ordre croissant des heures au plus tôt, des heures préférentielles et des heures au plus tard. Nous complétons la population initiale par des individus générés aléatoirement.

2.4.3 Calcul des heures d'atterrissage des avions

Notre algorithme génétique fonctionne sur les permutations des numéros des avions et pour décider de l'heure d'atterrissage de chaque avion, nous utilisons les trois méthodes suivantes :

- **Calcul déterministe initial des heures d'atterrissage des avions (II.1)**

SEQ une séquence d'atterrissage fixée ;
 $i \leftarrow 0$;
 $X_{SEQ(i)} = T_{SEQ(i)}$;
Répéter
 $i \leftarrow i + 1$;
 $DatePoss[i] = \text{Max}((X_{SEQ(j)} + SEP_{SEQ(j),i}), j = 0, \dots, (i - 1))$;
 $X_{SEQ(i)} = \text{Min}(L_{SEQ(i)}, \text{Max}(E_{SEQ(i)}, DatePoss[i]))$;
jusqu'à ce que ($i=P-1$)

- **Calcul aléatoire initiale des heures d'atterrissage des avions (II.2)**

```

SEQ une séquence d'atterrissage fixée ;
i ← 0 ;
XSEQ(i) = TSEQ(i) ;
Répéter
    i ← i + 1 ;
    DatePoss[i] = Max((XSEQ(j) + SEPSEQ(j)i), j = 0, ..., (i - 1)) ;
    Si (TSEQ(i) < DatePoss[i]) Alors
        XSEQ(i) = DatePoss[i] + y * (LSEQ(i) - DatePoss[i]) ;
    Sinon
        XSEQ(i) = TSEQ(i) + y * (LSEQ(i) - TSEQ(i)) ;
    Fin Si
jusqu'à ce que (i=P-1)

```

Avec :

- $y \in [0, 1]$ choisi aléatoirement ;
- Le coût total de pénalités est calculé par la formule (2.4).
- **Amélioration du calcul des X_{SEQ} (III)**

```

Meilleur ← X ; Fmeilleur ← FX ;
DkAv=DkAp=0;
Pour i de 0 à P-1 faire
    Si (  $X_{SEQ(i)} < T_{SEQ(i)}$  ) Alors
        Alpha =  $T_{SEQ(i)} - X_{SEQ(i)}$  ;
        Si ( Alpha > DkAv ) Alors
            DkAv=Alpha
        Fin Si
    Sinon
        Alpha =  $X_{SEQ(i)} - T_{SEQ(i)}$  ;
        Si ( Alpha > DkAp ) Alors
            DkAp=Alpha
        Fin Si
    Fin Si
Fin Pour
DkAv=Min{DkAv, Min{( $T_i - E_i$ ), i=0,...,(P-1)}} ;
DkAp=Min{DkAp, Min{( $L_i - T_i$ ), i=0,...,(P-1)}} ;
Si ( DkAv > 0 ) Alors
    Pour i de 1 à DkAv faire
        Pour j de 0 à P-1 faire
             $Y_{SEQ(j)} = X_{SEQ(i)} + i$  ; Calcul de FY=F(Y) ;
            Si ( FY < Fmeilleur ) Alors
                Meilleur =Y et Fmeilleur=FY
            Fin Si
        Fin Pour
    Fin Pour
Fin Si
Si ( DkAp > 0 ) Alors
    Pour i de 1 à DkAp faire
        Pour j de 0 à P-1 faire
             $Y_{SEQ(j)} = X_{SEQ(i)} - i$  ; Calcul de FY ;
            Si ( FY < Fmeilleur ) Alors
                Meilleur =Y et Fmeilleur=FY ;
            Fin Si
        Fin Pour
    Fin Pour
Fin Si
X<- Meilleur et FX<-Fmeilleur ;
    
```

La variable SEQ n'est autre que la première ligne du chromosome qui donne l'ordre d'atterrissage des avions sur la piste de l'aéroport. Vous remarquez que la première méthode (II.1) de calcul de X_i est purement déterministe et veille à ce que la formule (2.3) soit

respectée. Elle correspond à la contrainte de séparation entre deux heures d'atterrissage. La deuxième méthode (II.1) de calcul des heures d'atterrissage des avions introduit la variable aléatoire y qui permet d'obtenir des heures d'atterrissage aléatoires sauf pour la première avion en respectant la contrainte de séparation (2.3). La méthode (III) essaie d'ajuster au mieux les X_i que nous avons trouvé par le biais de l'une des deux précédentes méthodes de calcul d'atterrissage des avions.

2.4.4 Opérateur de croisement

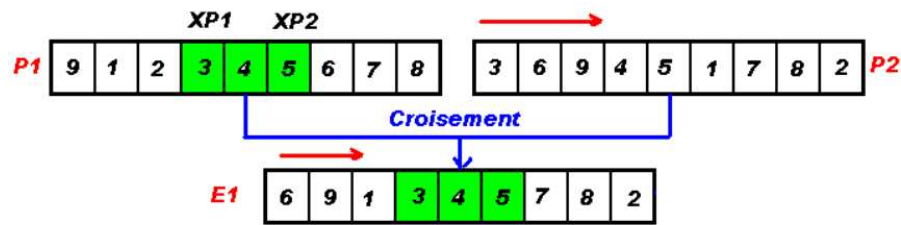


FIG. 2.1: Opérateur de croisement.

Il consiste à copier dans l'enfant E1 la séquence délimitée par XP1 et XP2 du parent P1 dans le même emplacement et de le compléter en commençant par les premières cases vides de cet enfant par les premiers éléments du parent P2 et en sautant les numéros d'avion qui ont été déjà casés dans l'enfant E1. De la même manière, nous construisons l'enfant E2 en intervertissant le rôle des parents.

2.4.5 Opérateur de mutation

Notre opérateur de mutation appliqué avec une probabilité de mutation égale à 0.5 ;

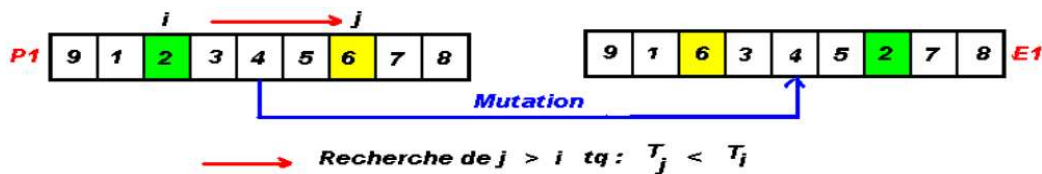


FIG. 2.2: Opérateur de mutation.

Il consiste à chercher aléatoirement deux emplacements i et j tels que $i < j$ et $T_i > T_j$ dans l'individu et les échanger.

2.4.6 Organigramme de notre algorithme génétique

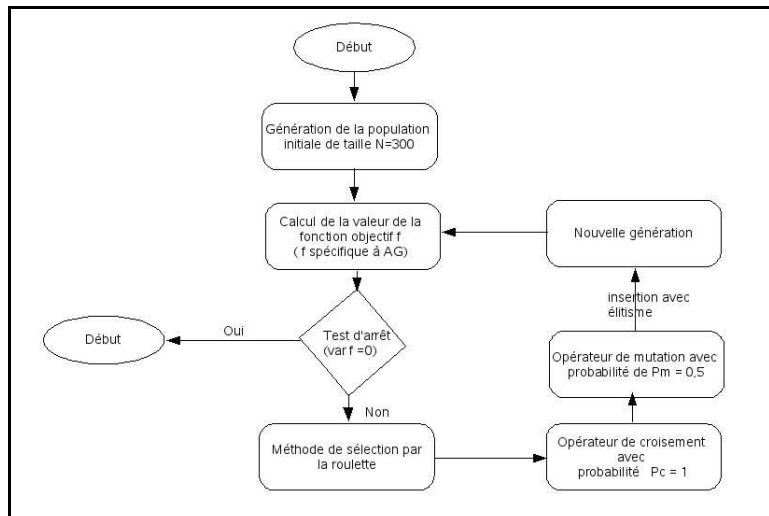


FIG. 2.3: L'organigramme de notre algorithme génétique pour la résolution du ALP avec une seule piste.

Notre algorithme génétique est caractérisé par les propriétés suivantes :

- La taille de la population est fixée à 300 chromosomes ;
- La méthode de sélection est la roulette ([53]) ;
- L'opérateur de croisement (2.4.4) est appliqué avec une probabilité égale à 1 ;
- L'opérateur de mutation 2.4.5 est appliqué avec une probabilité égale à 0.5 ;
- Méthode d'insertion est l'élitisme ([53]) ;
- L'A.G s'arrête lorsque la population est constituée de chromosomes identiques.

Nous fixons les opérateurs de l'algorithme génétique et nous changerons la manière de calculer la fonctions objectif. Nous avons obtenu quatre variantes d'algorithme génétique :

- (A.G.1) dont la fonction objectif : (II.1) +(2.4)+(III)
- (A.G.2) avec la fonction objectif : (II.2) +(2.4)+(III)
- L'algorithme (A.G.3) prend la solution obtenue par l'algorithme (A.G.1) et essaye de l'améliorer en utilisant le mécanisme (II.2) du calcul de la fonction objectif. Ces améliorations sont répétées et à chaque fois qu'il trouve une meilleure solution, il la sauvegarde et tente de la perfectionner jusqu'à l'épuisement de 100000 itérations.
- L'algorithme (A.G.4) ressemble à l'algorithme (A.G.3) sauf en un point. Il travaille sur la solution obtenue par l'algorithme génétique (A.G.2).

2.5 Résultats numériques

Nous présenterons dans ce tableau les valeurs moyennes de dix exécutions de ces A.G appliqués sur les instances de Beasley ([4]) :

Instances	P	A.G.1	A.G.2	A.G.3	A.G.4
Airland1	10	1150	700	<i>700</i>	<i>700</i>
Airland2	15	1720	1750	<i>1700</i>	1750
Airland3	20	1610	1180	<i>1020</i>	1040
Airland4	20	4480	4520	<i>4480</i>	4520
Airland5	20	4800	4840	<i>4800</i>	4840
Airland6	30	27984	27194	27984	<i>27194</i>
Airland7	44	2082	1550	1550	<i>1550</i>
Airland8	50	3840	3365	3195	<i>3050</i>
Airland9	100	9980	9890,88	9520	<i>9654,58</i>
Airland10	150	30153	25973,9	<i>24415</i>	25452,8
Airland11	200	25715	22011	21671	<i>21509</i>
Airland12	250	33181	29538	<i>27983</i>	28946,3

TAB. 2.2: Tableau des résultats numériques pour une piste.

La meilleure valeur entre A.G.1 et A.G.2 est présentée en gras. Alors que les meilleures valeurs obtenues pour chaque instance sont écrites en italique.

D'après les simulations numériques, nous pouvons confirmer les observations suivantes :

1. A.G.2 donne des solutions meilleures que A.G.1 pour la plupart des instances et surtout pour celles ayant le nombre d'avions dépassant 30.
2. A.G.3 a permis d'améliorer significativement A.G.1 en diminuant la fonction objectif au plus bas de ces niveaux pour neuf instances sur douze. Parmi les neuf instances, 2/3 sont des instances de nombre d'avions supérieure à 44 avions.
3. A.G.4 a une chance sur deux pour améliorer A.G.2 et toutes les instances ayant un nombre pour les tailles des instances inférieures à 30 et reste équivalent à A.G.3 sauf pour l'instance airland 6.
4. Pour les instances de taille supérieure à 100, A.G.3 est meilleure que tous les autres A.G. sauf pour airland11 où il est dépassé par A.G.4.
5. A.G.2, en moyen, est plus lent que A.G.1.
6. A.G.3 est plus rapide que A.G.2.

En conclusion, nous pouvons constater que l'algorithme (A.G.3) est l'algorithme le plus stable par rapport aux autres et donne de bons résultats.

2.6 Algorithmes de population heuristique

Les algorithmes de population sont basés sur les principes fondamentaux de la théorie d'évolution. Il regroupe tous les algorithmes qui fonctionnent sur un ensemble de solutions. Ils intègrent une méthode de codage de solutions, une méthode de sélection, les opérateurs de croisement, une méthode d'insertion des individus générés et optionnellement des opérateurs de mutation. Tout ceci est contrôlé par un test d'arrêt. Dans cette grande famille,

nous nous intéressons en plus de l'algorithme génétique à deux autres algorithmes Scatter search et Bionomic algorithm.

