République algérienne démocratique et populaire
Ministère de l'enseignement supérieur et de la recherche scientifique

UNIVERSITE : EL-HADJ LAKHDHAR – BATNA
Faculté des Sciences de l'Ingénieur
Département de Génie Industriel

Mémoire

Présenté au
Laboratoire d'Automatique et Productique LAP

En vue de l'obtention du diplôme de

Magister

Spécialité : Génie Industriel
Option : Génie Industriel

Par :

KEBABLA Mebarek
Ingénieur en Informatique
Université de Batna

Utilisation des stratégies Métaheuristiques pour l'ordonnancement des ateliers de type Job Shop

Soutenu le 02 juillet 2008 devant le jury composé de :

Nadia MOUSS,	M.C. *Université de Batna* :	Président
Hayet MOUSS,	M.C. *Université de Batna* :	Rapporteur
Abderrahmane DIB,	M.C. *C.U. Oum Elbouaghi* :	Examinateur
Mebarek DJEBABRA,	Pr. *Université de Batna* :	Examinateur
Hacène SMADI,	M.C. *Université de Batna* :	Examinateur
Djamel MOUSS,	M.C. *Université de Batna* :	Examinateur

Année : 2008

Remerciements

Je tiens à remercier, tout d'abord : Dr. N. MOUSS, Dr. A. DIB, Pr. M. DJEBABRA et Dr. H. SMADI pour nous avoir fait l'honneur d'être membres du jury. Ainsi que pour avoir consacré une partie de leur temps précieux pour lire et corriger ce mémoire.

Je souhaite remercier Dr. H. Mouss pour avoir accepter de me suivre dans l'élaboration de ce mémoire. Je lui suis particulièrement reconnaissant pour sa patience, son aide, sa disponibilité et ses nombreuses suggestions qui ont amélioré ce travail.

Je remercie également toute l'équipe de LAP (Laboratoire d'Automatique et Productique) de l'Université de Batna.

Je remercie tous ceux qui ont aidé de loin ou de près dans la réalisation de ce mémoire.

Merci à tout le monde.

Résumé

Ce travail traite l'application des Métaheuristiques au problème d'ordonnancement des ateliers de type Job Shop. Les méthodes retenues sont : les Algorithmes Génétiques, la Recherche Tabou et le Recuit Simulé. Le problème étudié est l'ordonnancement d'ateliers de type Job Shop simple, *n*-tâches, *m*-machines. L'objectif n'est pas de proposer de nouvelles techniques, mais plutôt d'explorer et justifier l'application des Métaheuristiques standards à ce problème.

Pour ce faire, une implémentation informatique des méthodes retenues selon plusieurs approches est réalisée, conduisant à développer une application informatique « Meta-JSS » (Metaheuristcs for Job Shop Scheduling). Cette application a permis de réaliser plusieurs séries d'expérimentations portant sur les différents choix implantés.

Ces expérimentations permettent de déduire que ces méthodes dans leur version basale sont capables de fournir de précieux outils adaptés aux problèmes industriels réels.

Mots clés : *Ordonnancement, Job Shop, Métaheuristiques, Algorithmes Génétiques, Recherche Tabou, Recuit Simulé.*

Abstract

This work treats the application of Metaheuristics to the Job Shop scheduling problem. The methods adopted are : Genetic Algorithms, Tabu Search and Simulated Annealing.
The studied problem is the simple Job Shop *n*-jobs, *m*-machines. Our objective is not to propose new techniques to resolve the problem, but rather to explore and justify the application of the Metaheuristics in their standard aspects to this problem.

With this aim, a computer implementation of the adopted methods according to several approaches is carried out, resulting in developing a software called "Meta-JSS" (Metaheuristcs for Job Shop Scheduling). This application (software) allowed us to carry out several series of experiments relating to the various established choices.

These experiments make it possible to deduce that these methods in their basal (simple) version are able to provide invaluable tools adapted to the real industrial problems.

Keywords : *Scheduling, Job Shop, Metaheuristics, Genetic Algorithms, Tabu Search, Simulated Annealing.*

SOMMAIRE

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

INTRODUCTION GENERALE

Introduction générale

La situation actuelle des entreprises manufacturières a connu de grands changements. S'intéresser uniquement à *produire*, comme c'était le cas autrefois, n'est plus un atout de réussite. L'objectif majeur de toute entreprise actuelle n'est pas seulement la *productivité* mais aussi et surtout la *compétitivité* [Vac, 00]. Cette compétitivité passe forcément par la diversification du travail, un outil de production de plus en plus flexible et une gestion de production fiable qui devient de plus en plus complexe [Gia, 88]. L'amélioration de la gestion de production est alors, à la base de toute tentative de changement [Jav, 04]. Cette fonction vise en effet à organiser le fonctionnement du système de production, et à mieux gérer ses différentes opérations.

Les problèmes d'ordonnancement d'ateliers constituent sûrement pour les entreprises une des difficultés importantes de leurs systèmes de gestion et de pilotage de la production. En effet, c'est à ce niveau que doivent être prises en compte les caractéristiques réelles multiples et complexes des ateliers. Dans ce type d'ordonnancement, les ressources sont généralement des machines, et chaque travail à ordonnancer concerne un produit ou un lot de produits à fabriquer en respectant les gammes de fabrication.

Vu la diversité des systèmes de production au niveau opérationnel, les problèmes étudiés en ordonnancement sont extrêmement divers. Théoriquement, la littérature a l'habitude de distinguer : le Job Shop, le Flow Shop, l'Open Shop, le Mix Shop, les machines parallèles etc.

Parmi les problèmes d'ordonnancement les plus difficiles et les plus étudiés, nous avons le problème d'ordonnancement d'ateliers de type *Job Shop*. Considéré comme un modèle d'ordonnancement intéressant, le problème de Job Shop est très représentatif du domaine général de production. Ce problème correspond, réellement, à la modélisation d'une unité de production disposant de moyens polyvalents utilisés suivant des séquences différentes en fonction des produits [Duv, 00]. L'objectif consiste à programmer la réalisation des produits de manière à optimiser la production, en respectant un certain nombre de contraintes sur les machines utilisées pour effectuer chaque opération élémentaire entrant dans la fabrication de chaque produit.

La résolution de ce problème de manière optimale s'avère dans la plupart des cas impossible à cause de son caractère fortement combinatoire. Les méthodes exactes requièrent un effort

calculatoire qui croît exponentiellement avec la taille du problème. Alors, des méthodes approchées ont été proposées pour résoudre ce problème en temps raisonnable. Parmi ces méthodes, apparaissent celles dites "Métaheuristiques" dont trois méthodes ont démontré leur efficience dans nombreuses applications : les Algorithmes Génétiques appartiennent aux méthodes évolutives ; la Recherche Tabou et le Recuit Simulé qui sont basés sur le principe de la recherche locale.

L'avantage de ces Métaheuristiques se résume dans leur adaptabilité à tous les problèmes, et le bon compromis qu'elles présentent entre le temps de recherche, et la qualité de résolution. Grâce à ces Métaheuristiques, on peut proposer des solutions approchées pour des problèmes difficiles et de plus grande taille qu'il était impossible de traiter auparavant.

La visée du travail présenté dans ce mémoire est de traiter la problématique d'application des Métaheuristiques au problème d'ordonnancement de Job Shop. Ce travail porte sur le problème de Job Shop simple qui est présentatif du contexte général de l'industrie, sans s'attacher à répondre aux problèmes industriels réels. Nous voulons, par ce travail, montrer qu'il est possible d'arriver à des résultats satisfaisants avec des méthodes de base, peu exigeantes en conception et implantation, en temps de résolution et en espace mémoire, au lieu de chercher des approches plus sophistiquées. Nous concentrons nos travaux sur l'implémentation informatique de ces méthodes dans leurs formes simples (standards) avec plusieurs alternatives pour chacune, pour dégager des résultats prouvant leur efficacité.

La contribution du travail exposé dans ce mémoire se voit essentiellement au travers les trois points suivant :

- Explorer l'application des Métaheuristiques, comme paradigme important de résolution des problèmes combinatoires, dans le domaine de la production industrielle. Il ne s'agit pas de proposer de nouveaux opérateurs ou techniques, mais de réutiliser des principes déjà proposés par des chercheurs et les implanter à l'aide d'un langage informatique.
- Justifier l'utilisation de ces Métaheuristiques dans leurs formes basales, et montrer leur potentiel pour traiter les problèmes de l'ordonnancement de Job Shop.
- Proposer une application informatique désignée « Meta-JSS », par implémentation des différents choix étudiés concernant les opérateurs, paramètres et ingrédients de ces méthodes.

L'organisation du mémoire découle naturellement de cette problématique traitée. Il est structuré en cinq chapitres qui permettent un cadrage progressif du sujet.

Dans le premier chapitre, le contexte général de l'ordonnancement et des problèmes d'ordonnancement est décrit afin de replacer le sujet dans son cadre général. Le chapitre commence par une présentation concise des systèmes de production et de la gestion de production où apparaît le rôle primordial de l'ordonnancement. La suite du chapitre est consacrée au cadrage de la problématique générale de l'ordonnancement par présentation des points essentiels : définition, modélisation, classification, critères d'optimisation et organisation des systèmes concernés.

Le deuxième chapitre fait un recentrage sur le problème d'ordonnancement de Job Shop. La spécificité du problème est mise en évidence via présentation de sa formulation, sa modélisation et sa difficulté de résolution. Un tour d'horizon des différentes approches de résolution est fait ensuite, en indiquant pour chaque approche les principaux travaux remarqués. Finalement, sont évoqués les fameux benchmarks du problème.

Le troisième chapitre est consacré à la présentation des trois Métaheuristiques constituant notre sujet d'application, qui sont les *Algorithmes Génétiques,* la *Recherche Tabou* et le *Recuit Simulé.* Ainsi, après des généralités sur les Métaheuristiques, le principe général et les mécanismes de base de chaque méthode sont donnés.

Dans le quatrième chapitre, nous décrivons notre approche d'implémentation des trois Métaheuristiques. Pour chaque méthode, plusieurs choix de stratégies et de mécanismes sont adoptés. Pour ces choix, les algorithmes d'implantation sont donnés, en s'aidant parfois d'exemples pour illustrer le principe. Le chapitre termine par une description de l'application informatique résultante de l'implémentation des différentes techniques envisagées.

Le cinquième chapitre, fait appel à une démarche expérimentale afin de montrer l'efficience des Métaheuristiques utilisées dans la résolution du problème étudié. Avec l'outil développé, plusieurs expérimentations sont effectuées sur des Benchmarks retenus comme échantillon de test. Les résultats obtenus sont discutés et analysés afin de ressortir certaines conclusions importantes.

Finalement, un bilan et des perspectives de recherche issues de ce travail sont présentés en conclusion.

CHAPITRE I

L'ordonnancement des activités de production

Résumé : *L'objectif de ce chapitre introductif aux problèmes d'ordonnancement est de faire un cadrage de notre thème. Nous présentons alors, au travers ce chapitre les points essentiels concernant les problèmes d'ordonnancement d'une façon générale : la place de l'ordonnancement en gestion de production, définition et modélisation du problème d'ordonnancement, les critères d'optimisation, un formalisme de classification de ces problèmes, et finalement, les modèles d'organisations importants (Flow Shop, Job Shop, Open Shop,…).*

CHAPITRE I

L'ordonnancement des activités de production

1. Introduction

Dans ce chapitre introductif, nous nous intéressons aux problèmes d'ordonnancement d'une manière générale afin de situer la problématique de notre travail. Il s'agit de présenter les concepts et les bases fondamentales du problème d'ordonnancement.

Dans un premier temps, il est nécessaire de replacer l'ordonnancement dans le cadre général des systèmes de production. Nous rappelons alors, dans la deuxième section, des notions de base concernant la production et la gestion de production, en mettant l'accent sur le rôle de l'ordonnancement au sein de ces systèmes. La troisième section vise à présenter le problème de l'ordonnancement en précisant sa définition et les différents éléments qui le déterminent. La quatrième section du chapitre présente les critères d'optimisation comme élément de grande importance. Un formalisme de classification des problèmes d'ordonnancement est évoqué dans la cinquième section. La dernière section est consacrée à la description des différentes organisations d'ateliers qui déterminent des problèmes d'ordonnancement spécifiques.

2. Production et gestion de production

La production est le processus conduisant à la création de produits par l'utilisation et la transformation de ressources [Gia, 88]. Le processus de production est alors, constitué d'un ensemble d'opérations qui sont les activités conduisant à la création de biens et de services [Jav, 04].

Le système de production, est l'ensemble de ressources réalisant une activité de production. C'est un ensemble de moyens divers : humains, matériels, informationnels et d'autres, constituant un tout, dont l'objectif est la réalisation de biens ou de services [Fon, 99].

Les systèmes de production industrielle se sont considérablement diversifiés et compliqués [Blo, 04] [Vac, 00]. En effet, ils peuvent se décomposer en plusieurs sous-systèmes, qui s'intègrent en vue d'assurer la pérennité et la compétitivité de l'entreprise.

2.1. Décomposition du système de production

Classiquement, un système de production peut se décomposer en trois sous-systèmes : le système physique de production, le système de décision et le système d'information [Let, 01] [Jav, 04].

