Faculté des Sciences et Techniques du Havre

École doctorale SPMII Normandie

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

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

Laboratoire de Mathématiques Appliquées du Havre (LMAH) — EA 3821

Table des matières

Chapitre 1

Algorithme Génétique

Sommaire

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 utilisé 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.

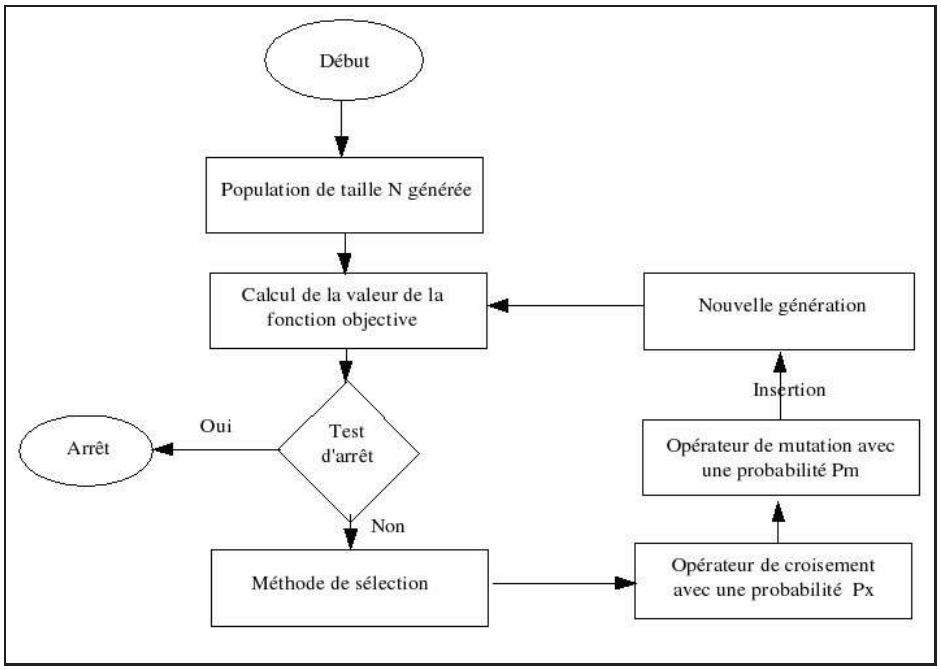

FIG. 1.1: L'organigramme de l'algorithme génétique.

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 :

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

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=$\frac{Rang}{Total(Rang)}$
	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 $Prob_i$, proportionelle à sa valeur F_i de la fonction objectif :

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

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=$\frac{Fitness}{Total(Fitness)}$
	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 à 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 appellée 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).

				⋆							
parent 1	1	0	1	0	1	0	1	0	⟹	enfant1	1	0	1	0	0	1	0	0
parent 2	0	0	1	0	0	1	0	0		enfant2	0	0	1	0	1	0	1	0

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.

			⋆			⋆		
parent 1	1	0	1	0	1	0	1	0
parent 2	0	0	1	0	0	1	0	0

$\Longrightarrow$

enfant1	1	0	1	0	0	1	1	0
enfant2	0	0	1	0	1	0	0	0

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'effectuait sur les gènes en échangeant sa valeur de 0 à 1 ou de 1 à 0 et non sur le chromosome tout entier.

$parent$	0	1	1	0	1	0	0

$\Longrightarrow$

$enfant$	0	1	0	0	1	0	1

TAB. 1.6: Opérateur de mutation d'un bit

Mais avec un algorithme génétique codé autrement, on applique cette probabilté 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 séléctionnés 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'ensembles 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.

Chapitre 1

Le problème de Job Shop (JSP)

Sommaire

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'ordonnancement, 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'ordonnancement

Un générateur d'ordonnancement 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'ordonnancement :

Ordonnancement 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.

Ordonnancement actif c'est un ordonnancement semi-actif dont il est impossible d'avance 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'ordonnancement 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 le 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'ordonnancement 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 ∀ machine k ;
T : ensemble de toutes les opérations de toutes les tâches ;
Tant que (T ≠ ∅) 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 = ∅) Alors
        t(k*) ← 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'occurence 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 $O_{i,j}$ $(i = 1, ..., J \, et \, j = 1, ..., M)$ 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 $\{1, 2, .., M\}$.

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_{ij}. 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 précédence 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 *et al.* [44], Croce *et al.* [34], Kobayashi *et al.* [67]
Job pair relation-based representation	Nakano *et al.* [82], Paredis [91]
Problem specific representation	Bagchi *et al.* [8]
Completion time-based representation	Yamada *et al.* [107]
Disjunctive graph-based representation	Tamaki *et al.* [101]
Job-based representation	Holsapple *et al.* [61], Byung *et al.* [24]
Operation-based representation	Fan *et al.* [45], Gen *et al.* [48]
Random key representation	Bean [10], Norman *et al.* [83], Gonçalves *et al.* [55]
Priority rule-based and machine-based	Dorndorf *et al.* [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	$O_{ij} \Longrightarrow$ Machine (durée opératoire)		
tâche 0	$O_{00} \Longrightarrow$ 2 (3)	$O_{01} \Longrightarrow$ 1 (6)	$O_{02} \Longrightarrow$ 0 (2)
tâche 1	$O_{10} \Longrightarrow$ 0 (2)	$O_{11} \Longrightarrow$ 1 (5)	$O_{12} \Longrightarrow$ 2 (2)
tâche 2	$O_{20} \Longrightarrow$ 1(1)	$O_{21} \Longrightarrow$ 0 (2)	$O_{22} \Longrightarrow$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*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 l * 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 : $T_{ij} = Max(F_{i(j-1)}, T_{hl} + P_{hl})$

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 :

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

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 derniére 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.

Bibliographie

[1] J. Abela, D. Abramson, M. Krishnamo orthy, 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, ERonald, M. Schoenauer, and D. Snyers, editors, Artificial Evolution'97, LNCS. Springer Verlag. 1997.

[4] http ://people.brunel.ac.uk/m̃astjjb/jeb/orlib/airlandinfo.html

[5] D. Applegate, R. Bixby, V. Chvàtal, 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. Krishnamo orthy, Y. M. Sharaiha, and D. Abranmson, *Scheduling Aircraft Landings-the static case.* Transportation Science, 34(2), 180-197, 2000.

[12] JJ. 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/m̃astjjb/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 Sub ject 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 nture 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'ordonnancement 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 Rev. journal, 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 schedulingwith genetic algorithm. In Proc. of the First Int. Conf. on Genetic Algorithms (Edited by J. Grefenstette),pp. 136-140. LawrenceErlbaum 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. Krishnamo orthy, 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.* Naturwissenchaften, vol. 76, pp. 579-581, 1989.

[57] C. Guéret, C. Prins, M. Sevaux, *Programmation linéiaire 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 Scho ol 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. l. 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 Soloing 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, imbreeding, 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 Plubishers, 1991.