2.6.1 Algorithme génétique (AG)

L'algorithme génétique que nous emploierons dans cette section est différent du précédent en plusieurs points. D'abord, au niveau du codage (2.3), il utilise un tableau de trois lignes de même longueur égale au nombre d'avions. Dans la deuxième ligne, il y a des proportions réelles y_i qui permettent d'obtenir l'heure d'atterrissage de l'avion i en utilisant la formule (2.5) :

$$X_i = E_i + y_i * (L_i - E_i) \text{ où } y_i \in [0, 1] \tag{2.5}$$

La troisième ligne est consacrée à la désignation du piste d'atterrissage par un entier entre 1 et R , nous nous contentons d'utiliser une seule piste.

9	3	1	4	6	7	2	5	8
y_9	y_3	y_1	y_4	y_6	y_7	y_2	y_5	y_8
r_9	r_3	r_1	r_4	r_6	r_7	r_2	r_5	r_8

TAB. 2.3: Le codage du chromosome pour cet algorithme génétique associé au ALP

Ensuite, au niveau de la population initiale qui est constituée de 100 chromosomes dont trois sont choisis selon l'ordre croissant de l'heure d'arrivée au plus tôt (E_i), l'heure prévue d'arrivée (T_i) et l'heure au plus tard (L_i) avec des y_i qui vérifie la formule suivante : $y_i = \frac{(T_i - E_i)}{(L_i - E_i)}$ afin que leurs heures d'atterrissage soient égales à leurs heures prévues T_i . Les autres individus sont constituées de permutations des avions et des proportions y_i générées aléatoirement.

Après, au niveau de la fonction d'évaluation car les chromosomes construits ainsi peuvent ne pas vérifier la contrainte de séparation entre les avions. Pour cela nous utilisons :

$$\sum_{i \in I} (Av_i \times \max\{0, T_i - X_i\} + Ap_i \times \max\{0, X_i - T_i\}) + 10^6 \times \sum_{i \in I} \sum_{\substack{j \in I \setminus \{i\} \\ X_j \geq X_i}} \max(0, SEP_{ij} - (X_j - X_i)) \tag{2.6}$$

Cette fonction de pénalisation pousse l'algorithme à réduire le nombre d'individus dans la population qui ne respectent pas la contrainte de séparation. Le coefficient de 10^6 a été choisis par rapport aux résultats que nous avons obtenu dans la section précédente.

Ensuite, au niveau de l'opérateur de croisement qui est appliqué avec une probabilité égale à 1 :

<i>parent1</i>	2	3	5	1	4	6
	y_2	y_3	y_5	y_1	y_4	y_6
	r_2	r_3	r_5	r_1	r_4	r_6

<i>parent2</i>	1	2	3	4	5	6
	y'_1	y'_2	y'_3	y'_4	y'_5	y'_6
	r'_1	r'_2	r'_3	r'_4	r'_5	r'_6

⇒

<i>Enfant1</i>	2	3	5	4	1	6
	$\frac{(y_2+y'_2)}{2}$	$\frac{(y_3+y'_3)}{2}$	$\frac{(y_5+y'_5)}{2}$	$\frac{(y_4+y'_4)}{2}$	$\frac{(y_1+y'_1)}{2}$	$\frac{(y_6+y'_6)}{2}$
	r_2	r_3	r_5	r'_4	r'_1	r'_6

<i>Enfant2</i>	1	2	3	5	4	6
	$\frac{(y_1+y'_1)}{2}$	$\frac{(y_2+y'_2)}{2}$	$\frac{(y_3+y'_3)}{2}$	$\frac{(y_5+y'_5)}{2}$	$\frac{(y_4+y'_4)}{2}$	$\frac{(y_6+y'_6)}{2}$
	r'_1	r'_2	r'_3	r_5	r_4	r_6

Notre opérateur de croisement ressemble à l'opérateur de croisement en un point. En effet, il coupe chaque parent en deux parties en une position choisie au hasard.

La première ligne de l'enfant1 est constituée de deux parties; la première partie copiée telle qu'elle est du parent1 Alors que la deuxième partie est prise du parent2. Dans la deuxième partie, il y aura des gènes doubles que nous les corrigeons en utilisant le parent2; le premier gène doublé est remplacé par le premier gène du parent2 qui n'a pas été placé dans l'enfant1 et le deuxième gène doublé est changé par le deuxième du parent2 qui n'a pas été mis dans l'enfant1 et ainsi de suite.

La deuxième ligne de l'enfant1 est constituée de la proportion moyenne obtenue à partir des proportions associées à chaque avion dans chaque parent.

La troisième ligne de l'enfant1 est composée des entiers indiquant les pistes d'atterrissage des avions. Les avions ramènent avec eux leurs pistes d'origines du parent2 ou du parent1; l'avion qui vient du parent1 a la même qu'il a dans parent1.

L'enfant2 est construit de la même manière que l'enfant1 en intervertissant les rôles des parents.

Et enfin, au niveau de l'opérateur de mutation qui est appliqué avec une probabilité de 0.5 :

2	3	5	1	4	6
y_2	y_3	y_5	y_1	y_4	y_6
r_2	r_3	r_5	r_1	r_4	r_6

====>

2	1	5	3	4	6
y_2	y'_1	y_5	y'_3	y_4	y_6
r_2	r_1	r_5	r_3	r_4	r_6

avec y'_i est un nombre réel appartenant à l'intervalle $[0,1]$.

Nous choisissons aléatoirement deux avions que nous échangeons leurs emplacements. Les proportions y_i associées à ces avions sont régénérées aléatoirement.

2.6.2 Méthode de recherche Scatter (SS)

La méthode Scatter search se base sur des anciennes stratégies qui combinent les règles de décision et les contraintes du problème. Elle a pour but de permettre à une procédure basée sur des éléments combinés de rapporter meilleures solutions que celles issues seulement des éléments originaux. Historiquement, ces stratégies ont été présentées lors de résolution du problème d'ordonnancement de Job Shop en minimisant le makespan par Glover [50]. De nouvelles règles ont été produites en créant les combinaisons pondérées des règles existantes, convenablement restructurées pour que leurs évaluations aient incarné une métrique commune.

Cette approche est motivée par la supposition que l'information désirée peut être obtenue par différents itinéraires en utilisant différentes règles. L'idée est d'exploiter un mécanisme qui intègre une combinaison de ces différentes règles afin d'obtenir rapidement l'information désirée. Intuitivement, cette stratégie semble meilleure que celle qui sélectionne à chaque fois une seule règle particulière isolée des autres. Par analogie, les règles ici sont les individus et la méthode prévoit de choisir un ensemble de parent qui sera l'objet de l'opérateur de croisement linéaire.

L'algorithme de la méthode est présentée ci-dessous :

générer une population initiale appelée une population de référence ;
améliorer chaque individu de la population de référence ;
Répéter
 sélectionner aléatoirement un sous-ensemble de l'ensemble de référence ;
 créer un nouvel individu en utilisant le croisement linéaire ;
 améliorer le nouvel individu ;
 mise-à-jour de l'ensemble de référence et du meilleur individu ;
jusqu'à ce que (satisfaction du critère d'arrêt)

Algorithme 11: L'algorithme SS

Pour améliorer l'individu, nous utilisons la fonction suivante :

$\pi(s)$ est l'ordre d'atterrissage de la $s^{\text{ème}}$ avion sur une piste donnée
 $s = 1$;
 $X_{\pi(1)} = E_{\pi(1)}$;
Répéter
 $s = s + 1$;
 $X_{\pi(s)} = \min[L_{\pi(s)}, \max[E_{\pi(s)}, X_{\pi(t)} + S_{\pi(t)\pi(s)} \ t = 1, \dots, s - 1]]$;
jusqu'à ce que (tous les avions dans la séquence soient visités)

Algorithme 12: Méthode d'amélioration

L'opérateur de combinaison linéaire qui permet de profiter de l'apport de chaque vecteur est réalisé de la manière suivante :

Soient $p(1), p(2), \dots, p(K)$ les parents sélectionnés dans l'ensemble des parents ;
 générer K nombres $w_i, i = 1, \dots, K$, aléatoirement entre 0 et 1 ;
 $W = w_1 + \dots + w_K$;
 $\alpha_i = \frac{w_i}{W}$ for $i = 1, \dots, K$;
 $y_{child}^k = \sum_{i=1}^{i=K} \alpha_i y_k \ \forall k \in \{1, \dots, P\}$;
 répéter pour chaque avion ;

Algorithme 13: Opérateur de combinaison linéaire

2.6.3 Algorithme Bionomique (BA)

Cet algorithme, moins connu que Scatter Search, est présenté pour la première fois en 1994 par Christofides [32]. Comme la méthode Scatter Search, il utilise l'opérateur de croisement linéaire qui fonctionne sur un ensemble de parents, choisi soigneusement de telle manière à représenter la structure de la population. Après, il procède à améliorer chaque individu engendré. L'algorithme suivant présente ses principales étapes :

```
généraler la population initiale ;
améliorer chaque individu dans la population initiale ;
Répéter
    construire un graphe représentant la structure de la population ;
    calculer l'ensemble des parents à partir de ce graphe ;
    créer un nouvel individu comme combinaison linéaire de tout les éléments de
    l'ensemble des parents ;
    améliorer chaque nouvel individu ;
    mise-à-jour de la population et du meilleur individu
jusqu'à ce que (satisfaction du critère d'arrêt)
```

Algorithme 14: algorithme BA

Pour créer l'ensemble des parents, nous construisons tout d'abord, le graphe qui caractérise la population en procédant de la manière suivante :

```
chaque individu représente un noeud ;
Si (distance entre les fitness de deux noeuds  $< d$ ) Alors
    il y a une arête entre les deux noeuds ;
Sinon
    il n'y a pas d'arrête entre les deux noeuds ;
Fin Si
```

Ensuite, nous utilisons l'algorithme suivant afin de construire l'ensemble du parent qui sera assujetti à l'opérateur de croisement linéaire :

```
Répéter
    sélectionner un noeud au hasard ;
    insérer le noeud dans l'ensemble du parents ;
    supprimer tout les noeuds sélectionnés et leurs voisins du graphe ;
jusqu'à ce que (le graphe est vide)
```

2.7 Notre algorithme de population

La population est composée de trois sous populations dont chacune est constituée de 100 individus. Nous appliquons sur chacune une des trois méthodes de population. Les deux sous-populations initiales pour les algorithmes Scatter et Bionomiques sont construites de la même façon que la sous-population initiale pour l'algorithme génétique. Pour la méthode de recherche Scatter, nous fixons de la taille de la sous-population à 10 individus sélectionnés aléatoirement. Alors que nous prenons $d = 10 \times \frac{F_{max}-F_{min}}{100}$ pour la création du graphe qui constitue le premier pas dans l'algorithme Bionomique. Où F_{max} (resp. F_{min}) est la valeur maximum (resp. minimum) de la fitness trouvée dans la population courante. La fonction d'évaluation pour les trois algorithmes est la même (2.6). L'algorithme génétique utilise en plus des opérateurs de croisement et de mutation cités ci-dessus, une méthode de sélection du tournoi binaire (choisir le meilleur parmi deux), $P_x = 1$, $P_m = 0.5$ et la méthode d'insertion élitisme. À la fin de chaque itération, nous copions le meilleur individu dans les trois sous-populations à la place du mauvais individu. Nous avons fixé le critère d'arrêt à un nombre maximum d'itérations égale à 10 000. Il va sans dire que les trois méthodes s'exécutent en parallèle.

2.8 Résultats

Les valeurs moyennes obtenues de dix exécutions de notre algorithme de population sont enregistrés dans ce tableau : Pour les instances airland1 et airland7 l'algorithme

Instances	P	Meilleur valeur du tableau 2.2		Notre algorithme de population
		valeur	Algorithme	
Airland1	10	700	2, 3, 4	700
Airland2	15	1700	3	1480
Airland3	20	1020	3	820
Airland4	20	4480	1, 3	2520
Airland5	20	4800	1, 3	3100
Airland6	30	27194	2, 4	25556
Airland7	44	1550	2, 3,4	1550
Airland8	50	3050	4	1960
Airland9	100	9520	3	5700
Airland10	150	24415	3	12330
Airland11	200	21509	4	12500
Airland12	250	27983	3	16331

TAB. 2.4: Comparaison des résultats de notre algorithme de population avec ceux du tableau 2.2.

génétique (A.G.3) et l'algorithme de population trouvent la même valeur optimale. Pour les autres instances, nous remarquons que l'algorithme de population a obtenu les meilleurs

résultats par rapport à la meilleure valeur moyenne obtenue par les algorithmes génétiques de la section précédente.