Cette décomposition est structurée en fonction de la nature des flux qui traversent chaque système *i.e.* flux de décisions, flux d'informations et flux physique (figure 1.1).

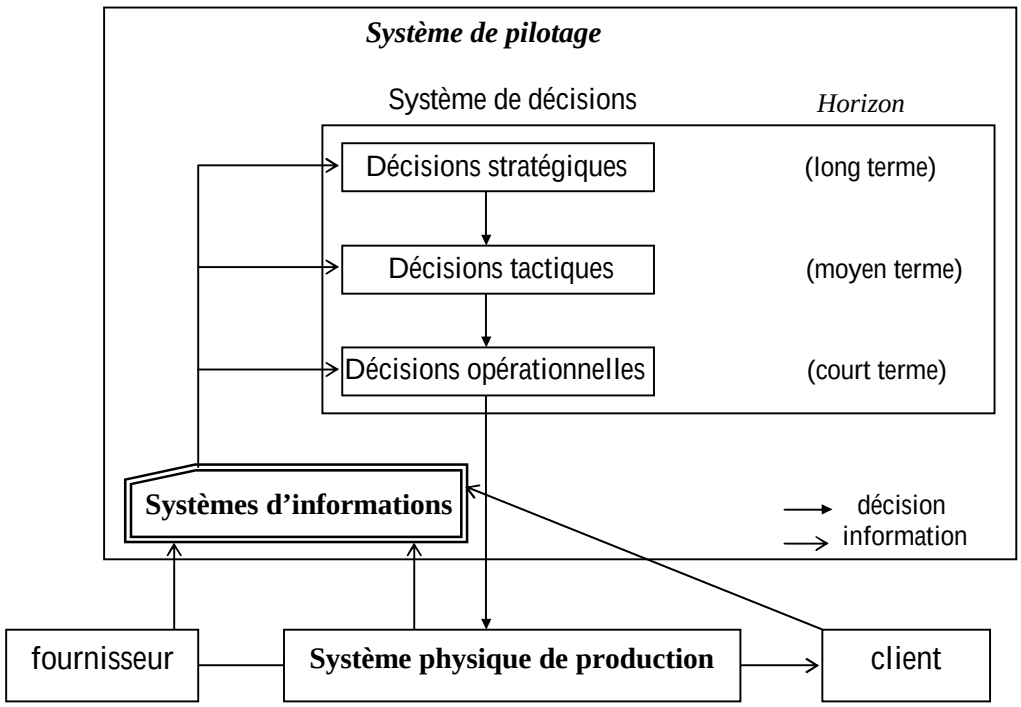


Figure 1.1 : Les sous-systèmes constituant le système de production d'après [Fon, 99] [Let, 01].

- **Le système physique de production :** Transforme les matières premières ou composants en produits finis. Il est constitué de ressources humaines et physiques.
- **Le système de décision :** Contrôle le système physique de production. Il en coordonne et organise les activités en prenant des décisions basées sur les données transmises par le système d'information.
- **Le système d'information :** Intervient à plusieurs niveaux : à l'interface entre les systèmes de décision et de production ; à l'intérieur du système de décision, pour la gestion des informations utilisées lors de prises de décisions ; et à l'intérieur du système physique de

production. Son rôle est de collecter, stocker et transmettre des informations de différents types [Fon, 99].

2.2. La gestion de production

Le système de décision "drainé" par le système d'information constitue ce qu'on appelle le système de gestion de production.

2.2.1. Définition et rôle de la gestion de production

La gestion de production est « un ensemble de processus qui permet de mener à bien la fabrication de produits à partir d'un ensemble de données et de prévisions » [Vac, 00].

F. Blondel [Blo, 04], définit la gestion de production comme étant « la fonction qui permet de réaliser les opérations de production en respectant les conditions de qualité, délai, coûts qui résultent des objectifs de l'entreprise ».

En fait, la gestion de production s'occupe d'un ensemble de problèmes liés à la production tels que la gestion des données, la planification, le contrôle (suivi) de la production, la gestion des stocks, la prévision, l'ordonnancement etc. (figure 1.2).

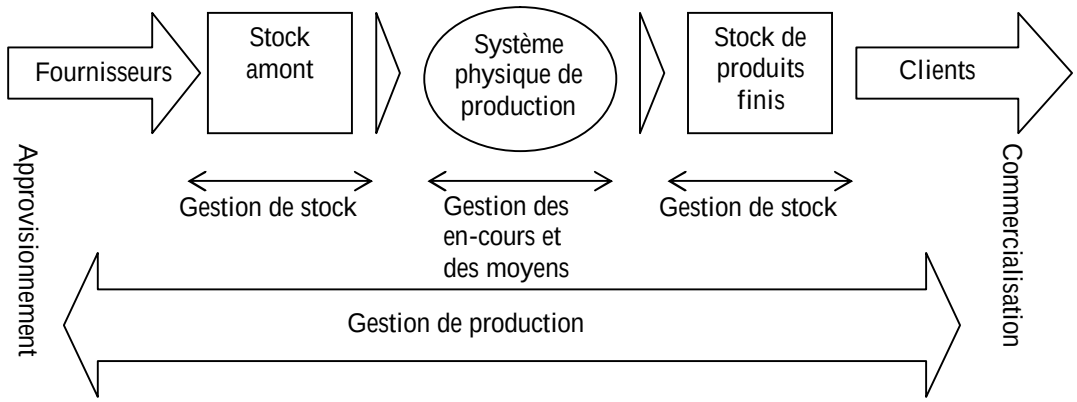


Figure 1.2 : Objectifs de la gestion de production d'après [Jav, 04].

La gestion de production est une fonction complexe. Cette complexité conduit généralement à l'hiérarchiser afin de la simplifier et de rendre possible la gestion du système.

2.2.2. Organisation hiérarchique de la gestion de production

Les niveaux hiérarchiques de la gestion de production couramment retenus sont en nombre de trois : stratégique, tactique et opérationnel [Gia, 88] [Vac, 00] [Fon, 99] [Let, 01] :

a. Le niveau stratégique :

Il s'agit de la formulation de la politique à long terme de l'entreprise (à un horizon de plus de deux ans). Elle porte essentiellement sur la gestion des ressources durables, afin que celles-ci soient en mesure d'assurer la pérennité de l'entreprise.

b. Le niveau tactique :

Il s'agit de décisions à moyen terme. Elles assurent la liaison entre le niveau stratégique et le niveau opérationnel. L'objectif est de produire au moindre coût pour satisfaire la demande prévisible, en s'inscrivant dans le cadre fixé par le plan stratégique de l'entreprise.

c. Le niveau opérationnel :

Il s'agit des décisions à court et à très court terme. C'est une gestion quotidienne pour faire face à la demande au jour le jour, dans le respect des décisions tactiques.

Parmi les décisions opérationnelles, on trouve : la gestion des stocks, la gestion de la main d'œuvre, la gestion des équipements [Blo, 04].

2.3. Rôle de l'ordonnancement en gestion de production

Le modèle général en gestion de production, décompose les décisions en trois niveaux : stratégique, tactique (planification à long et à moyen terme) et opérationnel (pilotage et suivi quotidien des flux de matières et du travail). Cette hiérarchisation se traduit par une échelle de responsabilités (direction, cadre, agent…) et par un enchâssement des horizons de temps (long terme, moyen terme, cours terme) [Lop & Esq, 99].

Dans ce schéma pratique et classique, il est impossible de résoudre le problème d'ordonnancement globalement (au niveau supérieur), d'autant plus que les évènements aléatoires endogènes (pannes de machines, grèves,...) et exogènes (commandes imprévues et prioritaires,...) qui interviennent constamment nécessitent de recalculer fréquemment l'ordonnancement. Donc les problèmes d'ordonnancement sont traités aux niveaux inférieurs.

La place de l'ordonnancement varie entre le niveau tactique et le niveau opérationnel [Jef & al., 06] [Lop & Esq, 99]. Il s'occupe de la réalisation des décisions venant de niveaux supérieurs. Il couvre un ensemble d'actions qui transforment les décisions de fabrication définies par le programme directeur de production en instructions d'exécution détaillées destinées à contrôler et piloter à court terme l'activité des postes de travail.

En sortie de la fonction ordonnancement, on obtient un planning ou ordonnancement, qui restitue l'affectation des tâches fournies en entrée à des dates précises pour des durées

déterminées sur les différentes ressources. Ce planning cherche à satisfaire des objectifs, en respectant le plus possible les contraintes imposées [Jav, 04].

Pratiquement, la fonction ordonnancement se décompose en trois sous-fonctions (figure.1.2) [Jav, 04] :

- L'élaboration des ordres de fabrication (OF) : cette tâche consiste à transformer les informations du programme directeur de production (suggestion de fabrication) en OF.
- L'élaboration du planning d'atelier : cette tâche consiste à déterminer, en fonction des ordres de fabrication et de la disponibilité des ressources, le calendrier prévisionnel de fabrication.
- Le lancement et le suivi des opérations de fabrication.

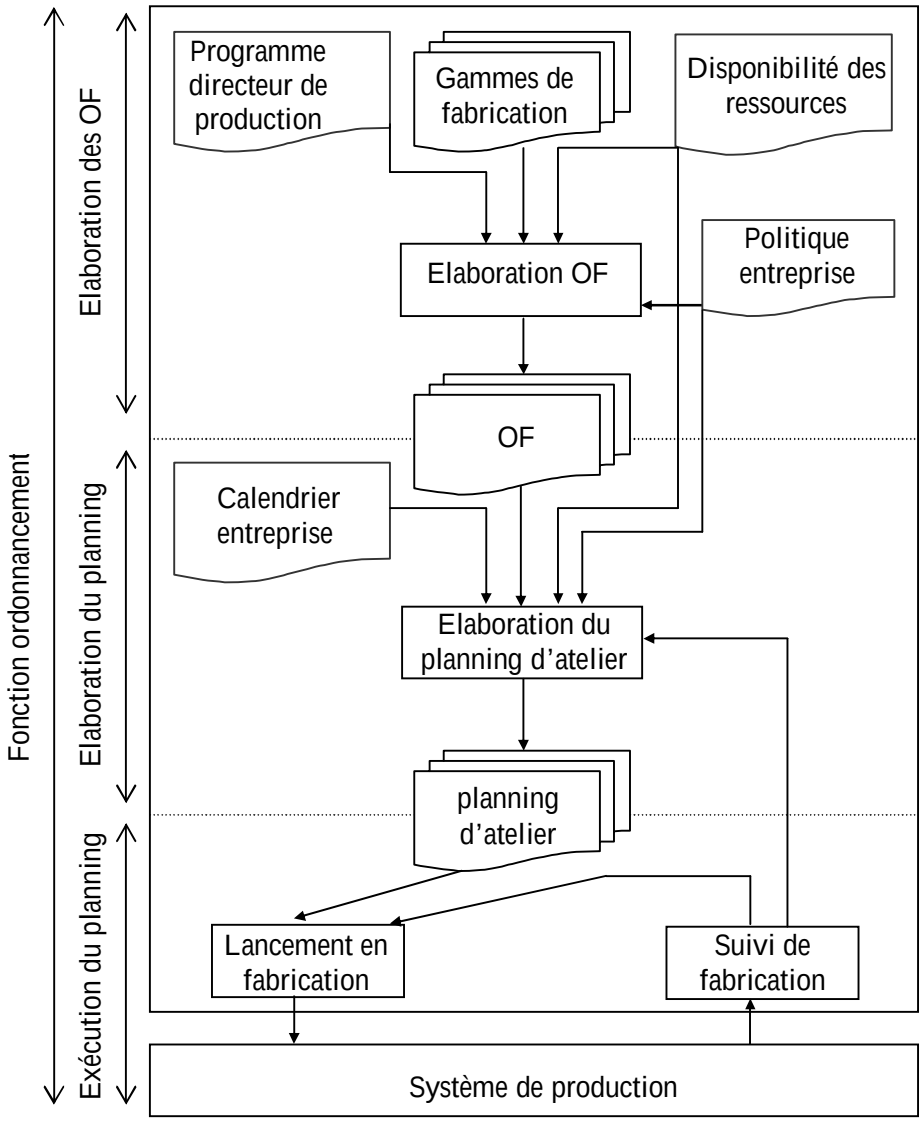


Figure 1.3 : Sous-fonctions de l'ordonnancement dans l'atelier [Jav, 04].

3. Présentation du problème d'ordonnancement

En dépit de la complexité réelle des problèmes d'ordonnancement, il existe un modèle standard sur lequel est basée la définition du problème. Dans ce qui suit, nous nous intéressons à décrire ce modèle théorique du problème d'ordonnancement.

3.1. Définition du problème d'ordonnancement

Nombreuses sont les définitions proposées au problème d'ordonnancement. Dans ces définitions, on retrouve l'aspect commun de l'affectation de ressources aux tâches, en recherchant une certaine optimisation.

On peut évoquer parmi ces définitions ce qui suit :

« Etant donné un ensemble de tâches à accomplir, le problème d'ordonnancement consiste à déterminer quelles opérations doivent être exécutées, et à assigner des dates et des ressources à ces opérations de façon à ce que les tâches soient, dans la mesure du possible, accomplies en temps utile, au moindre coût et dans les meilleures conditions » [Vac, 00].