2.9 Conclusion

Dans ce chapitre, nous nous sommes focalisés sur la résolution du problème d'atterrissage de P avions sur une piste de l'aéroport (ALP). Le problème est simple lorsque les temps prévus d'atterrissage des avions respectent les contraintes de séparation SEP_{ij} et les fenêtres de temps d'atterrissage $[E_i, L_i]$. L'objectif escompté est d'obtenir une séquence d'avions avec leurs heures d'atterrissage qui minimise la fonction (2.4).

Pour cela, nous avons imaginé deux façons de faire :

- .1. algorithme génétique qui dépendait de sa fonction objectif;
- .2. algorithme de population.

Pour la première, nous avons utilisé un algorithme génétique qui emploie un tableau de deux lignes et P colonnes pour coder les solutions. La première ligne concerne les numéros des avions et permettent d'obtenir leur ordre d'atterrissage. La deuxième ligne nous donne l'heure d'atterrissage de chaque avion X_i . L'idée de base est d'employer un algorithme génétique semblable à celui que nous avons présenté pour la résolution de T.S.P. qui fonctionnent sur les permutations afin de trouver l'ordre selon lequel les avions atterrissent et nous calculons les X_i en utilisant trois méthodes :

- **Calcul déterministe initial des heures d'atterrissage des avions (II.1) :** C'est une méthode purement déterministe qui initialise X_i en respectant la contrainte de séparation entre deux heures d'atterrissage (2.3).
- **Calcul aléatoire initial des heures d'atterrissage des avions (II.2) :** Elle permet d'obtenir des heures d'atterrissage totalement aléatoire sauf pour la première avion en respectant la contrainte de séparation (2.3).
- **Amélioration du calcul des X_{SEQ} (III) :** C'est une méthode heuristique qui permet d'ajuster au mieux les décalages entre X_i (calculer par l'une des méthodes (II.1) ou (II.2)) et T_i de telle manière à trouver celle qui minimise la fonction (2.4).

Et nous avons confectionné quatre algorithmes génétiques :

- (A.G.1) dont la fonction objectif : (II.1) +(2.4)+(III)
- (A.G.2) avec la fonction objectif : (II.2) +(2.4)+(III)
- (A.G.3) algorithme génétique qui prend la solution de (A.G.1) et essaye de l'améliorer par la méthode (III).
- (A.G.4) perfectionne la solution de l'algorithme (A.G.3) en utilisant la méthode d'amélioration (III).

Ces quatre algorithmes ont les caractéristiques communes suivantes :

- La taille de la population est fixée à 300 chromosomes;
- La population initiale est composée de trois individus générés selon l'ordre croissant des heures au plus tôt, des heures préférentielles et des heures au plus tard. Et

- complétée par des individus générés aléatoirement ;
- La méthode de sélection est la roulette ([53]) ;
 - Notre opérateur de croisement (2.4.4) est appliqué avec une probabilité égale à 1 ;
 - Notre opérateur de mutation 2.4.5 est appliqué avec une probabilité égale à 0.5 ;
 - Méthode d’insertion est l’élitisme ([53]) ;
 - Ils s’arrêtent lorsque la population est constituée de chromosomes identiques.

Les dix exécutions faites sur chaque instance de Beasley ([4]) montrent que l’algorithme génétique (A.G.3) donnent de bons résultats par rapport aux trois autres.

Pour la deuxième façon, nous avons utilisé trois algorithmes de population :

- **algorithme génétique** caractérisé par son opérateur de croisement, créer spécialement pour le problème ALP, appliqué avec $P_x = 1$ et son opérateur de mutation $P_m = 0.5$ et la méthode de sélection tournoi binaire qui choisit le meilleur parmi deux individus prélevés aléatoirement ;
- **algorithme de recherche *Scatter*** s’illustre par la constitution d’une sous population de 10 éléments sélectionnés aléatoirement afin de créer un individu par l’opérateur de croisement linéaire qui sera amélioré et inséré dans la nouvelle population ;
- **algorithme *Bionomique*** qui se distingue par la création d’un graphe de la population qui servira par la suite pour la construction de l’ensemble des parents qui sera assujetti à l’opérateur de croisement linéaire afin de fabriquer un enfant qui sera amélioré et insérer dans la nouvelle population.

Pour profiter de la collaboration de ces algorithmes. Nous avons employé le même type de codage et la même fonction d’évaluation. En effet, pour le codage, nous avons utilisé des tableaux de trois lignes et P colonnes ; la première ligne donne la séquence d’atterrissage des avions alors que la deuxième ligne est constituée de proportions y_i qui permettent d’obtenir l’heure d’atterrissage X_i en utilisant la formule (2.5). Pour évaluer ces chromosomes, nous nous sommes servis de la fonction objectif pénalisée(2.6) qui octroie aux individus qui ne respectent pas les contraintes de séparation des grandes valeurs.

Notre algorithme de population est constitué d’une population initiale composée de trois sous-populations de 100 individus chacune. Elles sont créées de manière semblable pour chaque algorithme ; trois individus sont choisis selon l’ordre croissant de l’heure d’arrivée au plus tôt (E_i), l’heure prévue d’arrivée (T_i) et l’heure au plus tard (L_i) avec des y_i qui vérifient que les heures d’atterrissage sont égales aux heures prévues T_i . Les autres individus sont constituées de permutations des avions et des proportions y_i générées aléatoirement. Sur chaque sous-population, nous appliquons un algorithme. La meilleure solution obtenue parmi les trois est insérée dans les nouvelles sous- populations à la place du plus mauvais ce qui fait une itération et nous la répétons jusqu’à la réalisation de 10 000 itérations.

La collaboration de ces algorithmes s’avère payante et donne les meilleurs résultats pour toutes les instances de Beasley ([4]) par rapport aux algorithmes (A.G.3) et (A.G.4).

Chapitre 3

Ordonnancement des véhicules dans une chaîne de production

Sommaire

3.1	Description du problème de séquençement des véhicules dans une chaîne de fabrication	137
3.1.1	Processus d'ordonnancement	137
3.1.2	Les contraintes du problème	138
3.1.3	Le problème à résoudre	139
3.1.4	Comptabilisation des violations de ratio	140
3.2	Les instances	141
3.3	Algorithmes génétiques	142
3.3.1	Méthode pas-à-pas	142
3.3.2	L'algorithme génétique avec une fonction d'évaluation pondérée	146
3.4	Résultats numériques	146
3.5	Conclusion	148

3.1 Description du problème de séquençement des véhicules dans une chaîne de fabrication

3.1.1 Processus d'ordonnancement

Quotidiennement, les commandes clientes sont transmises en temps réel dans les usines d'assemblage de véhicules.

La tâche quotidienne des usines est :

1. d'affecter une journée de fabrication à chaque véhicule commandé en fonction de contraintes capacitaires des lignes de fabrication et des délais promis aux clients ;

2. d'ordonnancer les véhicules à l'intérieur de chaque journée de fabrication, en satisfaisant au mieux les besoins respectifs des ateliers de la ligne de fabrication : ateliers de tôlerie, peinture et montage. La séquence de véhicules calculée constitue le « film » qui sera envoyé à la fabrication.

Nous nous focaliserons sur les besoins des ateliers de peinture et de montage, considérant que l'atelier de tôlerie ne génère pas d'impératifs sur l'ordonnancement de chaque journée.

Lors de l'ordonnancement de chaque journée, nous ne remettons plus en cause la composition des journées décidée à l'étape (1).

3.1.2 Les contraintes du problème

Contraintes de l'atelier peinture

L'atelier peinture utilise un solvant très coûteux pour purger les pistolets de peinture à chaque changement de teinte dans le film de véhicules. L'objectif est de minimiser la consommation de ce solvant en cherchant donc à « grouper » les véhicules de même teinte, ce qui revient à minimiser le nombre de purges, ce qui revient encore à générer les rafales de peinture les plus longues possibles.

Mais chaque rafale de teintes ne doit pas dépasser une longueur maximale, car il faut périodiquement purger les pistolets de peinture, même s'il n'y a pas eu de changements de teinte. C'est une contrainte dure.

Contraintes de l'atelier montage

Afin de lisser la charge du travail sur les différents postes en bord de chaîne, nous cherchons à « écarter » les véhicules, dites « difficiles », ayant des équipements nécessitant des opérations lourdes. En d'autres termes, nous voulons limiter la densité des véhicules difficiles pour ne pas surcharger les postes de travail concernés par l'assemblage de ces véhicules.

Ce besoin d'écartement est formalisé par *une contrainte de ratio N/P*. La contrainte porte sur une caractéristique technique qui nécessite des opérations lourdes (le toit ouvrant, la climatisation, etc).

La contrainte de ratio N/P signifie qu'on ne doit pas trouver une densité supérieure à N/P de véhicules concernés par la contrainte de ratio.

Par exemple,

- si $N/P = 3/5$, nous ne devons pas trouver plus de 3 véhicules concernés par la contrainte de ratio dans toute fenêtre de longueur de 5 véhicules dans le film ;
- si $N/P = 1/P$, cela signifie que chaque deux véhicules concernés par cette contrainte de ratio doivent être séparés par au moins (P-1) positions dans le film.

L'entreprise définit, en plus des contraintes de ratio prioritaires, des contraintes de ratio non prioritaires.

Ces dernières sont associées à des opérations plus légères et leur satisfaction passe en second rang derrière les contraintes de ratio prioritaires. En d'autres termes, le respect

d'un ratio associé à une opération légère ne se fasse pas au détriment d'un ratio associé à une opération lourde.

Les contraintes de ratio sont des contraintes molles. En effet, lorsqu'on ordonnance une journée de fabrication, nous ne savons pas a priori si l'ensemble des contraintes de ratio est respecté. L'objectif de l'optimisation consiste donc à minimiser le nombre de violations des contraintes de ratio.

Les instances de problèmes fournis avec des contraintes de ratio prioritaires sont soit « faciles » soit « difficiles ». Ce qui n'est pas le cas pour les contraintes de ratio non prioritaires.

Une instance de problème est considérée comme ayant des contraintes de ratio prioritaires faciles si le solveur de l'entreprise parvient à construire une séquence ne produisant aucune violation sur l'ensemble des contraintes de ratio prioritaires. Les contraintes de ratio prioritaires faciles sont donc réalisables.

Une instance de problème est considérée comme ayant des contraintes de ratio prioritaires difficiles si le solveur de l'entreprise ne parvient pas à construire de séquence ne produisant aucune violation sur l'ensemble des contraintes de ratio prioritaires. C'est le cas notamment des instances avec des ratios supérieurs au nombre maximum de véhicules. Par exemple, la journée à ordonnancer contient 25% de véhicules d'une certaine caractéristique technique lourde et on impose une contrainte de ratio prioritaire de 1/5.

Rôle de la journée J-1

Lorsque nous ordonnancions les véhicules de la journée J, nous ne touchons pas les véhicules restantes de la journée (J-1). Ils sont considérés comme déjà séquencés et que leurs positions sont figées et fixées dans le film de la journée J. Il faut donc tenir compte des derniers véhicules de la journée (J-1) pour compter les violations de contraintes de ratio sur la journée J. En revanche, les véhicules de la journée (J+1) ne sont pas prises compte.

3.1.3 Le problème à résoudre

Il s'agit de construire une séquence de véhicules satisfaisant les besoins de l'atelier peinture et ceux de l'atelier montage. Il y a deux stratégies possibles à adopter selon les usines :

contraintes de ratio en priorité : les besoins de l'atelier montage sont prioritaires par rapport à ceux de l'atelier peinture

rafales de peinture en priorité : les besoins de l'atelier peinture sont prioritaires par rapport à ceux de l'atelier montage.

Pour les deux stratégies d'optimisation, la seule contrainte « dure » du problème à respecter obligatoirement est le respect de la longueur maximale des rafales de peinture.