« Résoudre un problème d'ordonnancement consiste à ordonnancer *i.e.* programmer ou planifier dans le temps, l'exécution des tâches en leur attribuant les ressources nécessaires, matérielles ou humaines de manière à satisfaire un ou plusieurs critères préalablement définis, tout en respectant les contraintes de réalisation » [Lop & Esq, 99].

D'une façon générale, il y a problème d'ordonnancement quand il existe [Jef & al., 06] :

- Un ensemble de travaux (tâches ou encore jobs ou lots) à réaliser,
- Un ensemble de ressources à utiliser par ces travaux,
- Un programme (calendrier) à déterminer, pour allouer convenablement les ressources aux tâches.

Le problème de l'ordonnancement consiste donc, à programmer dans le temps l'exécution de certaines tâches, en leur allouant les ressources requises, et en respectant les contraintes imposées afin d'atteindre certains objectifs.

Pour la notion d'ordonnancement, on peut envisager sa définition de deux points de vue :

- Comme une fonction de la gestion de production :

  Se définit alors, comme étant « l'ensemble des actes de gestion visant à l'établissement d'un ordre de déroulement des opérations de production qui puisse permettre d'atteindre un certain optimum économique préalablement défini » [Blo, 04].

Ou encore « c'est une fonction permettant de déterminer les priorités de passage des produits et leur affectation sur les différentes machines » [Hen, 99].

- Comme une solution réalisable au problème d'ordonnancement défini ci-avant. C'est le sens qui correspond à l'emploi du terme dans la suite de ce mémoire.

3.2. Eléments du problème d'ordonnancement

Dans la définition du problème d'ordonnancement, quatre éléments fondamentaux interviennent : les tâches, les ressources, les contraintes et les objectifs. Alors, la formulation et la description de ce problème se fait par la détermination de ces quatre éléments dits "de base" [Hen, 99].

3.2.1. Les tâches

Une tâche est un travail mobilisant des ressources et réalisant un progrès significatif dans l'état d'avancement du projet compte tenu du niveau de détail retenu dans l'analyse du problème [Gia, 88].

D'une façon plus schématique : une tâche qu'on note i est une entité élémentaire de travail localisée dans le temps par une date de début t_i ou de fin c_i, dont la réalisation nécessite une durée p_i telle que $p_i = c_i - t_i$, et qui utilise des ressources k avec une intensité a_i^k [Lop & Esq, 99].

Généralement trois paramètres caractérisent une tâche [T'ki & Bil, 06] :

- La durée opératoire p_i (*processing time*) : c'est la durée d'exécution de la tâche.
- La date de disponibilité r_i (*release time*) : c'est la date de début au plus tôt.
- La date d'échéance d_i (*due date*) : c'est la date de fin au plus tard.

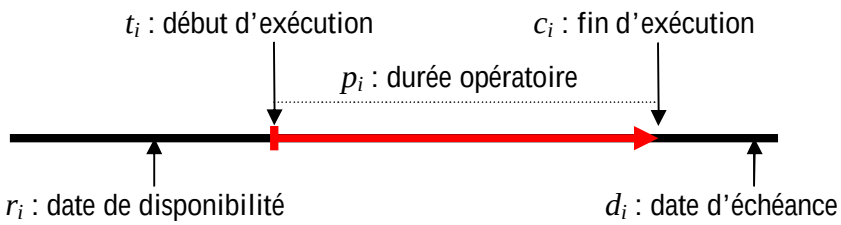


Figure 1.4 : Caractéristiques d'une tâche i.

Selon les problèmes, une tâche peut être exécutée par morceau ou sans interruption. Notre cas (Job Shop) appartient à la première catégorie, chaque tâche est constituée d'un ensemble d'opérations liées entre elles par des contraintes technologiques. Effectivement, en production manufacturière, on distingue souvent plusieurs phases dans l'exécution d'une tâche : la préparation, la phase principale, la finition, le transport etc.

Nous remarquons ici que certains auteurs utilisent le terme de "*tâche*" pour désigner une opération, et le terme de "*travail*" ou "*job*" pour l'ensemble des opérations constituant le même travail. Dans ce mémoire, nous adoptons la terminologie courante, par l'utilisation des termes tâche et opération, où une tâche est constituée d'un ensemble d'opérations.

3.2.2. Les ressources

Une ressource est un moyen destiné à être utilisé pour la réalisation d'une tâche, et disponible en quantités limitées [Lop & Esq, 99].

Dans le contexte industriel, les ressources peuvent être des machines, des ouvriers, des équipements, des locaux ou encore de l'énergie, des budgets, etc.

La disponibilité d'une ressource peut varier dans le temps suivant une fonction $a_k(t)$. Cette disponibilité qui s'appelle la capacité a_k de la ressource k, est une caractéristique qui détermine la quantité de la ressource.

Plusieurs classifications des ressources peuvent être distinguées :

a. Ressources renouvelables *vs*. ressources consommables

Une ressource est dite renouvelable, si après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible en même quantité comme : les hommes, les machines, l'espace, etc.

Par contre, une ressource est consommable, lorsque elle devient non disponible après avoir été utilisée ; comme la matière première, l'énergie, etc. [Lop & Esq, 99].

b. Ressources disjonctives *vs*. ressources cumulatives

Une ressource est dite disjonctive (ou non partageable), si elle ne peut être affecter qu'à une seule tâche à la fois. Ce type concerne surtout les ressources renouvelables.

Par contre, une ressource cumulative (partageable) peut être utilisée par plusieurs tâches simultanément (équipe d'ouvriers, poste de travail, …) [Lop & Esq, 99].

Dans notre problème, les ressources qui sont des machines, sont renouvelables et disjonctives.

3.2.3. Les contraintes

Les contraintes expriment des restrictions sur les valeurs que peuvent prendre conjointement une ou plusieurs variables de décision [Lop & Esq, 99]. Donc les contraintes représentent les limites imposées par l'environnement, tandis que l'objectif est le critère d'optimisation.

Plusieurs types de ces contraintes doivent être respectés. On distingue notamment [Bap, 98] [Lop & Esq, 99] [T'ki & Bil, 06] :

a. Les contraintes de localisation temporelle :

Sont issues des impératifs de gestion, et relatives aux dates limites des tâches ou du projet entier. On a surtout :

- La date de disponibilité r_i (avant laquelle la tâche ne peut pas commencer) ;
- La date d'échéance d_i (avant laquelle la tâche doit être achevée).

b. Les contraintes d'enchaînement :

Nous qualifions de contrainte d'enchaînement ou de succession, une contrainte qui lie le début ou la fin de deux activités par une relation linéaire [Bap, 98]. Ce sont des contraintes imposées généralement par la cohérence technologique (les gammes opératoires dans le cas d'ateliers) qui décrivent des positionnements relatifs devant être respectés entre les tâches.

D'autres contraintes plus spécifiques entre deux tâches ou plus, telles que : la synchronisation, la simultanéité, le recouvrement,… sont également imposées dans certains systèmes [Lop & Esq, 99].

c. Les contraintes de ressources :

Une contrainte de ressources représente le fait que les activités utilisent une certaine quantité de ressources, tout au long de leur exécution [Bap, 98]. Ces contraintes sont essentiellement soit [Lop & Esq, 99] :

- Des contraintes d'utilisation des ressources qui expriment la nature, la quantité et les caractéristiques d'utilisation de ces ressources.
- Des contraintes de disponibilité des ressources qui déterminent les quantités des ressources disponibles au cours du temps.

On peut distinguer les contraintes suivant qu'elles sont strictes ou non. Les contraintes strictes sont des obligations à respecter. Alors que, les contraintes dites "relâchables" (appelées aussi préférences) peuvent éventuellement n'être pas satisfaites [Let, 01].

3.2.4. Les objectifs

Tout ordonnancement est guidé par un ou plusieurs objectifs qu'il doit chercher leur optimisation. Les objectifs que doit satisfaire un ordonnancement sont variés. D'une manière générale, on distingue plusieurs classes d'objectifs concernant un ordonnancement donné [Lop & Esq, 99] :

- Les objectifs liés au temps : on trouve par exemple, la minimisation du temps total d'exécution, du temps moyen d'achèvement, des durées totales de réglage ou des retards par rapport aux dates de livraison,
- Les objectifs liés aux ressources : par exemple, maximiser la charge d'une ressource ou minimiser le nombre de ressources nécessaires pour réaliser un ensemble de tâches.
- Les objectifs liés au coût : ces objectifs sont généralement de minimiser les coûts, de lancement, de production, de stockage, de transport, etc.
- Les objectifs liés à l'énergie ou au débit.

Dans la section suivante nous reviendrons avec une description plus poussée sur les différents objectifs que doit satisfaire l'ordonnancement.

4. Les critères d'optimisation

Dans la résolution d'un problème d'ordonnancement, on peut choisir entre deux grands types d'objectifs, visant respectivement à *l'optimalité* des solutions (trouver la meilleure solution) ou plus simplement à leur *admissibilité* (trouver une solution).

La recherche de l'optimalité impose à mesurer la qualité d'une solution de répondre aux exigences du problème étudié. Dans ce but, des critères d'optimisation sont définis et utilisés.

4.1. Classification des critères d'optimisation

Les critères que doit satisfaire un ordonnancement sont variés. D'une manière générale, on peut distinguer trois catégories importantes [Hen, 99] :

- Les critères liés aux dates de fin et de livraison,
- Les critères liés aux volumes des encours,
- Les critères liés à l'utilisation des ressources.

Avant de dénombrer les critères importants de chacune de ces catégories, rappelons-nous les notations suivantes :

- c_i *(completion time)* : la date d'achèvement de la dernière opération de la tâche i,
- d_i *(due date)* : la date de livraison de i,
- r_i *(release time)* : la date de disponibilité de i,
- p_{ij} : la durée de l'opération j de la tâche i,
- $F_i = c_i - r_i$ *(flow time)* : la durée de séjour de la tâche i depuis sa disponibilité jusqu'à la fin de son exécution,

- $L_i = c_i - d_i$ *(lateness)* : le retard algébrique, qui détermine le retard d'exécution de la tâche i par rapport à sa date d'échéance,
- $T_i = max(L_i,0)$ *(tardiness)* : le retard absolu de la tâche i,
- $E_i = max(-L_i,0)$ (*earliness)* : l'avancement de la tâche i.

4.1.1. Les critères liés aux dates de fin et de livraison

Ce sont des critères liés aux paramètres temporels caractérisant les tâches. Les plus utilisés sont [Tana & al., 94] [Pin, 95] [Fre, 82] :

- $C_{max} = \max(c_i)$: (*Makespan*), est la plus grande date d'achèvement des différentes tâches. C'est le temps nécessaire pour terminer toutes les tâches. Le makespan, est le critère le plus fréquemment utilisé [Bap, 98].
- $C_\Sigma = \Sigma(c_i)$: la somme des dates de fin des tâches.
- $\Sigma w_i C_i$: la somme des dates de fin pondérées.
- $\overline{C}$: la moyenne arithmétique des dates de fin des tâches.
- $F_{max} = max(F_i)$: le maximum des temps de séjour de l'ensemble des tâches.
- $L_{max} = max(L_i)$: le maximum des retards algébriques de l'ensemble des tâches.
- $T_{max} = max(T_i)$: le maximum des retards absolus de l'ensemble des tâches.
- $E_{max} = max(E_i)$: le maximum des temps d'avancement de l'ensemble des tâches.

F_Σ, L_Σ ,T_Σ , E_Σ qui correspondent aux temps de : séjour total, retard total, retard absolu total et l'avancement total de l'ensemble des tâches, sont utilisés aussi [Jen, 01].

D'autres critères correspondant aux moyennes arithmétiques de ces grandeurs, sont également utilisé : $\overline{F}$, $\overline{L}$, $\overline{T}$, $\overline{E}$ [T'ki & Bil, 06].

Tous ces critères sont à minimiser. Certains d'entre eux ne sont pas utilisés seuls comme : E_{max} ,E_Σ… mais habituellement en association avec d'autres critères afin de construire un critère composé plus performant par exemple : aT_Σ + bE_Σ [Jen, 01].

4.1.2. Les critères liés aux volumes des encours

Soit t_o la date de début de l'opération O et p_o sa durée d'exécution. A chaque instant t, on peut calculer les mesures suivantes caractérisant les volumes des encours [Hen, 99] :

- Le nombre de tâches en cours d'exécution :

$$N_p(t) = \sum_{i=1}^{n} e_i(t) \ ; \text{ avec } e_i(t) = \begin{cases} 1 & si\ (t_o \leq t < t_o + p_o) \\ 0 & sinon \end{cases}$$

L'optimisation de l'ordonnancement vise la maximisation de ce critère. Plus la valeur de ce critère est grande, plus le nombre de machines en attente est faible.

- Le nombre de tâches en attente d'exécution :

$$N_w(t) = \sum_{i=1}^{n} e_i(t) \ ; \text{ avec } e_i(t) = \begin{cases} 1 & si\ (t_o + p_o \leq t < t_{o+1}) \\ 0 & sinon \end{cases}$$

La minimisation de ce critère permet de diminuer le nombre de tâches en attente dans les stocks intermédiaires des machines, et par conséquent de réduire la capacité intrinsèque des stocks.