L'entreprise a demandé de ne pas y avoir de compensation entre les fonctions objectifs. Ainsi, la résolution de la deuxième fonction objectif ne doit pas dégrader le résultat de la première fonction objectif.

3.1.4 Comptabilisation des violations de ratio

Pour chaque contrainte de ratio N/P , nous cherchons à obtenir que toute fenêtre glissante, de taille P dans le film, ne doit pas contenir plus de N véhicules concernés par la contrainte de ratio.

Dans le cas d'une contrainte de ratio du type $1/P$, nous visons à avoir un film où deux véhicules concernés par la contrainte de ratio sont distants d'au moins $(P-1)$ positions :

X _ _ _ X pour un ratio de $1/4$.

Si nous ne parvenons pas à obtenir zéro violations sur cette contrainte de ratio, nous cherchons à « écarter » au maximum les véhicules concernés par la contrainte. Il s'agit de lisser au mieux la charge de travail sur les postes de travail contraints, c.-à.-d. les postes à l'origine d'une contrainte de ratio.

Nous essayons d'obtenir cet « écartement » en comptabilisant les contraintes de ratio sur des fenêtres glissantes sur tout le film. Moins les véhicules seront écartés et plus nombreuses seront les violations, car la violation avec des véhicules moins écartés sera comptabilisée dans davantage de fenêtres glissantes, et le nombre de violations pourra être plus élevé dans chaque fenêtre glissante.

Exemple :

- Supposons que nous avons la contrainte de ratio de $1/5$, nous comptabilisons les violations avec des fenêtres glissantes de taille 5.

_ _ _ X _ _ _ X

Sur ce film de véhicules, il n'y a qu'une seule violation sur la fenêtre glissante X _ _ _ X.

- Supposons que nous avons une contrainte de ratio de $1/5$.

_ _ X X _ _ X

Il y a une violation sur la fenêtre glissante _ _ X X _ et sur la fenêtre glissante X X _ _ X, nous comptabilisons deux violations : 3 (= véhicules concernés par la contrainte de ratio) - 1 (= numérateur du ratio).

En général, nous utilisons la formule suivante pour calculer le nombre de violation sur une fenêtre glissante ($NbViolFenGliss$) :

$$NbViolFenGliss = Max\{0, (NbrVeFenGliRat) - (\text{numérateur du ratio})\} \quad (3.1)$$

avec $NbrVeFenGliRat$ c'est le nombre de véhicules concernés par le ratio dans la fenêtre glissante de taille P .

Cette méthode de comptabilisation s'applique à toutes les contraintes de ratio N/P ($\forall(N, P) \in \mathbf{N}^* \times \mathbf{N}^*$).

L'objectif consistera donc à minimiser le nombre de violations des contraintes de ratio sur l'ensemble des fenêtres glissantes sur le film, et ce pour l'ensemble des contraintes de ratio.

Il ne faut pas oublier que les véhicules de la journée (J-1), déjà séquencés, doivent être pris en compte dans le calcul de violation de contraintes ratio. Pour cela, les fenêtres glissantes doivent débiter dans la journée (J-1). Ainsi, pour un ratio N/P, la 1^{ère} fenêtre glissante contient les (P-1) derniers véhicules figés de la journée (J-1) et le 1^{er} véhicule de la journée J.

Ainsi, pour calculer le nombre total de violation d'un ratio N/P, nous utilisons une fenêtre de longueur P véhicules et nous commençons le comptage par la formule (3.1) en utilisant les (P-1) véhicules de la journée précédente et un véhicule de la journée J. Nous faisons glisser toute la fenêtre sur le chromosome de gauche à droite par un véhicule en utilisant (P-2) véhicules de (J-1) et 2 véhicules de J. Cette opération est répétée autant de fois jusqu'à ce que le dernier véhicule de la fenêtre coïncide avec le dernier véhicule du chromosome.

Pour calculer F1 le nombre total de violation des ratios prioritaires, nous additionnons le nombre total de violation pour chaque ratio prioritaire. Le calcul de F2 s'effectue de la même manière que F1 mais pour les ratios non prioritaires.

3.2 Les instances

Chaque instance correspond au jeu de données d'une usine. Chacune est constituée de quatre fichiers de données :

- Fichier de la longueur maximum de rafales de peinture (Batch Limit Paint).
- Fichier des objectifs d'optimisation qui désigne l'ordre d'optimisation des fonctions objectifs.
- Fichier des contraintes de ratio qui donne la liste des contraintes de ratio prioritaires et non prioritaires.
- Fichier des véhicules figés de la journée (J-1) et des véhicules à ordonner de la journée J. Pour chaque véhicule, il y a son identifiant, sa journée de fabrication (date J-1 ou J), sa teinte et une valeur binaire (0/1) par contrainte de ratio indiquant si le véhicule est concerné par la contrainte de ratio, le numéro de séquence calculé par le solveur de l'entreprise. Le nombre de véhicules ayant (J-1) est égale à (DenMax-1) véhicules, DenMax étant le maximum des dénominateurs des contraintes de ratio.

Il s'agit de fichiers textes codés en codage "ascii". Ils utilisent « ; » pour séparer entre les champs de chaque variable.

3.3 Algorithmes génétiques

Dans cette partie, nous comparons deux manières de procéder pour résoudre ce problème d'ordonnancement de véhicules sur une chaîne de montage. La première méthode consiste à utiliser l'algorithme génétique pour résoudre pas-à-pas chaque critère et à la fin de donner la solution optimale trouvée en respectant que la solution obtenue pour un critère d'une étape ne se dégrade pas lorsque nous optimisons le critère de l'étape suivante. La deuxième méthode est un algorithme génétique qui optimise les trois critères en même temps en utilisant une fonction d'évaluation pondérée.

Dans les deux cas, Nous n'utilisons pas d'opérateur de croisement. En effet, son utilisation aboutira à une détérioration incorrigible et irréversible de la séquence ordonnancée.

3.3.1 Méthode pas-à-pas

Pour résoudre ce problème pas-à-pas, nous avons opté pour l'utilisation de trois algorithmes génétiques à chaque étape. L'objectif est de minimiser à chaque fois un critère d'évaluation. En effet, nous minimisons successivement les critères d'évaluation suivants :

- le nombre de violation de ratios prioritaires F1, en utilisant l'algorithme génétique "Gen_RatP1";
- le nombre de purge F2, en utilisant l'algorithme génétique "Gen_Purge2";
- le nombre de violation de ratios non prioritaires F3, en utilisant "Gen_RatNP3".

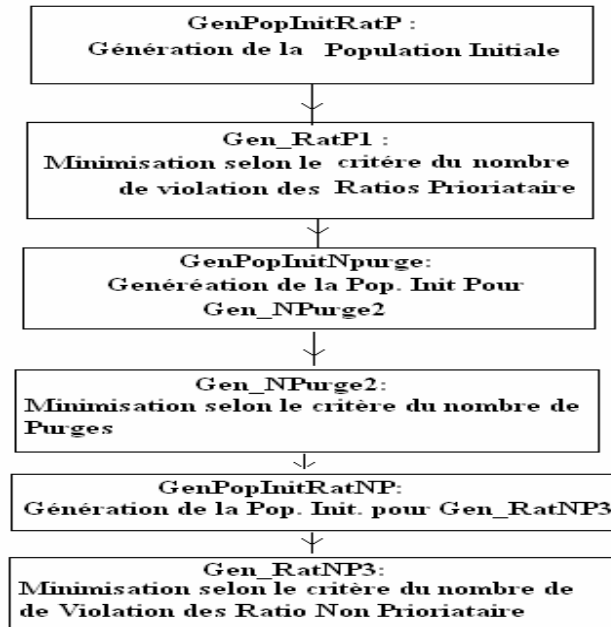


FIG. 3.1: Succession des procédures

Les points communs entre ces algorithmes génétiques sont les paramètres suivants :

- Le codage du chromosome utilisé est un tableau de nombres entiers. Ainsi, nous octroyons à chaque véhicule un numéro qui permet de l'identifier. La longueur du tableau est égale au nombre de véhicules du jour J à ordonnancer.
- Un chromosome faisable si c'est une séquence de véhicules vérifiant la contrainte dure. Elle s'agit de ne pas regrouper des véhicules de même couleur dépassant un nombre maximum défini par chaque instance.
- La population initiale est générée selon un processus qui prend en compte les spécificités du problème.
- Test d'arrêt est le nombre maximum d'itérations fixé à NMax=10.
- La méthode de sélection est le principe de la roulette.
- L'opérateur de mutation est adapté à chaque critère d'optimisation et respecte la contrainte dure. Il est appliqué avec une probabilité de 0.9.
- La méthode d'insertion est l'élitisme.

Le schéma général de chaque algorithme génétique est représenté dans la figure 3.2.

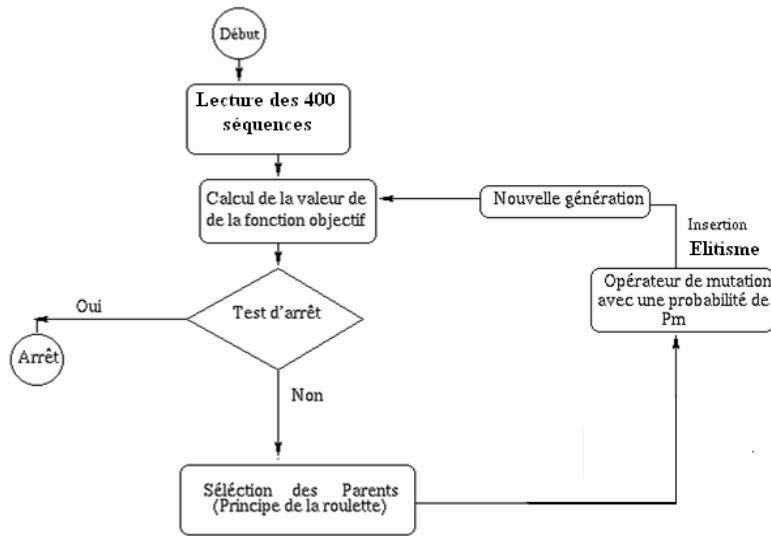


FIG. 3.2: Algorithme génétique pour minimiser le critère F «Gen_Critère».

Vous remarquez que chaque algorithme génétique utilisé, ici, est caractérisé par une méthode de génération de sa population initiale, sa fonction objectif dépendant fondamentalement du critère à optimiser et enfin de son opérateur de mutation.

Population initiale

Comme nous utilisons trois algorithmes génétiques, nous sommes amenés à générer à chacun une population initiale adaptée de taille fixée à 400 éléments :

GenPopInitRatP : Dans cette procédure, nous créons chaque individu en suivant les étapes suivantes :

1. choix aléatoire d'un ratio N/P parmi tous les ratios ;
2. création d'une séquence qui place, quand c'est possible, dans chaque fenêtre de longueur P véhicules, N véhicules ayant ce critère de ratio ;
3. recherche des sous-séquences ayant des véhicules de même couleur dépassant le nombre maximum toléré de véhicules de même teinte (Batch Limit Paint) ;
4. introduction d'un véhicule de couleur différent de chaque sous séquence.

GenPopInitNPurge : nous prenons le meilleur individu obtenu par l'algorithme génétique "Gen_RatP1" et nous le perturbons 399 fois par un opérateur qui laisse le nombre de contrainte prioritaire fixe.

GenPopInitRatNP : nous prenons la séquence obtenue par l'algorithme génétique "Gen_NPurge2" et nous le mutons 399 fois par un opérateur de mutation convenable qui fixe les valeurs des critères déjà optimisés notamment le nombre de violation des contraintes prioritaires et le nombre de purge.

Opérateurs de mutation

Nous avons confectionné, en plus des deux opérateurs de mutation employés dans la création des populations initiales, trois opérateurs de mutation afin d'optimiser chaque critère en laissant fixé la valeur des critères déjà optimisés.

DecRatP : il est employé dans l'algorithme génétique "Gen_RatP1" afin de minimiser le nombre de violation tous les critères de ratio prioritaire. Il procède comme suite :

1. choix aléatoire d'un ratio prioritaire N/P ;
2. recherche une première fenêtre de longueur P dans le chromosome où le critère est violé ;
3. recherche une deuxième fenêtre de longueur P dans le chromosome contenant au plus t véhicules concernés par le ratio prioritaire avec ($t < N$) ;
4. échange dans la première fenêtre de $(N - t)$ véhicules ayant le ratio par d'autres de la deuxième fenêtre qui ne l'ont pas.

DecNPurge : l'algorithme génétique "Gen_NPurge2" se sert de cet opérateur pour minimiser le nombre de purge en sauvegardant le même nombre de violation des ratios prioritaires. Il consiste à essayer de regrouper les véhicules de même couleur comme suite :

1. recherche d'un véhicule X ayant une couleur différente de ces deux voisins les plus proches ;
2. recherche d'un véhicule Y ayant la même couleur que l'un des voisins de X et semblable à X par rapport aux ratios prioritaires ;
3. échanger X par Y .

DecRatNP : l'algorithme génétique "Gen_RatNP3" recourt à cet opérateur afin de minimiser le nombre de violation de ratio non prioritaires tout en laissant les valeurs de nombre de violation de ratio prioritaire et nombre de purge fixes. Il essaye d'alléger les fenêtres qui viole l'un des ratios non prioritaire :

1. choix aléatoire d'un ratio non prioritaire N/P ;

2. recherche une première fenêtre de longueur P dans le chromosome où le critère est violé ;
3. recherche une deuxième fenêtre de longueur P dans le chromosome contenant au plus t véhicules concernés par le ratio non prioritaire avec ($t < N$) ;
4. échange avec précaution dans la première fenêtre de $(N - t)$ véhicules ayant le ratio par d'autres de la deuxième fenêtre qui ne l'ont pas.

Nous faisons attention à ce que les véhicules échangés aient les mêmes ratios prioritaires et la même teinte.

3.3.2 L'algorithme génétique avec une fonction d'évaluation pondérée

Cet algorithme génétique se base sur le fait de minimiser d'un seul coup les trois critères en utilisant la fonction d'évaluation pondérée suivante :

$$F = 10000 \times F1 + 100 \times F2 + F3.$$

Cette fonction offre plus de l'importance à $F1$ qu'à $F2$ et à $F2$ plus qu'à $F3$. Il utilise la méthode *GenPopInitRatP* pour générer la population initiale de taille fixée à 400. Chaque individu sélectionné par la méthode de la roulette est muté $M/3$ fois par les trois opérateurs de mutation *DecRatP*, *DecNPurge* et *DecRatNP* explicités précédemment. M est un paramètre indiquant la taille du chromosome. Le meilleur chromosome issu de ces mutations sera inséré dans la nouvelle population jusqu'à l'obtention de 400 individus. et nous répétons ce processus dix fois. Le meilleur chromosome obtenu dans la dernière itération est la solution finale de cet algorithme génétique.

3.4 Résultats numériques

Nous comparons les résultats obtenus par les deux méthodes : méthode pas-à-pas et la méthode avec la fonction objectif pondérée sur trois instances. Nous avons exécuté chaque algorithme trois fois sur chaque instance.

Au début, Nous avons fixé le nombre maximum d'itérations à 1000. Mais, Nous avons remarqué que l'algorithme pas-à-pas trouve la solution à partir de la dixième itérations. C'est pour cela que nous avons pris dans le test d'arrêt N_{Max} égale à 10. Les résultats obtenus sont enregistrés dans le tableau (TAB.3.4).

Nom de l'instance	F1	F2	F3	Temps (s)
022_3_4_EP_RAF_ENP	12	154	6	8.06
	12	154	6	7.92
(3;9;450;499)(*)	12	154	6	7.90
024_38_3_EP_RAF_ENP	304	617	274	27.90
	304	617	277	28.02
(5;13;10;1274)(*)	304	617	275	27.95
024_38_5_EP_RAF_ENP	334	789	222	27.95
	334	789	223	28.10
(5;13;10;1329)(*)	334	789	220	28.40

TAB. 3.1: Les valeurs minimales de différents critères pour trois instances et leurs temps d'exécution. (*)=(Nbre Ratio Prior.; Nbre Tot. des Ratios; Batch Limit Paint; Nbre Tot. Veh.)

Nous remarquons que les valeurs optimales obtenues par cette méthodes restent fixées pour les deux premières critères nombre total de violation de ratio prioritaires et le nombre de purge pour les trois exécutions sur chaque instance. Alors que les valeurs du troisième critère nombre total de violation de ratio non prioritaire fluctuent un petit peu.

Le temps d'exécution augmente lorsque le nombre de véhicules et le nombre de contraintes augmentent. Ainsi, la première instance est constituée de 490 véhicules à agencer, 9 ratios dont 3 sont prioritaires et la taille de sous-séquence de même couleur fixée à 450. Le temps moyen d'exécution de cet algorithme est d'environ de 7.96 secondes. Alors que la deuxième instance de 1261 véhicules à ordonnancer et 13 ratios avec Batch Paint Limit égale à 10 dure en moyenne 27.95 secondes. La durée moyenne d'exécution de la méthode pas-à-pas sur la troisième instance, constituée de 1316 véhicules à ordonner, est égale à 28.15 secondes.

En s'intéressant maintenant à l'algorithme multi-objectifs, nous trouvons le tableau (TAB.3.4) de résultats suivant :

Nom de l'instance	F1	F2	F3	Temps
022_3_4_EP_RAF_ENP	0	85	0	22 mn 52s
	0	85	1	22 mn 50s
(3;9;450;499)(*)	0	85	1	22 mn 52s
024_38_3_EP_RAF_ENP	156	340	338	3h 14mn 47s
	156	340	323	3h 10mn 48s
(5;13;10;1274)(*)	156	340	354	3h 9mn 32s
024_38_5_EP_RAF_ENP	167	434	241	3h 24 mn 25s
	167	434	222	3h 24 mn 59s
(5;13;10;1329)(*)	167	434	254	3h 25 mn 10s

TAB. 3.2: Les valeurs minimales de différents critères pour trois instances et leurs temps d'exécution en utilisant l'algorithme génétique multi-objectifs. (*)=(Nbre Ratio Prior.; Nbre Tot. des Ratios; Batch Limit Paint; Nbre Tot. Veh.)

En se concentrant sur les deux premiers critères à optimiser (nombre total de violation de ratio prioritaire et nombre de purge), l'algorithme génétique avec la fonction d'évaluation pondérée donne des résultats meilleurs que la méthode pas-à-pas pour les trois instances.

Alors que par rapport au dernier critère (nombre total de violation de ratio non prioritaire), la balance se penche en faveur de la méthode pas-à-pas qui obtient des valeurs inférieures pour les deux dernières instances. Néanmoins l'algorithme génétique avec la fonction objectif pondérée reste avantageux par rapport à ce critère en l'appliquant à la première instance.

Le succès de l'algorithme génétique multi-objectifs a été terni par sa durée d'exécution assez longue. En effet, sa durée moyenne est 22 minutes et 51 secondes pour résoudre la première instance alors qu'il prend entre 3h 10minutes et 3h 26 minutes pour la résolution de l'un des deux autres.

3.5 Conclusion

Dans ce chapitre, nous avons présenté un problème d'ordonnancement de véhicules pour une chaîne de production. C'est un problème réel multi-objectifs qui a été proposé par un constructeur automobile et dont l'objectif était de fournir une séquence de véhicules ("un film") qui optimise le mieux les trois critères suivants :

1. Nombre total de violation des ratios prioritaires;
2. Nombre de purges;
3. Nombre total de violations des ratios non prioritaires.

Une solution optimale de ce problème essaye généralement d'écarter les véhicules ayant le même critère ratio et à regrouper les véhicules ayant la même couleur.

L'ordre de priorité de chaque critère est indiqué dans les instances. Nous avons pris en considération trois instances qui proposent l'ordre de priorité cité ci-dessus.

Pour résoudre ce problème, il y a deux stratégies possibles :

Stratégie1 : optimiser pas-à-pas les trois critères. Autrement dit, minimiser le nombre total de violation du ratio prioritaire, ensuite minimiser le nombre de purge et à la fin le nombre total de ratio non prioritaire. Cette stratégie a été vivement recommandée par le constructeur automobile.

Stratégie2 : optimiser simultanément les trois critères.

Nous avons proposé pour chaque stratégie une variante d'algorithme génétique :

- **algorithme génétique pas-à-pas :** il est constitué de trois algorithmes génétiques :
 - **Gen_RatP1 :** c'est un algorithme génétique qui optimise le nombre total de violation de ratios prioritaire (F1) qui utilise la méthode *GenPopInitRatP* pour générer la population initiale et l'opérateur de mutation *DecRatP*.
 - **Gen_Purge2 :** Il optimise le critère de nombre de purge sans altérer la valeur F1 trouvée Gen_RatP1 en utilisant la méthode *GenPopInitNPurge* de génération de la population initiale et l'opérateur de mutation *DecNPurge*.
 - **Gen_RatNP3 :** Il optimise le critère F3 sans détériorer les valeurs de F2 et F1 en employant comme méthode de génération de la population initiale *GenPopInitRatNP* et comme opérateur de mutation *DecRatNP*.
- **algorithme génétique avec fonction objectif pondérée :** Il est composé de la fonction objectif pondérée $F = 10000 \times F1 + 100 \times F2 + F3$, de la méthode de génération de la population initiale *GenPopInitRatP* et trois opérateurs de mutation *DecRatP*, *DecNPurge* et *DecRatNP* utilisés M/3 fois chaque individu avec M est la longueur du chromosome. Le meilleur individu obtenu par ces mutations est inséré dans la population suivante.

Les résultats obtenus sur les trois instances mettent en exergue la deuxième stratégie et l'algorithme génétique correspondant en se référant aux valeurs optimales des critères malgré une durée d'exécution assez longue.

tel-00126292, version 2 - 8 Mar 2007

Discussion

Dans cette troisième partie, nous nous sommes intéressés aux problèmes d'ordonnements suivant :

chapitre 1 problème de Job Shop de type (J,M) ;

chapitre 2 problème d'atterrissage des avions sur une piste de l'aéroport ;

chapitre 3 problème d'ordonnement des véhicules dans une chaîne de production.

Pour la résolution du problème JSP, nous avons proposé la variante suivante de l'algorithme génétique qui est caractérisée par :

- Le codage du chromosome en nombre entier entre 0 et $(J \cdot M - 1)$;
- La population initiale est composée d'un individu généré heuristiquement et d'autres aléatoirement ;
- La méthode de sélection est la roulette ;
- L'exploration du domaine des solutions s'effectue par le biais de trois opérateurs de croisement *order1*, *edrx*, et notre opérateur de croisement *Cedrx*. La probabilité de croisement est de $P_x = 1$ mais chaque opérateur de croisement à une fréquence d'application : les opérateurs *order1* et *edrx* ont une fréquence de $2/5$ alors que *Cedrx* est employé avec une fréquence de $1/5$;
- Notre opérateur de mutation, adapté à la problématique et au codage utilisé, a permis de sortir des minimums locaux et d'obtenir des meilleurs résultats. Il est appliqué avec une probabilité $P_m = 0.5$;
- La méthode d'insertion est l'élitisme.

Les expérimentations numériques (TAB.1.7) ont montré l'efficacité et la robustesse de notre algorithme. En effet, dans la plupart des cas, il donne les meilleures valeurs de la fonction objectif déjà publiées regroupées par Ombuki *et al.* ([86]) concernant les instances de Lawrance ([62]).

La résolution du problème d'atterrissage de P avions sur une piste de l'aéroport (ALP) a été traité dans le deuxième chapitre. Nous y avons comparé deux algorithmes génétiques :

- .1. quatre algorithmes génétiques dépendant de leurs fonctions objectifs ;
- .2. algorithme de population avec une fonction objectif pondérée.

Parmi les quatre, A.G.3 s'est illustré en donnant de bons résultats par rapport aux autres. Il est caractérisé par un algorithme génétique qui fonctionne sur des permutations en employant un opérateur de croisement et de mutation taillés par notre soin sur mesure et pour calculer les heures d'atterrissage X_i , nous nous servons de trois opérations appliquées successivement :

- **Calcul déterministe initial des heures d'atterrissage des avions (II.1) ;**

- La formule (2.4) ;
- **Amélioration du calcul des X_i (III)** ;

Cette dernière sert à trouver le décalage qui permet de minimiser (2.4). Quand l'algorithme A.G.3 obtient cette solution. Celle-ci sera l'objet d'une application intensive de la méthode de recherche locale en se basant sur le mécanisme aléatoire (II.2) de calcul des X_i .