- Le nombre de tâches terminées sur la dernière machine :

$$N_f(t) = \sum_{i=1}^{n} e_i(t) \ ; \text{ avec } e_i(t) = \begin{cases} 1 & si\ (C_i \leq t) \\ 0 & sinon \end{cases}$$

La maximisation de ce critère permet de répondre dans les meilleurs délais aux demandes des clients.

On peut trouver dans certains cas que : $N_p(t) + N_w(t) + N_f(t) = N$, où N est le nombre de tâches à ordonnancer pour un horizon fixé. Par conséquent, les objectifs de production dans ce cas peuvent être complètement antagonistes [Hen, 99].

4.1.3. Les critères liés à l'utilisation des ressources

L'exploitation des ressources peut également être un objectif majeur du gestionnaire. Maximiser la charge d'une ressource ou l'utilisation moyenne des ressources, ou encore minimiser le temps d'inactivité de l'ensemble des ressources, sont des objectifs de ce type.

- L'utilisation moyenne des ressources :

$$\overline{U} = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} \right) \Big/ \left(\sum_{j=1}^{m} M_j \times C_{max} \right)$$

Ce critère permet de signaler éventuellement la sous-utilisation des ressources de l'atelier, en indiquant les périodes pleines et les périodes creuses de la chaîne de fabrication [Hen, 99].

- Le temps d'inactivité de l'atelier (*Idle time*) [Hen, 99] :

$$I = \sum_{j=1}^{m} \left(C_{max} - \sum_{i=1}^{n} p_{ij} \right)$$

C'est la somme des tranches de temps perdu pendant l'exécution des tâches sur les machines. La minimisation de cet indicateur permet une meilleure exploitation des ressources.

4.2. Propriétés des critères

Deux propriétés importantes des critères de performance qui sont : la régularité des critères et la notion de dominance décrivent certaines relations entre un critère donné et l'ordonnancement correspondant.

4.2.1. Critères réguliers

Soient $C_i(x)$ et d(x), respectivement la date de fin de la dernière opération de la tâche J_i et la valeur d'un critère d appliqué à un ordonnancement x ; d est fonction des dates de fin d'ordonnancement C_i.

Considérons deux ordonnancements x, x'. Si : $C_i(x) \leq C_i(x') \ \forall\ i = 1,2,\ldots n \Rightarrow$ d$(x) \leq$ d(x'), alors le critère d est dit régulier [T'ki & Bil, 06].

Autrement dit, un critère est régulier, s'il est une fonction décroissante des dates de fin d'exécution des opérations. Donc, on peut le dégrader en avançant l'exécution d'une tâche.

Parmi les critères réguliers : C_{max}, T_{max}, L_{max}, $\overline{L}$, $\overline{T}$. Par contre, $\overline{E}$, E_{max}, I sont des critères non réguliers [Let, 01].

4.2.2. Notion de dominance

Un sous-ensemble de solutions est dit dominant pour l'optimisation d'un critère donné, s'il contient au moins un optimum pour ce critère. De manière analogue, on définit la dominance par rapport à un ensemble de contraintes lorsque le sous-ensemble dominant contient au moins une solution *admissible* (qui satisfait toutes les contraintes), si elle en existe. La recherche d'une solution optimale ou admissible peut ainsi être limitée à un sous-ensemble dominant.

4.3. Relations entre les critères

Le traitement de l'ordonnancement dans la littérature s'est tout d'abord orienté vers une optimisation monocritère. L'environnement manufacturier évoluant rapidement, les objectifs des entreprises se sont diversifiés et le processus d'ordonnancement est devenu de plus en plus multicritères [Let, 01].

Les critères que doit satisfaire un ordonnancement sont variés. L'existence de plusieurs relations d'équivalence, de similitude et de réduction entre ces critères permet de déduire

certains des autres. En effet, une solution optimale pour un critère donné peut s'avérer optimale pour un autre.

Sur la figure 1.5 qui présente les relations entre certains critères importants, on lit : l'arc d• d' signifie qu'un problème avec un critère d peut se réduire en problème avec le critère d', en considérant la condition associée à la relation.

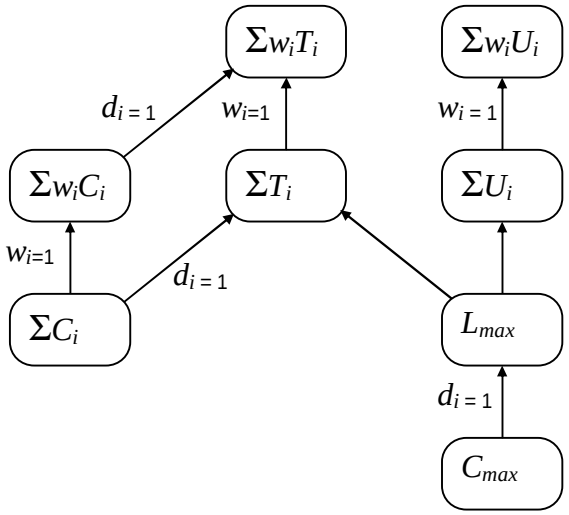


Figure 1.5 : Relations entre les critères d'optimisation [Pin, 95] [T'ki & Bil, 06].

5. Classification des problèmes d'ordonnancement

Etant donnée la diversité des problèmes d'ordonnancement, une classification formelle est nécessaire pour distinguer les différents problèmes. L'intérêt de telle classification réside, entre autre, dans [Hen, 99] :

- Le besoin de recenser les problèmes qui sont de plus en plus divers, afin de mieux les identifier ;
- La détermination de leur complexité et de leurs outils de résolution s'ils existent.

Plusieurs classifications ont été proposées. Il ne s'agit pas de formalismes différents, mais d'une approche qui commence avec les travaux de Conway & al., reprise et améliorée chaque fois par d'autres chercheurs. La classification "standard" proposée par Graham & al. en 1979, est couramment utilisée dans la littérature.

Dans cette classification, les problèmes d'ordonnancement sont décrits par la notation [Hen, 99] [Jef & al., 06] :

•/Š/• :

a) Le paramètre • se compose de deux sous-paramètres • $_1$ et • $_2$

- $\bullet_1 \in \{\phi, P, Q, R, O, F, J\}$ décrit l'organisation de l'atelier :

  $\bullet_1 = \phi$: une seule machine.

  $\bullet_1 = P$: machines parallèles identiques. Il s'agit de machines de même cadence, disposées en parallèle pouvant exécuter toutes les tâches.

  $\bullet_1 = Q$: machines parallèles uniformes. Les cadences des machines sont différentes, mais restent indépendantes des tâches.

  $\bullet_1 = R$: machines parallèles non liées. Les cadences des machines sont différentes deux à deux et dépendent des tâches exécutées.

  $\bullet_1 = F$: Flow Shop, chaque tâche doit être exécutée sur l'ensemble des machines disposées en série, selon un même routage.

  $\bullet_1 = O$: Open Shop, les tâches doivent être exécutées sur certaines machines sans restriction sur le routage des tâches.

  $\bullet_1 = J$: Job Shop, les tâches doivent être exécutées sur l'ensemble des machines disposées en série, mais peuvent avoir des routages différents.

- $\bullet_2 \in \{\phi, m\}$ détermine le nombre de machines dans le système.

  $\bullet_2 = \phi$: le nombre de machines est variable.

  $\bullet_2 = m$: le nombre de machines est constant et égal à m.

b) Le paramètre $\check{S}$ se compose de cinq sous-paramètres : $\check{S}_1$, $\check{S}_2$, $\check{S}_3$, $\check{S}_4$ et $\check{S}_5$.

- $\check{S}_1 \in \{\phi, prmpt\}$ indique l'autorisation ou non de la préemption :

  $\check{S}_1 = \phi$: la préemption n'est pas autorisée (lorsqu'une opération est commencée, elle doit être terminée avant de pouvoir exécuter une autre opération sur la même machine).

  $\check{S}_1 = prmpt$: la préemption est autorisée (l'exécution d'une opération peut être interrompue puis reprise sur la même machine ou sur une autre).

- $\check{S}_2$: identifie les ressources additionnelles du système.

- $\check{S}_3$: décrit les contraintes de précédence.

- $\check{S}_4 \in \{\phi, r_i\}$: décrit les dates de disponibilité des différentes tâches.

- $\check{S}_5$: définit les particularités des durées opératoires s'il y en a.

c) Le paramètre • représente le critère d'optimisation sur lequel l'ordonnancement est évalué.

En plus de cette classification formelle, la littérature a l'habitude de classifier les problèmes d'ordonnancement selon l'organisation de l'atelier. C'est une classification simple basée seulement sur le premier champ de l'approche précédente. D'où, on distingue les problèmes de machine unique, les problèmes de machines parallèles, les problèmes de Show Shop, de Job Shop, d'Open Shop etc.

6. Organisations d'ateliers

La diversification des systèmes de production se caractérise par l'existence d'une multitude d'organisations au niveau "*ateliers*". Ces organisations déterminent théoriquement et pratiquement des problèmes d'ordonnancement différents.

6.1. Les différents types d'organisations

La classification des systèmes de production peut se faire selon plusieurs critères. La classification suivante [Hen, 99] [T'ki & Bil, 06] qui prend en considération la disposition des ressources (machines), distingue trois types d'organisations :

- Les organisations à ressource unique ;
- Les organisations à ressources multiples ;
- Les organisations multi-étages parallèles.

6.1.1. Les organisations à ressource unique

Ce type d'organisations est un cas que l'on peut rencontrer rarement dans les entreprises industrielles. Il constitue alors, le problème classique de machine unique. Par contre, il constitue un domaine d'étude considérable dans certains systèmes : par exemple dans les systèmes informatiques partagés où l'on dénote souvent la présence d'une seule ressource : une seule imprimante, un seul microprocesseur,... avec plusieurs usagers.

6.1.2. Les organisations à ressources multiples

Ces organisations se caractérisent par l'existence de plusieurs ressources, sur lesquelles seront exécutées plusieurs tâches. Ce sont des organisations énormément répandues dans le domaine de la production industrielle, constituant ainsi des catégories différentes d'ateliers. Trois types ont une importance particulière dans les problèmes d'ordonnancement : le Job Shop, le Flow Shop et l'Open Shop.

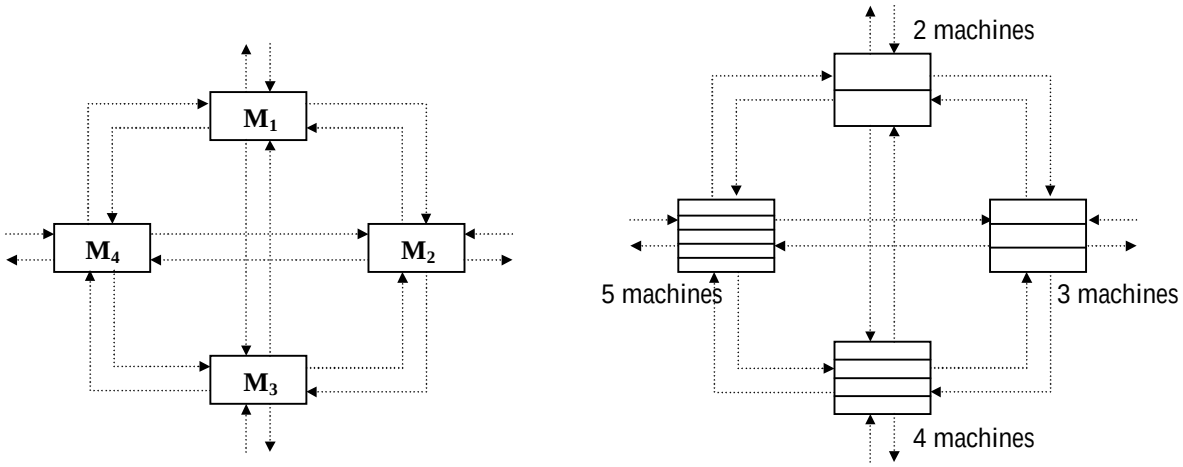
a. Atelier à cheminement multiple : Job Shop

Dans un atelier de type Job Shop les tâches ne s'exécutent pas sur toutes les machines dans le même ordre. En effet, chaque tâche emprunte un "chemin" qui lui est propre.

Ce type d'organisation correspond généralement à une production par lot, notamment dans une unité de production disposant de moyens polyvalents utilisés suivant des séquences différentes afin de réaliser des produits divers [Duv, 00].

Si, pour chaque machine, on dispose d'un et un seul exemplaire, on dit que l'organisation est un Job Shop simple.

Si, au contraire, pour un au moins des machines (postes de travail), on dispose de plus d'un exemplaire, on l'appelle Job Shop Hybride ;



(A) Job Shop simple à 4 machines. **(B)** Job Shop hybride à 4 postes de travail.

Figure 1.6 : Exemples de Job Shop simple et hybride.