Pour l'algorithme de population que nous avons proposé se base sur :

- **algorithme génétique** caractérisé par son opérateur de croisement, créer spécialement pour le problème ALP, appliqué avec $P_x = 1$ et son opérateur de mutation $P_m = 0.5$ et la méthode de sélection tournoi binaire qui choisit le meilleur parmi deux individus prélevés aléatoirement ;
- **algorithme de recherche *Scatter*** s'illustre par la constitution d'une sous population de 10 éléments sélectionnés aléatoirement afin de créer un individu par l'opérateur de croisement linéaire qui sera amélioré et inséré dans la nouvelle population ;
- **algorithme *Bionomique*** qui se distingue par la création d'un graphe de la population qui servira par la suite pour la construction de l'ensemble des parents qui sera assujetti à l'opérateur de croisement linéaire afin de fabriquer un enfant qui sera amélioré et insérer dans la nouvelle population.

Ces algorithmes emploient le même type de codage (utilise les proportions y_i au lieu des X_i) avec la même fonction d'évaluation pondérée (2.6) qui pénalise les individus qui ne respectent pas les contraintes de séparation. Notre algorithme de population fonctionne avec une population (300 chromosomes) composée de trois sous-populations de même taille. Initialement, elles sont créées de manière semblable et sur chacune, nous appliquons un algorithme. pour chaque algorithme ; trois individus sont choisis selon l'ordre croissant de l'heure d'arrivée au plus tôt (E_i), l'heure prévue d'arrivée (T_i) et l'heure au plus tard (L_i) avec des y_i qui vérifient que les heures d'atterrissage sont égales aux heures prévues T_i . Les autres individus sont constituées de permutations des avions et des proportions y_i générées aléatoirement. Sur chaque sous-population, nous appliquons un algorithme. Le meilleur individu obtenu par les trois est insérée dans les nouvelles sous- populations à la place du plus mauvais. Nous répétons cette itération 10 000 fois.

Pinol et Beasley [93] ont noté que les algorithmes SS et BA donnent des solutions de bonnes qualités pour les problèmes de grandes tailles mais ils sont *très lents*. Ciesielski et Scerri [28] ont utilisé différents opérateurs génétiques standards mais *les résultats n'ont pas été concluants*.

La collaboration de ces algorithmes a été mise en oeuvre pour contrecarrer ces fléaux. En effet, notre algorithme a donné les meilleurs résultats pour toutes les instances de Beasley ([4]) par rapport aux algorithmes (A.G.3) et (A.G.4).

Dans le troisième chapitre, nous avons présenté un problème réel multi-objectifs. Il s'agit du problème d'ordonnement de véhicules dans une chaîne de production. L'objectif est de minimiser trois critères :

1. Nombre total de violation des ratios prioritaires ;

-
2. Nombre de purge ;
 3. Nombre total de violation des ratios non prioritaires.

Une solution optimale de ce problème essaye généralement d'écartier les véhicules ayant le même critères ratio et à regrouper les véhicules ayant la même couleur.

Lors de la résolution de ce problème, nous avons mis dos à dos deux visions en utilisant toujours l'algorithme génétique :

- **algorithme génétique pas-à-pas** : il est constitué de trois algorithmes génétiques :
 - **Gen_RatP1** : c'est un algorithme génétique qui optimise le nombre total de violation de ratios prioritaire (F1) qui utilise la méthode *GenPopInitRatP* pour générer la population initiale et l'opérateur de mutation *DecRatP*.
 - **Gen_Purge2** : Il optimise le critère de nombre de purge sans altérer la valeur F1 trouvée Gen_RatP1 en utilisant la méthode *GenPopInitNPurge* de génération de la population initiale et l'opérateur de mutation *DecNPurge*.
 - **Gen_RatNP3** : Il optimise le critère F3 sans détériorer les valeurs de F2 et F1 en employant comme méthode de génération de la population initiale *GenPopInitRatNP* et comme opérateur de mutation *DecRatNP*.
- **algorithme génétique avec fonction objectif pondérée** : Il est composé de la fonction objectif pondérée $F = 10000 \times F1 + 100 \times F2 + F3$, de la méthode de génération de la population initiale *GenPopInitRatP* et trois opérateurs de mutation *DecRatP*, *DecNPurge* et *DecRatNP* utilisés M/3 fois chaque individu avec M est la longueur du chromosome. Le meilleur individu obtenu par ces mutations est inséré dans la population suivante.

Les résultats obtenus sur les trois instances mettent en exergue la deuxième stratégie et l'algorithme génétique correspondant en se référant aux valeurs optimales des critères malgré une durée d'exécution assez longue.

L'utilisation d'une fonction objectif pondérée pour une deuxième fois a montré son efficacité.

Conclusion générale

Dans ce travail, nous avons présenté plusieurs contribution à la construction de variantes d'algorithmes génétiques pour la résolution de problèmes d'ordonnancement en l'occurrence, le problème du voyageur de commerce (T.S.P.), le problème de Job Shop (JSP), le problème d'atterrissage des avions sur une piste (ALP) et le problème d'ordonnancement des véhicules dans une chaîne de production.

À travers la première partie, nous avons exposé en détail, dans son premier chapitre, les composantes des algorithmes génétiques :

- Codage du chromosome ;
- Génération de la population initiale ;
- Méthodes de sélection ;
- Opérateurs de croisement ;
- Opérateurs de mutation ;
- Méthode d'insertion.

Chaque composante précitée présente plusieurs choix. L'adoption d'une possibilité pour chaque composante permet de construire une variante de l'algorithme génétique. Vu la multitude des choix , nous pouvons construire plusieurs variantes. La question qui se pose d'elle même porte sur l'identification des paramètres de l'algorithme génétique qui le rendent efficace pour la résolution d'un problème d'ordonnancement.

Dans le deuxième chapitre, nous y avons présenté les modélisations théoriques des algorithmes notamment la théorie du schéma et la modélisation par chaîne de Markov. Cette dernière approche, a permet d'obtenir de prouver la convergence en probabilité d'une variante d'algorithme génétique.

Aussi, la deuxième partie a été consacrée à répondre à cette question et ce, en identifiant les paramètres de l'algorithme génétique efficaces pour la résolution du problème T.S.P. À cet effet, nous avons comparé, dans un premier temps, huit opérateurs de croisement entre eux en utilisant le critère *racine carré de la moyenne des erreurs carrées (RMSE)*. Ce faisant, nous avons trouvé qu'il vaut mieux utiliser l'opérateur **order1**. Car il est plus stable que *edrx* et obtient les plus petites valeurs de la fonction objectif. De même qu'au niveau de l'exploration de l'ensemble des solutions, l'opérateur **order1** est le meilleur.

Dans un deuxième temps, nous avons confronté cinq opérateurs de mutation. Nous avons obtenu que le meilleur opérateur de mutation est **im**, suivi par *cim*, qui dépasse

de peu *twors*. Ces opérateurs sont talonnés par *thrors* et en dernière position, se situe l'opérateur de mutation *throas*.

Puis, dans un troisième temps, nous avons pris en considération les opérateurs de croisement et de mutation, nous avons obtenu que l'opérateur de croisement *edrx* appliqué avec $P_x = 1$ et l'opérateur de mutation *im* avec $P_m = 0.9$ en utilisant une taille de population $N = 8 \times V$ ont donné des meilleurs résultats. Les valeurs de probabilités de croisement et de mutation que nous avons retenu sont loin des valeurs existantes dans la littérature ($P_x \in [0.6, 0.8]$ et $P_m \in [10^{-3}, 10^{-2}]$). Toutefois, ces dernières valeurs administrées à l'algorithme génétique n'ont pas fournis des solutions optimales pour T.S.P. et étaient médiocres.

À cet effet, l'opérateur de croisement a été fixé à *edrx* appliqué avec $P_x = 1$ et l'opérateur de mutation à *im* avec $P_m = 0.9$. Puis, nous nous sommes intéressés à l'influence des méthodes de sélection sur la convergence de cette variante de l'algorithme génétique. Pour cela, nous avons mis en compétition six méthodes de sélection :

- Méthode du tournoi ;
- Méthode du troncation ;
- Méthode de la roulette ;
- Méthode d'ordre exponentiel ;
- Méthode de Boltzmann ;
- Méthode d'ordre linéaire.

De cet examen, il en ressort que les méthodes de sélection du tournoi (avec $t = 0.8$ ou 0.9) et "d'ordre linéaire" (avec $\eta^- = 0.6$) sont mieux adaptées que les autres. Le paramètre de contrôle de la méthode de sélection du tournoi indique qu'il vaut mieux prendre un échantillon de taille proportionnelle à 0.9 de la taille de la population et d'en prendre le meilleur. Celui-ci n'est pas forcément le meilleur individu de toute la population. Alors que le paramètre de contrôle de la méthode "ordre linéaire" suggère qu'il est préférable d'octroyer une grande probabilité de sélection au mauvais individu. Ce que nous avons retenu de ces dernières simulations numériques que le succès de l'algorithme génétique dépend de la méthode de sélection qui oblige à choisir pour les croisements les mauvais et les meilleurs individus de la population et que la méthode de sélection par la roulette a donné des résultats satisfaisants par rapport aux méthodes susvisées.

Par la suite, nous nous sommes intéressés à la méthode de génération de la population initiale de l'algorithme génétique. À ce niveau, nous avons comparé l'influence de six méthodes de génération de la population initiale sur la convergence de l'algorithme génétique. Cette comparaison s'est avérée qu'il est plus judicieux d'utiliser une méthode heuristique (mécanisme du plus proche voisin) car l'algorithme génétique devient plus concurrente.

À la fin de cette deuxième partie, nous avons présenté notre variante de l'algorithme génétique pour la résolution du T.S.P.. Elle est caractérisée par ce qui suit :

1. Au niveau du codage, nous avons utilisé la représentation chemin en attribuant à la première position du chromosome un gène qui reste fixe tout au long de l'exécution

de l'algorithme génétique ;

2. Nous avons utilisé le mécanisme du plus proche voisin dans la génération de la population initiale ;
3. La population a été divisée en trois parties : la première contient les 1% meilleurs individus, les 55% moins meilleurs que les précédents composent la deuxième partie alors que les plus faibles constituent la troisième partie. Pour croiser deux individus choisis au hasard l'un de la première partie et l'autre de la deuxième partie ;
4. L'algorithme génétique se sert de deux opérateurs de croisement *edrx* et du notre *Cedrx* ;
5. Un individu est sélectionné au hasard pour être muté avec *im* ;
6. Lorsque la valeur de la fonction objectif du meilleur individu ne change pas pendant dix itérations. Nous procédons autrement pour la génération de la population suivante en renforçant la mutation pour chaque individu de l'ancienne population afin de sortir de ce minimum local.

L'opérateur de croisement *Cedrx* permet de faire une sorte de retour en arrière ("Backtracking") des arêtes qui ont été sauvegardés par *EDRX* et l'ajout de l'étape δ poussent notre variante à sortir des minimums locaux. Ce qui explique les bons résultats (TAB.5.7).

Dans la troisième partie, nous nous sommes intéressés aux problèmes d'ordonnancement, dont les solutions réalisables peuvent être codées par des permutations :

1. problème de Job Shop de type (J,M) ;
2. problème d'atterrissage des avions sur une piste d'un aéroport ;
3. problème d'ordonnancement des véhicules dans une chaîne de production.

La variante que nous avons proposé pour la résolution du problème JSP est caractérisée par les propriétés suivantes :

- La population initiale est composée d'un individu généré heuristiquement et d'autres aléatoirement ;
- L'exploration du domaine des solutions s'effectue par le biais de trois opérateurs de croisement *order1*, *edrx*, et notre opérateur de croisement *Cedrx*. La probabilité de croisement est de $P_x = 1$ mais chaque opérateur de croisement a une fréquence d'application : les opérateurs *order1* et *edrx* ont une fréquence de 2/5 alors que *Cedrx* est employé avec une fréquence de 1/5 ;
- Notre opérateur de mutation, adapté à la problématique et au codage utilisé, a permis de sortir des minimums locaux et d'obtenir des meilleurs résultats. Il est appliqué avec une probabilité de mutation $P_m = 0.5$.

Les expérimentations numériques (TAB.1.7) ont montré l'efficacité et la robustesse de notre algorithme. En effet, il donne les meilleures valeurs de la fonction objectif sur les instances de Lawrance ([62]).

Pour le problème d'atterrissage de P avions sur une piste de l'aéroport (ALP), nous avons présenté plusieurs algorithmes génétiques. Mais, celui qui s'est mis en exergue est l'algorithme de population. Sa population est composée de trois sous-populations de même

taille. Initialement, elles sont créées de manière semblable et sur chacune, trois individus sont choisis selon l'ordre croissant de l'heure d'arrivée au plus tôt (E_i), l'heure prévue d'arrivée (T_i) et l'heure au plus tard (L_i) avec des y_i qui vérifient que les heures d'atterrissage sont égales aux heures prévues T_i . Les autres individus sont constituées de permutations des avions et des proportions y_i générées aléatoirement. Sur chaque sous-population, nous appliquons l'un des algorithmes suivants :

- **algorithme génétique** caractérisé par son opérateur de croisement, créer spécialement pour le problème ALP, appliqué avec $P_x = 1$ et son opérateur de mutation $P_m = 0.5$ et la méthode de sélection tournoi binaire qui choisit le meilleur parmi deux individus prélevés aléatoirement ;
- **algorithme de recherche *Scatter*** s'illustre par la constitution d'une sous population de 10 éléments sélectionnés aléatoirement afin de créer un individu par l'opérateur de croisement linéaire qui sera amélioré et inséré dans la nouvelle population ;
- **algorithme *Bionomique*** qui se distingue par la création d'un graphe de la population qui servira par la suite pour la construction de l'ensemble des parents qui sera assujetti à l'opérateur de croisement linéaire afin de fabriquer un enfant qui sera amélioré et inséré dans la nouvelle population.

Le meilleur individu obtenu par les trois algorithmes précités est inséré dans les nouvelles sous-populations à la place du plus mauvais. Cette itération est répétée 10 000 fois. Ces algorithmes emploient le même type de codage (utilise les proportions y_i au lieu des X_i) avec la même fonction d'évaluation pondérée (2.6) qui pénalise les individus ne respectant pas les contraintes de séparation.

La coopération et la collaboration de ces algorithmes ont été mises en oeuvre pour contre-carrer les remarques de Pinol et Beasley [93] qui ont signalé que les algorithmes SS et BA donnent des solutions de bonne qualité pour les problèmes de grandes tailles mais ils sont très lents, et celles de Ciesielski et Scerri [28] qui ont utilisé différents opérateurs génétiques standards mais les résultats n'ont pas été concluants.

Notre algorithme a donné les meilleurs résultats pour toutes les instances de Beasley ([4]).

À la fin de la troisième partie, le problème d'ordonnement de véhicules dans une chaîne de production a été traité. Il s'agit d'un problème réel multi-critères dont l'objectif est de minimiser les trois critères, ci-après, dans cet ordre :

1. Nombre total de violation des ratios prioritaires ;
2. Nombre de purge ;
3. Nombre total de violation des ratios non prioritaires.

Nous avons présenté deux algorithmes génétiques qui suivent deux stratégies différentes ; optimisation critère par critère ou optimisation simultanée de tous les critères. L'algorithme génétique qui suit la deuxième stratégie l'a emporté sur le premier en utilisant une fonction d'évaluation pondérée et des opérateurs de mutation adéquats. Les résultats obtenus sur les trois instances mettent en exergue la deuxième stratégie

et l'algorithme génétique correspondant en se référant aux valeurs optimales des critères malgré une durée d'exécution assez longue.

Ce dernier problème a montré l'efficacité de l'algorithme génétique pour la résolution des problèmes multi-critères.

Bibliographie

- [1] J. Abela, D. Abramson, M. Krishnamoorthy, A. De Silva, and G. Mills, *Computing optimal schedules for landing aircraft*. Proceedings of the 12th National ASOR Conference, 71-90, 1993.
- [2] J. Adams, E. Balas and D. Zawack. *The shifting bottleneck procedure for job shop scheduling*. Journal Manage. Sci., vol. 34, n 3, 1988.
- [3] A. Agapie. *Genetic algorithms : minimal conditions for convergence*. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, Artificial Evolution'97, LNCS. Springer Verlag. 1997.
- [4] <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/airlandinfo.html>
- [5] D. Applegate, R. Bixby, V. Chvátal, W. Cook, *Finding cuts in the TSP(A preliminary report)*,1995.
- [6] D. Applegate, R. Bixby, V. Chvátal, W. Cook, Computational Combinatorial Optimization, M. Junger and D. Naddef, editors (Springer, 2001)
- [7] R. Axelrod. *The evolution of strategies in the iterated prisoner's dilemma*. in L. D. Davis, ed., Genetic algorithms and simulated annealing, Morgan Kaufmann, 1987.
- [8] S. Bagchi, S. Uckun, Y. Miyabe and K. Kawamura. *Exploring problem-specific recombination operators for job shop scheduling*. In Proc. of the Fourth Int. Conf. on Genetic Algorithms (Edited by Belew and Booker), pp. 10-17. Morgan Kaufman, San Mateo, Calif. 1991.
- [9] K. Baker, *Introduction to sequencing and scheduling*. John Wiley, 1974.
- [10] J. Bean. *Genetic algorithms and random keys for sequencing and optimization*. ORSA J. Computing 6, 154-160, 1994.
- [11] J. E. Beasley, M. Krishnamoorthy, Y. M. Sharaiha, and D. Abramson, *Scheduling Aircraft Landings-the static case*. Transportation Science, 34(2), 180-197, 2000.
- [12] J. E. Beasley, J. Sonander, and P. Havelock, *Scheduling aircraft landings at London Heathrow using a population heuristic*. Journal of the Operational Research Society, 52, 483-493, 2001.
- [13] H. Pinol, and J. E. Beasley, *Scatter Search and Bionomic Algorithms for the Aircraft Landing Problem*. to appear in the European Journal of Operational Research, Currently available from <http://people.brunel.ac.uk/~mastjjb/jeb/jeb.html>, 2004.
- [14] R. Beckers, J. L. Deneubourg, and S. Gross, *Trails and U-turns in the selection of the shortest path by the ant Lasius Niger*. Journal of Theoretical Biology, vol. 40, pp. 201-211,1992.

- [15] M. Bellmore, J. C. Malone, *Pathology of traveling-salesman subtour-elimination algorithms*. Oper. Res., 19, 278-307, 1972.
- [16] L. Bianco, A. Mingozzi and A. Ricciardelli, *The Traveling Salesman Problem with Cumulative Costs*. Networks, 23, 81-91, 1993.
- [17] L. Bianco and M. Bielli, *System Aspects and Optimization Models in ATC Planning*. Large Scale Computation and Information Processing in Air Traffic Control, 47-99, 1993.
- [18] L. Bianco, P. Dellolmo and S. Giordani, *Minimizing Total Completion Time Subject to Release Dates and Sequence Dependent Processing Times*. Annals of Operations Research, 86, 393-415, 1999.
- [19] C. Bierwirth, D.C. Mattfeld, and H. Kopfer, *On permutation representation for scheduling problems*. Parallel problem solving from nature IV, Springer-Verlag, pp. 310-318, 1996.
- [20] N. L. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory 1736-1936*. Clarendon Press, Oxford, 1976.
- [21] T. Blickle and L. Thiele, *A Comparison of Selection Schemes Used in Genetic Algorithms*, Technical Report 11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, 1995.
- [22] S. Bourazza et A. Yassine. *Evolution de l'algorithme génétique vers une meilleure application au problème du voyageur de commerce*. Colloque "Optimisation et Statistique"; Institut Henri Poincaré, 5 Février 2004.
- [23] S. Bourazza, A. Yassine et S. Gueye. *Un algorithme génétique pour un problème d'ordonnement des véhicules dans une usine*. 12èmes Journées MODE-SMAI, Université du Havre, Mars 2004.
- [24] J. P. Byung, R.C. Hyung and S.K. Hyun. *A hybrid genetic algorithm for the job shop scheduling problems*. Comput. Ind. Eng. journal, Vol. 45, N :4, 2003.
- [25] P.M. Camerini, L. Fratta, and F. Maffioli, *On improving relaxation methods by modified gradient techniques*. Mathematical Programming Study3, 26-34, 1974.
- [26] O. Catoni. *Large deviations for Annealing*. PhD thesis, Université de Paris XI, 1990.
- [27] R. Cerf, *Une théorie asymptotique des algorithmes génétiques*, Thèse de Doctorat en 1994, Université de Montpellier II.
- [28] V. Ciesielski, and P. Scerri, *Real time genetic scheduling of aircraft landing times*. Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC98), 360-364, 1998.
- [29] <http://www.prism.uvsq.fr/~vdc/ROADEF/CHALLENGES/2005/>
- [30] V. H. L. Cheng, L. S. Crawford, and P. K. Menon, *Air traffic control using genetic search techniques*. International Conference on Control Application, 1999.
- [31] N. Christofides, S. Eilon, *Algorithms for large-scale Travelling Salesman Problems*, Oper. Res. Quart. J., 44, 1972 : 511-518.

-
- [32] N. Christofides, *The bionomic algorithm*, Paper presented at the AIRO'94 Conference, Savona, Italy, 1994.
 - [33] Christian M. Reidys and Peter F. Stadler, *Combinatorial Landscapes*. SIAM Review, vol. 44, n° 1, 2002.
 - [34] F. Croce, R. Tadei and G. Volta. *A genetic algorithm for the job shop problem*. Complex Systems 22, 15-24, 1995.
 - [35] G.B. Dantzig, R. Fulkerson, S. Johnson, *Solution of a large-scale traveling salesman problem*, Operations Research 2 (1954), 393-410.
 - [36] C. Darwin. *The origin of species by means of naturel selection*. 1859.
 - [37] L. Davis. shop scheduling with genetic algorithm. In Proc. of the First Int. Conf. on Genetic Algorithms (Edited by J. Grefenstette), pp. 136-140. Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.
 - [38] L. Davis. *Applying Adaptive Algorithms to Epistatic Domains*. In Proc. International Joint Conference on Artificial Intelligence, 1985.
 - [39] L. Davis, D. Orvosh, A. Cox and Y. Qiu. *A Genetic Algorithm for Survivable Network Design*. ICGA, 408-415, 1993.
 - [40] K. Deb, A. Sinha, S. Kukkonen. *Multi-objective test problems, linkages, and evolutionary methodologies*. Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8-12, 2006.
 - [41] U. Dorndorf and E. Pesch. *Evolution based learning in a job shop scheduling environment*. Complex Systems 22, 25-40, 1995.
 - [42] M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, IT, 1992.
 - [43] A. T. Ernst, M. Krishnamoorthy, and T. H. Storer, *Heuristic and Exact Algorithms for Scheduling Aircraft Landings*. Networks, 34, 229- 241, 1999.
 - [44] E. Falkenauer and S. Bouffoix. *A genetic algorithm for job shop*. In Proc. 1991 IEEE Int. Conf. on Robotics and Automation, pp. 824-829, 1991.
 - [45] H. Fang, P. Ross and D. Corne. *A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems*. In Proc. of the Fifth Int. Conf. on Genetic Algorithms, pp. 375-382. Morgan Kaufmann Publishers, San Mateo, Calif. 1993.
 - [46] C. Fonlupt, D. Robilliard, Ph. Preux, and EG. Talbi. *Fitness landscape and performance of meta-heuristics*. In Meta-Heuristics — Advances and Trends in Local Search Paradigms for Optimization, chapter 18, pages 255–266. Kluwer Academic Press, 1999.
 - [47] M.I. Freidlin and A.D. Wentzell. *Random Perturbations of Dynamical Systems*. Springer-verlag, New-York, 1983.
 - [48] M. Gen, Y. Tsujimura and E. Kubota. *Solving job-shop scheduling problem using genetic algorithms*. In Proc. of the 16th Int. Conf. on Computer and Industrial Engineering, pp. 576-579. Ashikaga, Japan, 1994.