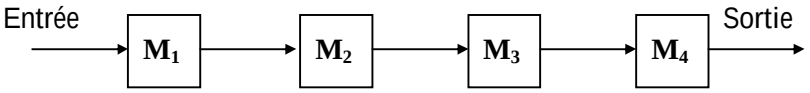
b. Atelier à cheminement unique : Flow Shop

C'est une organisation de production linéaire caractérisée par une séquence d'opérations identiques pour tous les produits. Une tâche est ainsi constituée d'opérations passant par différentes machines de manière linéaire, l'ordre de passage par les machines est le même pour toutes les tâches. Autrement dit, on dispose de m machines en série, tous les produits (tâches) doivent passer par chacune des machines : *i.e* toutes les tâches passent par *la machine* 1, puis *la machine* 2, etc.

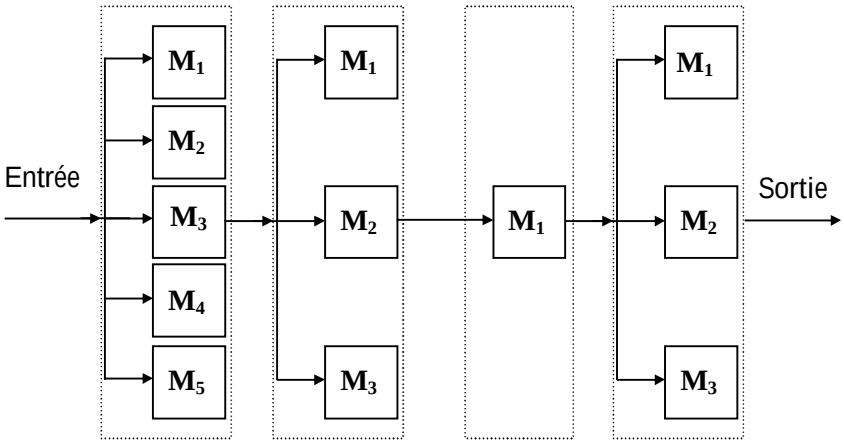
Ce type se rencontre fréquemment en pratique : montage, chaîne de traitement etc.

Le terme Flow Shop hybride est utilisé pour désigner des variantes de ce problème. C'est le cas rencontré dans plusieurs situations : Pour au moins un des postes de travail on dispose de plus d'un exemplaire de machine pour effectuer le même travail ou lorsque certaines machines

sont capables de réaliser simultanément plusieurs travaux ou encore lorsque les temps de rejet sont pris en compte [Duv, 00].



(A) Flow Shop simple à 4 machines.



(B) Flow Shop hybride à 4 postes de travail.

Figure 1.7 : Exemples de Flow Shop simple et hybride.

c. Atelier à cheminement libre : Open Shop

Dans les organisations de type Open Shop, les contraintes de précédence sont relâchées [Duv, 00]. Autrement dit, les opérations nécessaires à la réalisation de chaque tâche peuvent être effectuées dans n'importe quel ordre (les gammes sont libres). Ce cas se présente lorsque chaque produit à fabriquer doit subir une séquence d'opérations, mais dans un ordre totalement libre.

6.1.3. Les organisations multi-étages parallèles

Dans ces organisations, les machines sont regroupées en étages distincts. Chaque machine appartient à un seul étage. Toutes les machines du même étage sont capables d'effectuer la même opération. Plusieurs configurations de cette organisation peuvent être distinguées [T'ki & Bil, 06] :

- Les machines sont identiques (*P*) : une opération prend la même durée opératoire sur toutes les machines.

- Les machines sont uniformes (Q) : le temps opératoire d'une opération O_{ij} sur la machine M_k est égal à $p_{ijk} = q_{ij} / v_k$, où q_{ij} : est considéré comme le nombre d'unités constituant l'opération, et v_k le nombre de mêmes unités que peut la machine M_k traiter en unité de temps. Autrement dit, la capacité de chaque machine est une fonction linéaire des autres machines.
- Les machines sont différentes (R) : une opération prend des durées opératoires totalement indépendantes sur chaque machine.

Cette catégorie couvre en réalité une grande variété d'organisations où apparaît la notion de parallélisme. Ainsi, certaines classes des types vus précédemment appartiennent également à cette catégorie (Job Shop hybride, Flow Shop hybride,…).

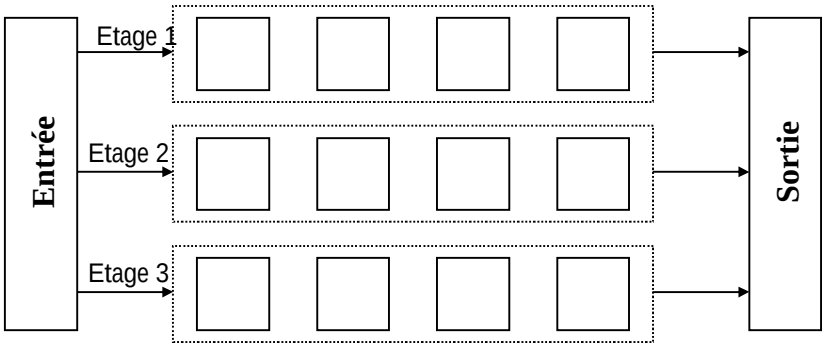


Figure 1.8 : Schéma général d'une organisation multi-étages parallèles.

6.2. Relations entre organisations

Les organisations présentées précédemment ne sont pas indépendantes. En effet, des relations de ressemblance, de réduction et de différentiation existent entre elles.

L'étude des relations qui existent entre les différents problèmes d'ordonnancement revêt un grand intérêt, dans la mesure où cela permet d'appliquer des algorithmes de résolution connus pour certaines classes de problèmes à d'autres classes qui leurs sont réductibles [T'ki & Bil, 06].

La figure 1.9 ci-dessous illustre ces différentes relations : chaque arc de A à B s'interprète que B est un cas particulier de A en considérant la condition associée. Par exemple, le Flow Shop simple est un cas particulier du Flow Shop hybride, où toutes les machines existent en un seul exemplaire. Il est également, un cas particulier du Job Shop simple avec gammes identiques pour toutes les tâches.

Dans certains cas, ce schéma peut être lu en terme de domaine de solutions réalisables. Par exemple, l'ensemble des solutions réalisables du problème de "machine unique" appartient également à l'ensemble des solutions réalisables du problème de Flow Shop simple [Hen, 99].

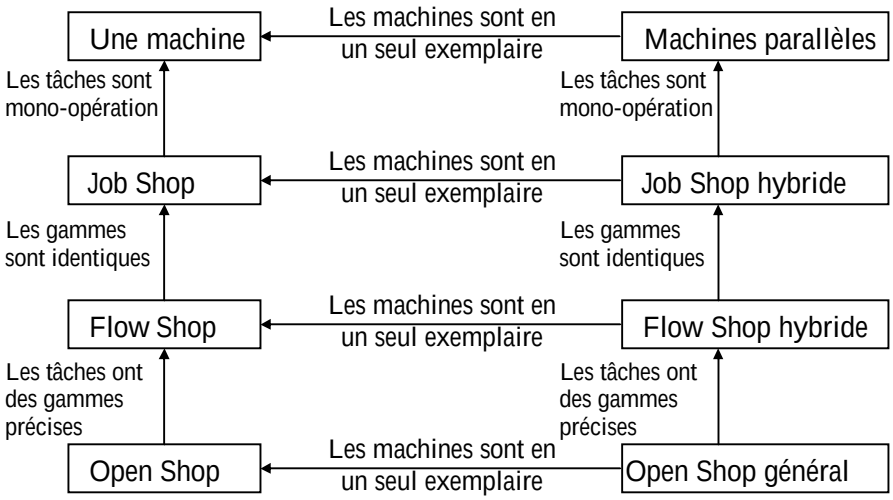


Figure 1.9 : Relations entre les différentes organisations [T'ki & Bil, 06].

7. Conclusion

Nous avons abordé dans ce chapitre l'aspect commun des problèmes d'ordonnancement en production, en exposant notamment les notions de base concernant ces problèmes. D'abord, le cadre général de l'ordonnancement comme fonction originale du système de gestion de production a été évoqué, en présentant succinctement les systèmes de production, la gestion de production et le rôle de l'ordonnancement au sein de ces systèmes.

Le deuxième point exposé dans ce chapitre, est la caractérisation du problème d'ordonnancement d'atelier en présentant sa définition, et précisant notamment les différents éléments qui le déterminent : les tâches, les ressources, les contraintes et les critères d'optimisation ; ainsi que les approches de classification qui s'appuient sur un ensemble de paramètres portant sur différentes caractéristiques du problème dont l'organisation d'ateliers (Job Shop, Flow Shop, Open Shop) décrites par la suite, constitue un champ important de la classification.

CHAPITRE II

Le problème d'ordonnancement de Job Shop

Résumé : *Le présent chapitre vise à circonscrire le problème d'ordonnancement du Job Shop, qui constitue l'un des problèmes d'ordonnancement les plus étudiés. Le chapitre commence par une présentation du problème en exposant sa formulation, sa modélisation et sa difficulté de résolution. Il présente par la suite, les méthodes envisagées pour sa résolution en s'intéressant surtout aux Métaheuristiques. Finalement, sont évoqués les fameux benchmarks du problème.*

CHAPITRE II

Le problème d'ordonnancement de Job Shop

1. Introduction

Le problème d'ordonnancement de Job Shop est un cas particulier du problème général d'ordonnancement. C'est l'un des problèmes de la théorie de l'ordonnancement et de l'optimisation combinatoire les plus traités [Pen, 94].

Dans ce chapitre, nous présentons les bases théoriques et pratiques importantes relatives au problème, en se concentrant sur sa spécificité en terme de contraintes, difficulté, modélisation et méthodes de résolution. Ainsi, la deuxième section est consacrée à la description du problème : les données, les contraintes et les critères d'évaluation. La troisième section présente les différentes classes d'ordonnancement. Les approches importantes de présentation du problème sont données à la quatrième section. Dans la cinquième section, nous discutons la complexité du problème. La sixième section est consacrée à une synthèse des méthodes de résolution connues en répertoriant les importantes méthodes envisagées pour la résolution du problème. Enfin, quelques instances-types, connues par "*benchmarks*", du problème sont présentées.

2. Présentation du problème de Job Shop

Il existe de nombreuses variations autour du problème de Job Shop. Nous donnons ici une formulation générale du problème de Job Shop simple, tout en précisant ensuite les restrictions qui caractérisent le cas du Job Shop traité dans ce mémoire.

Le problème d'ordonnancement de Job Shop consiste à réaliser un ensemble de n tâches sur un ensemble de m ressources (machines) en cherchant d'atteindre certains objectifs. Chaque tâche J_i est composée d'une suite de n_i opérations devant être exécutées sur les différentes ressources selon un ordre préalablement défini. Par ailleurs, un ensemble de contraintes concernant les ressources et les tâches doivent être respectées.

Donc, la détermination du problème de Job Shop $m \times n$, constitué de n-tâches et m-machines, se fait en précisant les données, les contraintes et les objectifs du problème.

2.1. Les données

Les données du problème sont [Fre, 82] [Vac, 00] :

- **Un ensemble M de m machines :** Une machine est notée M_k avec k = 1,..., m. Chaque machine ne peut effectuer qu'un seul type d'opérations. Le nombre total d'opérations exécutées par cette machine est noté m_k.
- **Un ensemble J de n tâches :** Une tâche est notée J_i avec i = 1,...,n. Chaque tâche est composée d'une gamme opératoire, *i.e.* une séquence linéaire fixée de n_i opérations. Cette séquence ne dépend que de la tâche, et peut varier d'une tâche à l'autre.

  Cet ensemble d'opérations d'une tâche $J_i \in J$, est défini par :

  $$O_i = \{O_{i,1} \bullet \ldots \bullet O_{i,k}\}, \text{ tel que : } O_i \subset O ;$$

  Où $\bullet$ est l'opérateur de précédence. Il définit un ordre total sur l'ensemble des opérations de la même gamme.
- L'opération $O_{i,j}$ est la $j^{ème}$ opération dans la gamme opératoire de J_i. Elle se caractérise par :

  -La machine sur laquelle elle s'exécute : M_k. Le fait que la machine demandée par l'opération $O_{i,j}$ soit M_k s'écrit : $R(O_{i,j}) = M_k$.

  -Le temps opératoire $p_{i,j}$ qui correspond à la durée de $O_{i,j}$ sur M_k.

  Le nombre total des opérations dans l'atelier est noté : $n_o = \sum_{i=1}^{n} n_i$.

2.2. Les contraintes

La définition du cas général de Job Shop se limite aux données décrites précédemment et n'impose pas de contraintes supplémentaires. Toutefois, les études menées sur ce sujet ajoutent des contraintes diverses afin de formuler des cas particuliers. Généralement, ces contraintes touchent à la fois les possibilités d'utilisation des machines et les liens qui peuvent exister entre les opérations.

En outre, les contraintes diffèrent d'une formulation à l'autre (selon le type du Job Shop). Mais, nous ne retiendrons ici que le cas de Job Shop simple, dont les contraintes sont les suivantes :

- Les machines sont indépendantes les unes des autres (pas d'utilisation d'outil commun, par exemple).
- Les tâches sont indépendantes les unes des autres. En particulier, il n'existe aucun ordre de priorité attaché aux tâches.
- Une tâche ne peut être en état d'exécution que sur une seule machine à la fois. Deux opérations de la même tâche ne peuvent être exécutées simultanément.
- Une machine ne peut exécuter qu'une seule opération à un instant donné.
- Seulement le temps d'exécution proprement dit, est pris en compte. Les temps de transport d'une machine à l'autre, de préparation, etc. ne sont pas considérés.
- Les machines sont disponibles jusqu'à la fin de l'ordonnancement. En particulier, les pannes de machines ne sont pas prises en compte.
- Une opération en cours d'exécution ne peut pas être interrompue (pas de préemption).
- Les tâches sont autorisées d'attendre les ressources autant qu'il faut. Il n'y a pas de dates d'échéance.
- La définition des tâches est déterminée avant le lancement de l'ordonnancement, *i.e.* il n'existe pas d'évènements aléatoires pendant l'exécution.

Il est fréquent de trouver une formulation plus restrictive du problème de Job Shop. Dans celle-ci, chaque tâche passe sur les m machines de l'atelier une fois et une seule. Toutes les tâches sont donc constituées de m opérations, et chaque machine doit effectuer n opérations (n : nombre de tâches). Le nombre total d'opérations est alors, $n_o = n \times m$. Si $n = m$, le problème est dit carré.

Les benchmarks (voir la section 7 de ce chapitre), qui constituent l'échantillon de test pour notre travail, vérifient de leur part ces restrictions.

Le problème de Job Shop sous cette forme est dit *simple* : chaque tâche est constituée d'un seul plan, et chaque opération ne peut être effectuée que sur une seule machine. Le Job Shop est dit généralisé (ou étendu), si les tâches sont constituées d'un ou de plusieurs plans (peuvent être répétitives) et les machines peuvent exister en un ou plusieurs exemplaires.

2.3. Les objectifs

L'objectif du problème d'ordonnancement est de fixer les dates de début des opérations. Pour cela, il faut déterminer l'ordre de passage de l'ensemble des tâches sur chaque

machine, en respectant les contraintes du problème. Le but est ensuite, de minimiser ou maximiser une *fonction objectif*, pour trouver la ou les meilleure(s) solution(s). Cette fonction objectif s'appelle aussi dans ce contexte : critère de performance, critère d'évaluation ou encore objectif de l'ordonnancement [Duv & al., 98].

Le résultat d'un ordonnancement, généralement représenté sur un diagramme de Gantt, est fonction des dates de début calculées des opérations [Pen, 94].

Nous avons déjà vu à la section 4 du premier chapitre, un nombre important de critères de performance concernant l'ordonnancement des ateliers en général, et qui s'appliquent, pour la plupart, au cas de Job Shop. Ces critères guident la résolution des problèmes d'ordonnancement.

Nous ne rappelons ici que quelques critères les plus importants :

- C_{max} : le *makespan*, est la durée totale de l'ordonnancement.
- $\overline{C}$: est la moyenne des temps d'achèvement des tâches.
- $\Sigma w_i Ci$: est la somme des dates de fin pondérées.
- T_{max}, E_{max} : sont respectivement, le retard maximum des tâches et le retard maximum des tâches par rapport à une date d'achèvement prévue.
- $\overline{T}$: est la moyenne des retards de l'ensemble des tâches.

Dans notre application, et pour tous les problèmes traités, nous prendrons comme objectif, la durée totale de l'ordonnancement C_{max}, puisque d'un côté, c'est un critère simple et le plus utilisé dans la littérature ; d'un autre côté, notre travail n'exige pas des critères trop compliqués, un critère simple suffit à la réalisation de l'étude.

3. Classes d'ordonnancement

Pour générer des solutions au problème d'ordonnancement de Job Shop, il est impératif d'exploiter le temps au maximum. En effet, certaines solutions comprennent "des trous" de temps, que l'on puisse exploiter par simples décalages des opérations retardées afin d'obtenir un ordonnancement plus "compact". Plus un ordonnancement est "compact", meilleure est sa qualité [Duv, 00].

Cette notion de *compacité* qui est un objectif majeur de tout ordonnancement, est à la base de distinction de plusieurs classes d'ordonnancement : admissible, actif, semi-actif, sans délai [Pin, 95] [T'ki & Bil, 06].

3.1. Ordonnancement admissible (acceptable)

Un ordonnancement est dit admissible ou acceptable, s'il respecte toutes les contraintes du problème. Par exemple, soit le problème de Job Shop 3-tâches 3-machines, défini par :

$$J_1[M_1{:}4\ ;\ M_2{:}3\ ;\ M_3{:}3]\ ;\ J_2[M_1{:}1\ ;\ M_3{:}5\ ;\ M_2{:}3]\ ;\ J_3[M_2{:}2\ ;\ M_1{:}4\ ;\ M_3{:}1]\ ;$$

L'ordonnancement présenté sur la figure 2.1, bien qu'il ne soit pas optimal, est admissible (puisqu'il respecte toutes les contraintes).

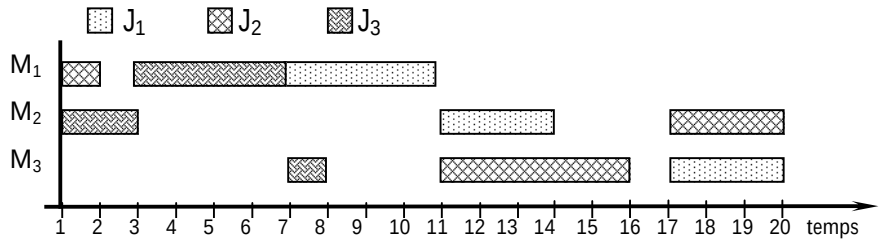


Figure 2.1 : Exemple d'un ordonnancement admissible.

Dans certains cas, des décalages à gauche sur certaines opérations sont nécessaires. Selon que l'ordre des opérations reste inchangé ou non, on distingue deux cas [Pin, 95] :

- On parle de décalage à gauche "*local*", lorsqu'on avance le début d'une opération sans remettre en cause l'ordre relatif entre les autres opérations ;
- On parle de décalage à gauche "*global*", lorsqu'on avance le début d'une opération en modifiant l'ordre relatif entre au moins deux opérations.

3.2. Ordonnancement semi-actif

Un ordonnancement est dit semi-actif, si aucun décalage à gauche local n'est possible. On ne peut exécuter aucune opération plus tôt, sans altérer l'ordre relatif au moins de deux opérations. Toutes les opérations sont calées, soit sur l'opération qui précède dans la gamme, soit sur l'opération qui précède sur la machine utilisée. A titre d'exemple, l'ordonnancement présenté sur la figure 2.2 est semi-actif, aucun décalage local n'est possible.

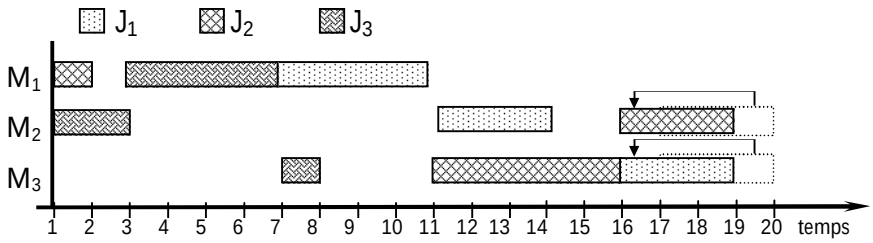


Figure 2.2 : Décalage à gauche local de $O_{1,3}$ et $O_{2,2}$ dans l'ordonnancement de la figure 2.1, conduit à un ordonnancement semi-actif.

3.3. Ordonnancement actif

Un ordonnancement est dit actif, si aucun décalage à gauche local ou global n'est possible. En conséquence, dans un ordonnancement actif, il est impossible d'avancer une opération, sans reporter le début d'une autre opération [Pin, 95].

L'ordonnancement présenté sur la figure 2.3 est actif puisque aucun décalage à gauche, local ou global ne peut être effectué.

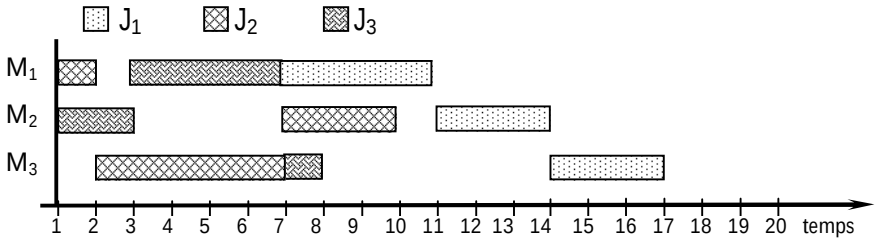


Figure 2.3 : Exemple d'ordonnancement actif.

3.4. Ordonnancement sans délai

Un ordonnancement est dit sans délai ou sans retard, si et seulement si aucune opération n'est mise en attente alors qu'une machine est disponible pour l'exécuter [T'ki & Bil, 06].

Ainsi, l'ordonnancement illustré sur la figure.2.3, bien qu'il soit actif, il n'appartient pas à cette classe (*i.e.* il est avec retard). En effet, l'opération $O_{1,1}$ qui attend la machine M_1 est mise en attente, alors que M_1 est oisive à t = 1. On peut le transformer sans retard comme cela est illustré sur la figure 2.4 :

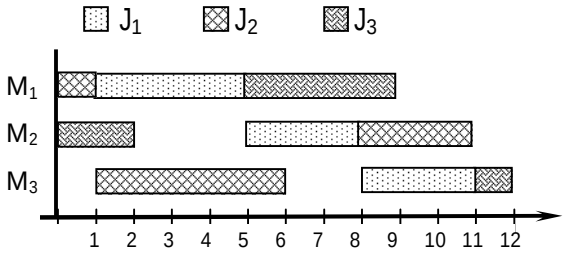


Figure 2.4 : Ordonnancement sans délai.

A noter que la transformation à un ordonnancement sans délai peut mener à une solution plus mauvaise du point de vue de makespan [Lop & Esq, 99].

La figure 2.5 représente les relations d'inclusion des classes d'ordonnancements vues précédemment. Ce schéma fait apparaître que les ordonnancements sans retard sont inclus dans le sous-ensemble des ordonnancements actifs, qui sont eux-mêmes inclus dans le sous-ensemble des ordonnancements semi-actifs. Les ordonnancements admissibles comprennent tous ceux qui vérifient les contraintes du problème.

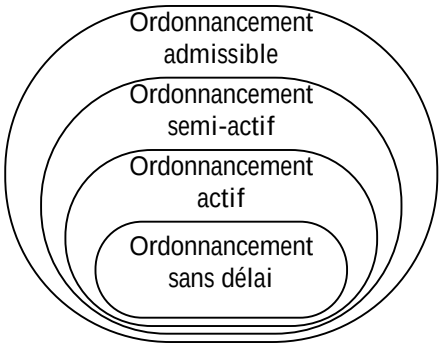


Figure 2.5 : Relations d'inclusion entre les différentes classes d'ordonnancements [Pin, 95].

4. Modélisation graphique du problème de Job Shop

Traiter un problème d'une telle complexité que celle du Job Shop nécessite une modélisation claire et précise. Il existe diverses manières de modéliser ce problème, et qui conduisent à choisir un mode de description adéquat.

Si la modélisation par graphe disjonctif est la plus classique et la plus répandue, les modélisations en programmation mathématique sont les plus anciennes [Dan & al., 03]. Il existe également, des modélisations algébriques, polyédrales, par les graphes potentiels-tâches, par les réseaux de Pétri, des modélisations basées sur UML etc.

4.1. Modélisation par graphe disjonctif

La modélisation sous forme d'un graphe disjonctif est très utilisée pour représenter les problèmes d'ordonnancement et en particulier les problèmes de Job Shop. Présentée pour la première fois par B. Roy et B. Sussmann en 1964, cette formulation a été à l'origine des premières méthodes de résolution du problème [Pen, 94] [Jai, 98].

Un problème de Job Shop peut alors se modéliser par un graphe disjonctif G (X , $C \cup D$) où :

- X : est l'ensemble des *sommets* : chaque opération correspond à un sommet, avec deux opérations fictives S (source) et P (puits) désignant le début et la fin de l'ordonnancement.
- C : est l'ensemble des arcs *conjonctifs* représentant les contraintes d'enchaînement des opérations d'une même tâche (gammes opératoires). Pour un arc (ij,ik) de la partie conjonctive, on a :

$$t_{ij} - t_{ik} \geq p_{i,k} \quad \forall\ (ij,ik) \in C$$

On aura donc un arc entre tous les sommets (i,j) , (i,j+1) pour $i = 1 \ldots n$ et $j = 1 \ldots n_i - 1$; (n : nombre de tâches, n_i : nombre d'opérations de la tâche i). De plus, il existe des arcs entre S et tous les sommets (i,1), et d'autres arcs entre les sommets (i, n_i) et le puits P.

- D : l'ensemble des arcs *disjonctifs* associés aux conflits d'utilisation d'une machine. Pour un arc (ij,ik) de la partie disjonctive, on a [Lop & Esq, 99] :

$$t_{jk} - t_{ik} \geq p_{ik} \text{ ou } t_{ik} - t_{jk} \geq p_{jk} \quad \forall (ik,jk) \in D$$

Que l'on modélise par un arc et un arc retour entre les sommets (t_{jk}, t_{ik}) représentant deux opérations utilisant la même machine.

L'ensemble des paires de disjonction associées à une même machine forment une *clique de disjonction*.