- [49] J.J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht. *Genetic algorithm for the tsp*. In Proceedings of the First International Conference on Genetic Algorithms, pages 160-168. Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.
- [50] Glover F. *Parametric Combinations of Local Job Shop Rules*. ONR Research Memorandum no. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA. 1963.
- [51] D. Goldberg and R. Lingle. *Alleles, loci, and the Traveling Salesman Problem*. In Proc. International Conference on Genetic Algorithms and their Applications, 1985.
- [52] D. Goldberg. *Genetic Algorithms*. Addison Wesley, 1989. ISBN : 0-201-15767-5.
- [53] D. Goldberg *Genetic Algorithm in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.
- [54] D.E. Goldberg, *Algorithmes génétiques, exploration, optimisation et apprentissage automatique*. Paris : Addison-Wesley, 1994.
- [55] J. F. Gonçalves, J. J. M. Mendes and M. G. C. Resende. *A hybrid genetic algorithm for the job shop scheduling problem*. ATT Labs Research Technical Report TD-5EAL6J, ATT Labs Research, NJ 07932 USA, September 2002.
- [56] S. Goss, S. Aron, J.L. Deneubourg, and J.M. Pasteels, *Self-organised shortcuts in the argentine ant*. Naturwissenschaften, vol. 76, pp. 579-581, 1989.
- [57] C. Guéret, C. Prins, M. Sevaux, *Programmation linéaire 65 problèmes d'optimisation modélisés et résolus avec Visual Xpress*, Ed. Eyrolles, 2000.
- [58] K. Helsgaun, *An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic*. DATALOGISKE SKRIFTER (Writings on Computer Science), No. 81, 1998.
- [59] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan, 1975.
- [60] B. Hölldobler and E.O. Wilson. *The ants*. Springer-Verlag, Berlin, 1990.
- [61] C. Holsapple, V. Jacob, R. Pakath and J. Zaveri. *A genetic-based hybrid scheduler for generating static schedules in flexible manufacturing contexts*. IEEE Trans. Systems, Man, and Cybernetics 23, 953-971 (1993).
- [62] OR Library site : <http://www.brunel.ac.uk/depts/ma/research/jeb/info.html>.
- [63] T. Jones and S. Forrest, *Fitness distance correlation as a measure of problem difficulty for genetic algorithms*. in Proc. 6th Int. Conf. Genetic Algorithms, L. Eshelman, ed. San Mateo, CA, Morgan Kaufmann, pp.184-192, 1995.
- [64] K. D. Jong. *Adaptive systeme design : A genetic approach*. IEEE Transactions on Systems, Man, and Cybernetics 10(3), 556-574, 1980.
- [65] G. Jung, and M. Laguna, Time segmenting heuristic for an aircraft landing problem, Working paper, Leeds School of Business, University of Colorado, Boulder, CO 80309-0419, USA. Submitted for publication. Currently available from <http://leedsfaculty.colorado.edu/laguna/articles/tsh.html>, March, 2003.
- [66] S. Kirkpatrick, C. D. Gelatt, Jr., M.P. Vecchi, *Optimization by Simulated Annealing*, Science, Number 4598, 13 May 1983.

-
- [67] S. Kobayashi, I. Ono and M. Yamamura. *An efficient genetic algorithm for job shop scheduling problems*. In Proc. of the Sixth Int. Conf. on Genetic Algorithms, pp. 506-511. Morgan Kaufmann Publishers, San Francisco, Calif. 1995.
- [68] K. Krishnakumar, D. Goldberg. *Control system optimization using genetic algorithm*. Journal of Guidance, Control, and Dynamics 15(3), 735-740, 1992.
- [69] M. Krishnamoorthy, A.T.Ernest, Scheduling aircraft landings optimally, Proceedings of the 41st Annual Symposium of AGIFORS, Sydney, Australia, 27 August- 1 September 2001.
- [70] S. Lin, B.W. Kernighan, An Effective Heuristic for the Traveling Salesman Problem. Operations Research 21, p. 498-516, 1973.
- [71] S.W. Mahfoud and D.E. Goldberg. *A Genetic Algorithm for Parallel Simulated Annealing*. eds.), Parallel Problem Solving from Nature 2, Elsevier,1992, 301-310.
- [72] S.W. Mahfoud and D.E. Goldberg. *Parallel recombinative simulated annealing : a genetic algorithm*. Parallel Computing 21, 1995, 1-28.
- [73] N. Marco, C. Godart, J. A. Désidéri, B. Mantel, J. Périaux. *A genetic algorithm compared with a gradient-based method for the solution of an active-control model problem*. Technical report, INRIA. Rapport de Recherche de l'INRIA - Projet SINUS, n0. 2948, 1996.
- [74] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, Equation of state calculations by fast computing machines. J. Chem. Phys. 21, 1087, 1953.
- [75] Z. Michalewicz, C. Janikow, J. Krawczyk. *A modified genetic algorithm for optimal control problems*. Computers and Mathematics with Applications 23(12), 83-94, 1992.
- [76] Z. Michalewicz., *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin : Springer-Verlag, seconde édition, 1994.
- [77] Z. Michalewicz. *Genetic Algorithms Data Structures = Evolution Programs*.Springer-Verlag, 1996.
- [78] Z. Michalewicz and D.B. Fogel. *How to solve it : Modern heuristics*.Springer-Verlag, 2000.
- [79] Z. Michalewicz and D.B. Fogel. *How to solve it : Modern heuristics*. Springer-Verlag, 2000.
- [80] D. L. Miller, J. F. Pekny, *Exact solution of large asymmetric traveling salesman problems*, Science, 251, 754-761, 1991.
- [81] H. Muhlenbein, *How genetic algorithms really work : Mutation and hillclimbing*. Parallel Problem Solving from Nature II, 15, 1992.
- [82] R. Nakano and T. Yamada. *Conventional genetic algorithms for job-shop problems*. In Proc. of the Fourth Int. Conf. on Genetic Algorithms (Edited by Belew and Booker), pp. 477-479. Morgan Kaufman, San Mateo, Calif. 1991.
- [83] B. Norman and J. Bean. *Random keys genetic algorithm for job-shop scheduling : unabridged version*. Technical report, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, 1995.

- [84] I. Oliver, D. Smith, and J. Holland. *A Study of Permutation Crossover Operators on the Traveling Salesman Problem*. In Proc. Second International Conference on Genetic Algorithms and their Applications, 1987.
- [85] B.Ombuki, M. Nakamura, and K.Onaga, *An Evolutionary Scheduling Scheme Based on $gkGA$ Approach for the Job Shop Scheduling Problem*. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science, Vol. E81-A, N0.6, 1998.
- [86] B. Ombuki et M.Ventresca. *Local search algorithms for job shop scheduling problem*. Technical report, november 2002.
- [87] S. Özyildirim, *Computing open-loop noncooperative solution in discrete dynamic games*. Evolutionary Economics 7(1), 23-40, 1997.
- [88] S. Özyildirim, N.Alemdar, *Learning the optimum as nash equilibrium*. Working Paper, 1998.
- [89] M. Padberg and G. Rinaldi, *Optimization of a 532-city symmetric traveling salesman problem by branch and cut*. Operations Research Letters6, 1-7, 1987.
- [90] M. Padberg and G. Rinaldi, *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems*. SIAM Review 33, 60-100, 1991.
- [91] J. Paredis. *Exploiting constraints as background knowledge for genetic algorithms : a case-study for scheduling*. In Proc. of the Second Int. Conf. on Parallel Problem Solving from Nature, pp. 281-290. Elsevier Science Publishers, North-Holland, 1992.
- [92] R. Pereira. *Genetic algorithm optimisation for finance and investment*. Technical report, La Trobe University, 2000.
- [93] H. Pinol, J.E. Beasley, Scatter search and bionomic algorithms for the aircraft landing problem, to appear in the European Journal of Operational Research.
- [94] G. Rudolph. *Convergence of non-elitist strategies*. IN Z. Michalewicz, J.D. Schaffer, H.P. Schwefel, D.B. Fogel, and H. Kitano, editors, Proc. of the first IEEE Int. Conf. on Evolutionary Computation, P : 63-66. IEEE Press, 1994.
- [95] A. Schrijver, *On the history of combinatorial optimization*. 1960.
- [96] M. J. So omer, G. J. Franx, *Scheduling Aircraft Landing using Airlines*. Preferences, http://www.math.vu.nl/~mjso_omer/aircraftlandings.pdf, Aug, 2005.
- [97] P.F. Stadler. Canonical approximation of landscapes. Technical Report 94-09-051, Santa Fe Institute, Santa Fe, NM, 1994.
- [98] P.F. Stadler. Landscapes and their correlations functions. Technical Report 95-07-067, Santa Fe Institute, Santa Fe, NM, 1995.
- [99] G. Syswerda. *Schedule Optimization Using Genetic Algorithms*. In Handbook of Genetic Algorithms. I. Davis, ed. Van Nostrand Reinhold, New York, 1990.
- [100] F. Glover. *Future Paths for Integer Programming and Links to Artificial Intelligence*. volume 5. Computers and Operations Research, 1986.
- [101] H. Tamaki and Y. Nishikawa. *A paralleled genetic algorithm based on a neighborhood model and its application to the jobshop scheduling*. In Proc. of the Second Int. Conf. on Parallel Problem Solving from Nature, pp. 573-582. Elsevier Science Publishers, North-Holland, 1992.

-
- [102] A. Trouvé. *Parallélisation massive du recuit simulé*. PhD thesis, Université de Paris XI, 1993.
- [103] www.iwr.uniheidelberg.de/iwr/comopt/software/TSPLIB95/.
- [104] E.D. Weinberger. *Correlated and uncorrelated fitness landscapes and how to tell the difference*. Biological Cybernetics, 63,325-336, 1990.
- [105] D. Whitley, T. Starkweather, and D. Fuquay. *Scheduling Problems and Traveling Salesman : The Genetic Edge Recombination Operator*. In Proc. Third Int'l. Conference on Genetic Algorithms and their Applications. J. D. Shaeffer,ed. Morgan Kaufmann,1989.
- [106] S. Wright. *The roles of mutations, inbreeding, crossbreeding and selection in evolution*. In D.F. Jones, editor, International Proceedings of the Sixth International Congress on Genetics, volume 1, pages 356-366, 1932.
- [107] T. Yamada and R. Nakano. *A genetic algorithm applicable to large scale job-shop problems*. In Proc. of the second Int conf. on Parallel Problem solving from Nature, pp. 573-582. Elsevier Science publisher, North Holland, 1992.
- [108] T. Yamada and R. Nakano. *Job-Shop Scheduling. Genetic Algorithms in Engineering Systems*. IEE control Engineering series 55, pp. 134-160, 1997.
- [109] A. A. Zhigljavsky. *Theory of Global Random Search*. Kluwer Academic Publishers, 1991.

Résumé

Mots-clés: Ordonnancement, Algorithme Génétique, T.S.P., Job Shop, Atterrissage des avions, Ordonnancement des véhicules dans une chaîne de production.

L'algorithme génétique (AG), est fondé sur les mécanismes de codage, sélection, croisement, mutation et insertion. Chacun présente plusieurs choix ce qui donne résultat à plusieurs variantes de AG.

Nous avons obtenu une variante meilleure d'algorithme génétique pour le problème du voyageur de commerce. Dans laquelle, nous avons introduit notre nouvel opérateur de croisement Cedrx qui est jumelé avec l'opérateur edrx donne de bons résultats.

Cette étude, nous a permis de créer des variantes efficaces d'algorithme génétique pour les problèmes suivants :

- Le problème d'ordonnancement dans les ateliers de type Job Shop ;
- Le problème des atterrissages d'avions (PAA) ;
- Le problème d'ordonnancement des véhicules sur une chaîne de production dans une usine (POV).

Abstract

The genetic algorithm (GA), is based on the mechanisms of coding, selection, crossover, mutation and insertion. Each mechanism presents several choices to giving result to several GA's variants.

We obtained a better variant of genetic algorithm for resolving Travelling Salesman Problem. In this GA's variant, we introduced our new crossover operator Cedrx which when coupled with crossover operator edrx gave good results.

This led to the creation of effective variants of genetic algorithm for resolving the following problems :

- Job Shop Scheduling Problem (JSP) ;
- Aircraft Landing Problem (ALP) ;
- Scheduling Problem of Vehicles on a production line in a factory (SPV).

Keywords: Scheduling, Genetic Algorithm, Travelling Salesman Problem, Job Shop, Aircraft landing Problem, Population algorithm.