Le graphe disjonctif sous cette forme, modélise un problème d'ordonnancement. Pour construire un ordonnancement admissible sur ce graphe, il suffit de choisir pour chaque paire d'arcs, l'arc qui déterminera l'ordre de passage des deux opérations sur la machine ; c'est à dire *arbitrer* chacune des disjonctions. L'arbitrage doit être complet (toutes les disjonctions sont arbitrées), et compatible (le graphe est sans circuits) [Pen, 94] [Hur & Knu, 05].

La figure 2.6 correspond au graphe disjonctif non arbitré du problème précédent (§ 3.1). Le graphe arbitré présentant l'ordonnancement de la figure 2.3 est illustré sur la figure 2.7.

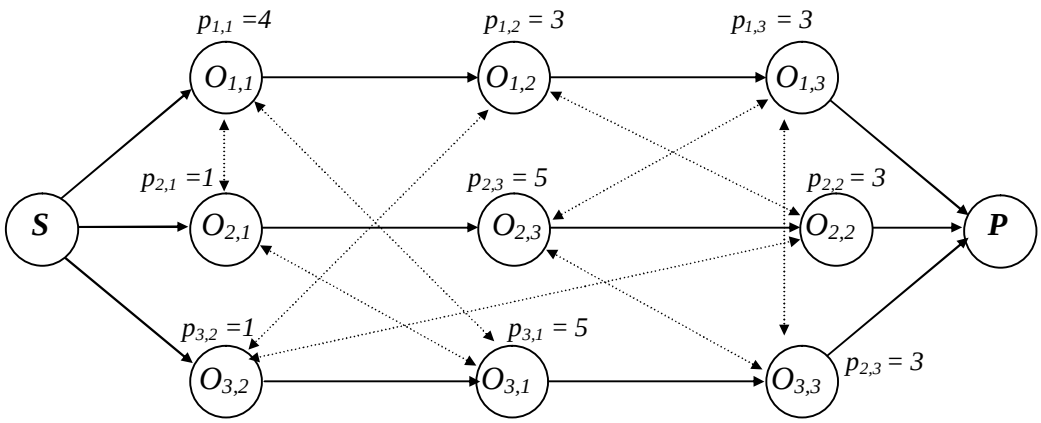


Figure 2.6 : Le graphe disjonctif du problème de Job Shop du § 3.1.

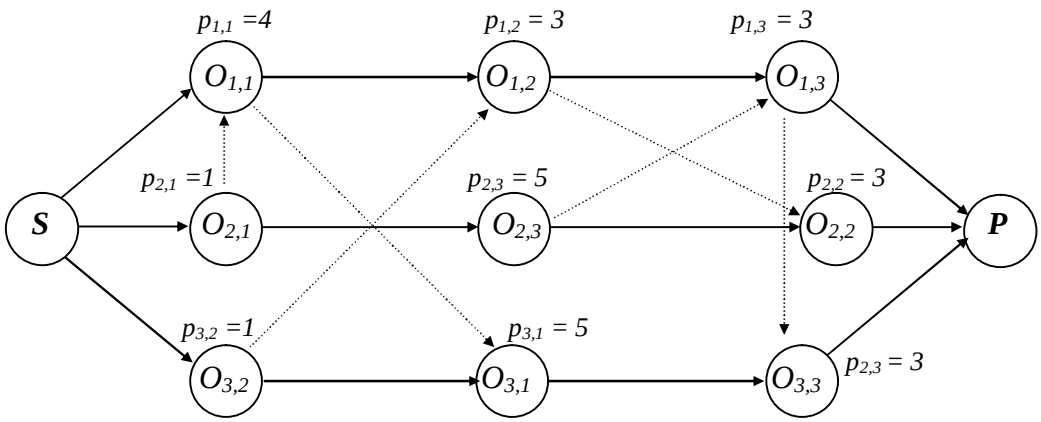


Figure 2.7 : Le graphe disjonctif arbitré simplifié de l'ordonnancement de la figure 2.3.

Pour un ordonnancement donné, certaines notions bien évidentes sur le graphe disjonctif sont intéressantes à évoquer :

- **L'opération critique :**

Soit un ordonnancement actif, une opération est dite *critique*, si elle provoque nécessairement l'augmentation du makespan de l'ordonnancement, lorsqu'elle est retardée (alors que l'ordre d'exécution des opérations défini par l'ordonnancement ne change pas) [Duv, 00].

- **Le chemin critique :**

Le chemin critique est une suite d'opérations critiques liées par des relations de précédence. La longueur d'un chemin critique est égale à la somme des durées des opérations qui le composent.

Pour un ordonnancement donné, il peut exister plus d'un chemin critique.

- **Le bloc critique :**

Le bloc critique est une succession d'opérations critiques s'exécutant sur la même machine [Hur & Knu, 05]. Tout chemin critique peut se décomposer en b blocs critiques. Pour un problème n-tâches m-machines, b est compris entre 1 et n : $1 \leq b \leq n$.

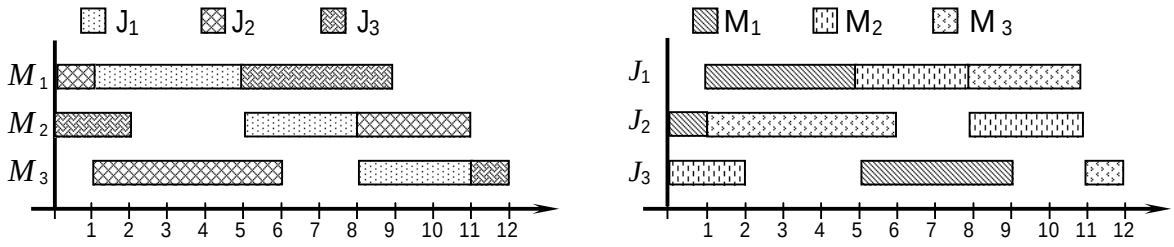
4.2. Représentation de l'ordonnancement par le diagramme de Gantt

La représentation la plus courante pour l'ordonnancement est le *diagramme de Gantt*. Celui-ci représente une opération par un segment ou une barre horizontale, dont la longueur est proportionnelle à sa durée opératoire.

Donc, sur ce diagramme sont indiqués, selon une échelle temporelle : l'occupation des machines par les différentes tâches, les temps morts et les éventuelles indisponibilités des machines dues aux changements entre produits [Jef & al., 06].

Deux types de diagramme de Gantt sont utilisés : Gantt ressources et Gantt tâches (voir la figure 2.8) [Lop & Esq, 99] :

- Le diagramme de Gantt ressources, est composé d'une ligne horizontale pour chaque ressource (machine). Sur cette ligne, sont visualisées les périodes d'exécution des différentes opérations en séquence et les périodes de l'oisiveté de la ressource.
- Le diagramme de Gantt tâches, permet de visualiser les séquences des opérations des tâches, en représentant chaque tâche par une ligne sur laquelle sont visibles, les périodes d'exécution des opération et les périodes où la tâche est en attente des ressources.



(A) Gantt ressources. **(B)** Gantt tâches.

Figure 2.8 : Diagrammes de Gantt (problème de Job Shop du § 3.1).

5. Complexité du problème de Job Shop

La théorie de complexité permet de classer les problèmes en deux classes P et NP. La classe P regroupe les problèmes résolubles par des algorithmes polynomiaux. Un algorithme est dit polynomial, lorsque son temps d'exécution est borné par $O(p(x))$ où p est un polynôme et x est la longueur d'entrée d'une instance du problème [Fre, 82]. Les algorithmes dont la complexité ne peut pas être bornée polynomialement sont qualifiés d'exponentiels et correspondent à la classe NP [Fre, 82]. Les problèmes NP-complets ou NP-difficiles, représentent une large classe parmi la classe NP.

5.1. L'importance de l'espace de solutions

Le problème de Job Shop est classé par la théorie de complexité parmi les problèmes NP-difficiles au sens fort [Bri & Bru, 01]. Afin de donner une idée sur l'importance de l'espace de solutions pour une instance du problème, rappelons-nous qu'il existe $(n!)^m$ différentes solutions pour un problème de n tâches à réaliser sur m machines. Ainsi, pour un problème de taille de 10×10, il existe 39594×10^{65} solutions différentes (par comparaison, l'âge de la Terre ne dépasse pas 10^{18} secondes). Sachant que ce nombre ne comptabilise pas les ordonnancements semi-actifs. Enumérer toutes ces possibilités pour arriver à la solution optimale n'est pas une chose réaliste.

Il est à noter que ce nombre évolue plus vite en fonction de J qu'en fonction de M. Autrement dit, l'ajout d'une tâche a beaucoup plus d'impact que l'ajout d'une machine. Par exemple :

- Pour un problème de 4 tâches et 5 machines (4×5) : $(n!)^m = 7962624$;
- Alors que, pour un problème de 5 tâches et 4 machines (5×4) : $(n!)^m = 207360000$.

La difficulté du Job Shop est fonction du nombre de tâches, du nombre de machines, du nombre d'opérations par tâche, et de la durée des opérations.

Le tableau 2.1, suivant [Duv, 00], résume les cas où le problème peut être résolu en temps polynomial et les cas où il devient NP.

<table>
<tr><th>P</th><th colspan="2">NP</th></tr>
<tr><td>$m = 2, \forall j \ n_j \leq 2$</td><td>$m = 2, \forall j \ n_j \leq 3$ [$\exists k, n_k = 3$]</td><td rowspan="2">$m = 2$
autre cas</td></tr>
<tr><td>$m = 2, \forall o$ d(o) = 1</td><td>$m = 2, \forall o \ d(o) \leq 2$ [$\exists k, d(k) = 3$]</td></tr>
<tr><td rowspan="2"></td><td>$m = 3, \forall j \ n_j \leq 2$</td><td rowspan="2">$m \geq 2$</td></tr>
<tr><td>$m = 3, \forall o \ d(o) = 1$</td></tr>
<tr><td>$n = 2$</td><td colspan="2">$n \geq 3$</td></tr>
<tr><td>$C_{max} \leq 3$</td><td colspan="2">$C_{max} \geq 3$</td></tr>
</table>

Tableau 2.1 : Complexité du Job Shop. [Duv, 00].

n : nombre de tâches, m: nombre de machine, n_j : nombre d'opération par tâche, $d(o)$: durée de l'opération.

5.2. Difficulté relative des instances

La taille des instances que l'on traite est très importante dans la détermination de leur difficulté. Une classification des problèmes en fonction du : nombre de job, nombre d'opérations par job et nombre de machines est proposé par B. Penz [Pen, 94].

Cette classification, même moins rigoureuse [Pen, 94], permettra de repérer facilement la taille d'un problème donné, en déterminant trois classes :

- Problèmes de petite taille : Problèmes de moins de 20 tâches, moins de 10 machines et moins de 10 opérations par tâche.
- Problèmes de taille moyenne : Problèmes entre 20 et 50 tâches, de 5 à 15 machines et de 5 à 15 opérations par tâche.
- Problèmes de grande taille : Problèmes de plus de 50 tâches, de plus de 5 machines et de plus 5 opérations par machine.

La taille n'est pas le seul déterminant de la difficulté, en effet d'autres paramètres influent sur cette difficulté des problèmes. Ainsi, il est démontré que le rapport $\frac{J}{M}$ détermine bien la difficulté de l'instance. Ce rapport interprète la conjecture de Taillard, dont l'énoncé est le suivant [Duv, 00] :

> « Les instances telle que $n \approx m$ sont plus difficiles à résoudre que celles pour lesquelles $n >> m$; et la frontière entre les instances *faciles* et les instances *difficiles* se situe aux environs de $n = 5m$ ».

6. Méthodes de résolution du problème de Job Shop

Nous avons vu précédemment que le problème d'ordonnancement de Job Shop à l'exception de certains cas, est NP-difficile. Les problèmes appartenant à la classe P ont des algorithmes polynomiaux permettant de les résoudre. Pour les problèmes appartenant à la classe NP, l'existence d'algorithmes polynomiaux semble peu réaliste. Différentes méthodes de résolution (exactes ou heuristiques), sont largement utilisées pour appréhender les problèmes de Job Shop NP-difficiles.

6.1. Les différentes méthodes de résolution

Une présentation de la totalité des méthodes et de leurs variantes apparaît une tâche difficile au vu du nombre d'approches proposées. Nous présentons sur le tableau 2.1 les grandes orientations des méthodes, en essayant de repérer les travaux de recherche essentiels. Ce tableau est synthétisé à partir des travaux de [Duv, 00] [Jai, 98] [Jai & Mee, 99(2)] [Pen, 94]. Pour les Métaheuristiques, nous essayons de circonscrire avec plus de détails - sans prétendre toutefois être exhaustif - l'important des travaux effectués sur ces méthodes dans un paragraphe à part.

Méthodes de résolution			**Principaux auteurs**
Exactes	Méthodes pour les problèmes polynomiaux	Le problème à deux machines	Johnson (54), Akers (56), Jackson (56), Szwarc (60).
		Le problème à deux tâches	Akers (56), Brucker (88, 94), Brucker & Jurisch (93) Kravchenko & Sotskov (96), Kubiak & Timkovsky (96), Timkovsky (97), Williamson & al. (97), Brucker & al. (97).
	La programmation mathématique		Bowman (59), Wagner (59), Manne (60), Balas (65), Dyer & Wolsey (90), Van Den Akker (94).
	Le "branch and bound"		Cook (71), Lageweg & al. (77) Pinson (88, 95).
Approximatives	Les méthodes de relaxation		Fisher (73, 76) Fisher & al. (75), Van De Velde (91), Hoitomt & al. (93), Della Croce & al. (93) Hoogeveen & Van De Velde (95).
	Les méthodes de décomposition		Ashour (67) Kanzedal (83), Chu et al. (92) Krüger et al. (95).
	Les heuristiques constructives	L'approche par opérations	Rowe & Jackson (56), Giffler & Thompson (60), Fisher & Thompson (63), Crowston et al. (63), Jeremiah & al. (64), Gere (66), Moore (68) Panwalkar & Iskander (77), Blackstone & al. (82) Viviers (83), Lawrence (84), Haupt (89), Chang & al. (96), Sabuncuoglu & Bayiz (97).

Approximatives (suite)	Les heuristiques constructives (suite)	L'approche par machines (Shifting Bottleneck)	Adams & al. (88), Applegate & Cook (91), Dauzère-Pérès & Lasserre (93), Balas & al. (95), Dauzère-Pérès (95), Holtsclaw & Uzsoy (96), Demirkol & al. (97), Balas & Vazacopoulos (98).
		L'approche par tâches	Penz (94),
	Les Métaheuristiques	Algorithmes Génétiques	Davis (85), Falkenauer & Bouffouix (91), Nakano & Yamada (91, 92), Tamaki & Nishikawa (92), Davidor & al. (93), Fang & al. (93), Mattfeld & al. (94), Della Croce & al. (95), Kobayashi & al. (95), Norman and Bean (97), Bierwirth (95), Bierwirth & al. (96), Cheng & al. (96), Shi (97).
		Recherche Locale Génétique	Aarts & al. (91, 94), Pesch (93), Della Croce & al. (94), Dorndorf & Pesch (95), Mattfeld (96), Yamada & Nakano (95, 96).
		Recuit Simulé	Matsuo & al. (88), Van Laarhooven & al. (88, 92), Aarts & al. (91, 94), Yamada & al. (94), Sadeh & Nakakuki (96), Yamada & Nakano (95, 96), Sadeh & al. (97), Kolonko (98), Aarts & al. (91, 94), Storer & al. (92), Aarts & al. (91, 94).
		Recherche Tabou	Taillard (89, 94), DellAmico & Trubian (93), Hara (95), Barnes & Chambers (95), Sun & al. (95), Nowicki & Smutnicki (96), Ten Eikelder & al. (97), Thomsen (97), Jain & Meeran (98, 98), Jain & al. (98).
	Autres méthodes	Satisfaction de Contraintes	Erschler & al. (76), Fox & Sycara (90) Fox & Sadeh (90), Caseau & Laburthe (94,95), Nuijten & Aarts (96), Harvey & Ginsberg (95), Sadeh & al. (95), Baptiste & Le Pape (95), Baptiste & al. (95), Pesch & Tetzlaff (96), Sadeh & Fox (96).
		Réseaux de neurones	Foo & Takefuji (88), Zhou & al. (90, 91), Van Hulle (91), Hanada & Ohnishi (93), Chang & Nam (93), Lo & Bavarian (93), Gang & Shuchun (94) Willems & Rooda (94), Satake & al. (94), Foo & al. (94, 95), Sabuncuoglu & Gurgun (96), Dagli & al. (91), Watanabe & al. (93), Cedimoglu (93), Sim & al. (94), Kim & al. (95).
		Systèmes experts	Alexander (87), Kusiak & Chen (88), Biegel & Wink (89), Charalambous & Hindi (91), Shakhlevich & al. (96), Sotskov (96).

Tableau 2.2 : Principales méthodes de résolution du problème de Job Shop.

6.2. Les Métaheuristiques

Dès l'avènement des Métaheuristiques comme approche de résolution importante des problèmes difficiles offrant un bon compromis entre le coût de résolution et la qualité des solutions, le problème d'ordonnancement de Job Shop constitue l'un des domaines d'application de ces méthodes intensivement investigué. Bien qu'il existe une multitude de Métaheuristiques, trois ont en trouvé une place importante : Les Algorithmes Génétiques, Le Recuit Simulé et la Recherche Tabou.

6.2.1. Les Algorithmes Génétiques

Les Algorithmes Génétiques AG, sont largement appliqués aux problèmes d'ordonnancement de Job Shop. Leur application requiert les adapter au problème, en définissant adéquatement la représentation et les opérateurs.

Plusieurs approches sont proposées pour représenter le problème de Job Shop avec cette méthode. Cheng & al. (1996) [Jai & Mee, 99(2)] en recensent neuf qui sont :

- *Operation based representation* : représentation basée sur l'opération,
- *Job based representation* : représentation basée sur la tâche,
- *Job pair relation based representation* : représentation basée sur relations de paire de tâches,
- *Completion time based representation* : représentation basée sur le temps d'achèvement,
- *Random keys representation* : représentation basée sur des clés aléatoires,
- *Preference list based representation* : représentation basée sur des listes de préférences,
- *Priority rule based representation* : représentation basée sur les règles de priorité,
- *Disjunctive graph based representation* : représentation basée sur le graphe disjonctif,
- *Machine based representation* : représentation basée sur la machine.

Les cinq premières approches sont des représentations directes du problème, tandis que les autres sont indirectes, leur implantation nécessite la mise en place de décodeur.

Parmi les premiers travaux exploitant les Algorithmes Génétiques pour résoudre ce problème, figure celui de Davis (1985) [Bie, 95] [Sha & Yao, 04], qui emploie une technique de codage basée sur des listes d'opérations en ordre préféré pour chaque machine. Cette stratégie fut étendue par Falkenauer & Bouffouix [Fal & Bou, 91], en employant des listes de préférence de tâches. Della Croce & al. [DCr & al., 95] ont adopté aussi cette stratégie. L'un des travaux les plus important se basant sur la même représentation est réalisé par Kobayashi & al. [Kob & al., 95] où un chromosome est une chaîne de symboles de longueur n et chaque symbole identifie une opération qui s'exécutera sur une machine donnée. Ce schéma exige un croisement

spécifique comme SXX développé par Kobayashi & al. [Kob & al., 95], ou LOX de Falkenauer & al. [Fal & Bou, 91], ou encore JOX développé par Yamamura & al. [Yam & al., 96].

Yamada & Nakano (1991), utilisent un codage binaire basé sur les relations de précédence entre les opérations s'exécutant sur la même machine. Le codage dans ce cas est connu par "codage basé sur le temps d'achèvement des opérations" [Yam & Nak, 97].

Tamaki & Nishikawa (1992) utilisent une représentation indirecte basée sur le graphe disjonctif. Un chromosome consiste en une chaîne binaire correspondant à une liste ordonnée de préférences de nœuds relatives au graphe disjonctif [Jai & Mee, 99(2)].

Un autre codage proposé par Holsapple & al. [Hol & al., 93] repose sur des listes de tâches : l'ordonnancement porte sur toutes les opérations d'une tâche donnée avant de passer à la suivante.

Dans la représentation basée sur les règles de priorité, utilisée par Dorndorf & al. [Dor & Pes, 95], le chromosome est une suite de gènes désignant chacun une règle de priorité. Les auteurs utilisent un constructeur d'ordonnancement basé sur l'algorithme de Giffle & Thompson. Ces règles sont très nombreuses, Sauer en décompte plus d'une centaine [Sha & Yao, 04].

Bierwirth [Bie, 95] crée un AG dont les chromosomes représentent des permutations de tâches. Dans un travail similaire, Bierwirth & al. [Bie & al., 96] analysent trois opérateurs de croisement, qui préservent respectivement l'ordre : relative, de position et absolue des permutations d'opérations. Un schéma proche est employé par Fang & al. [Fan & al., 93]. L'avantage de ce schéma est qu'il n'exige qu'un simple constructeur ; de plus, plusieurs croisements peuvent s'appliquer : GOX proposé par Oliver & al. [Sha & Yao, 04], PPX proposé par Bierwirth & al. [Bie & al., 96] [Mat & Bie, 98].

Norman & Bean (1997) proposent l'approche de clefs aléatoires, chaque gène (clé aléatoire) du chromosome consiste en deux parties : un entier de $\{1,2,...,m\}$ représentant la machine allouée, et une fraction générée aléatoirement de {0,1} dans un ordre croissant, détermine l'ordre des opérations sur chaque machine [Jai & Mee, 99(2)].

Shi (1997) a appliqué une technique de croisement qui divise arbitrairement le potentiel génétique en deux parties, à partir desquelles sont générés les descendants [Jai & Mee, 99(2)].

6.2.2. La Recherche Tabou

La Recherche Tabou (RT) issue des travaux de Glover, est une variante du paradigme de la recherche locale. La Recherche Tabou est une sorte de procédure de recherche déterministe,

orientée en s'aidant d'une mémoire qui garde l'historique des solutions, et qui interdit le retour à des solutions récemment visitées. La fonction de voisinage est la composante primordiale de la Recherche Tabou qui détermine amplement l'efficacité de recherche.

Une contribution remarquable en fonction de voisinage est provenue de Van Laarhooven & al. [Laa & al., 92], qui consiste à inverser l'ordre d'exécution sur la même machine de deux opérations critiques adjacentes. De plus, ils démontrent la propriété qu'il existe forcément une séquence de mouvements qui mènent à une solution optimale.

Laguna & al. (1991, 1993) présentent un des travaux précoces concernant l'application de cette méthode à l'ordonnancement des Job Shop. Ils ont construit trois stratégies de Recherche Tabou reposant sur des mouvements simples. Barnes & Laguna, suggèrent six composantes qui assurent la fiabilité de la méthode aux problèmes d'ordonnancement [Jai & Mee, 99(1)]. Dans leur méthode de Recherche Tabou, Barnes & Chambers (1995) utilisent la liste des meilleures solutions trouvées comme point de redémarrage de la recherche [Jai & Mee, 99(2)].

Dans sa notable contribution, Taillard [Tai, 94] incorpore une technique qui accélère la recherche en ne mettant à jour que les dates de début de certaines opérations. Cette technique ne permettant pas le calcul exact du makespan des mouvements est considérée comme une méthode d'estimation rapide mais relative.

Dell'Amico & Trubian [Del & Tru, 93] incorporent à leur algorithme de RT un nouveau générateur de solution initiale. De plus, deux voisinages sont formulés : le premier considère le reversement de jusqu'à trois arcs impliquant : i, $MP[i]$ et $MP[MP[i]]$ (resp. i, $MS[i]$ et $MS[MS[i]]$) (où $MP[i]$ et $MS[i]$ dénote resp. l'opération qui précède (suit) i sur la même machine). Si le reversement de l'un de ces arcs est tabou, alors le mouvement entier est interdit. Le deuxième voisinage repose sur le block critique. En vue d'accélérer la recherche, les auteurs emploient l'estimation précédente de Taillard.

Les meilleures résultats fournis par une simple Recherche Tabou sont obtenus par Nowicki & Smutnicki (1996), qui ont appliqué un voisinage plus restreint visant à inverser les deux premières ou les deux dernières opérations de chaque bloc critique non dernier (resp. non premier) [Jai & al., 00] [Wat & al., 06].

Ten Eikelder & al. [Eik & al., 97] ont construit des méthodes comportant plusieurs procédures de RT séquentielles et parallèles exploitant la fonction de voisinage de Nowicki & Smutnicki.

Thomsen 1997 a pu améliorer la résolution de plusieurs benchmarks difficiles par l'hybridation de la RT et la méthode par séparation et évaluation. Citons aussi parmi plusieurs autres contributions à la résolution du problème par la Recherche Tabou : Hurink & al.(1994), Brucker & Kr•mer (1995), Dauzère-Pérès & Paulli (1997), Brucker & Neyer (1998), Baar & al. (1997) [Jai & Mee, 99(2)].

6.2.3. Le Recuit Simulé

Le Recuit Simulé RS est une autre Métaheuristique basée sur la recherche locale. Il est proposé par Kirkpatrick & al. (1983) et Cerny (1985), mais ces origines remontent aux travaux de Metropolis & al. (1953). Son application au problème d'ordonnancement de Job Shop marque plusieurs travaux :

En se basant sur leur première fonction de voisinage, Van Laarhooven & al. (1992) [Laa & al., 92] construisent un Recuit Simulé générique ne demandant pas une vue profonde sur la structure du problème.

Un autre voisinage important est défini par Matsuo & al. (1988), qui ont prouvé que le reversement de deux opérations critiques ne conduit plus à une amélioration immédiate du makespan, si l'opération qui les précède et celle qui les suit sont également critiques, ce voisinage plus restreint, est incorporé à un Recuit Simulé et utilisé par ces auteurs [Sch, 01].

Le majeur inconvénient rencontré lors de l'application de la méthode de Recuit Simulé, est le temps d'exécution : à titre d'exemple certaines instances de *swv* (voir § 2.7) exigent plus de 375 millions itérations, qui se déroulent en plus de 44 heures sur les ordinateurs de l'époque. A cause de ces exigences, Johnson & al. (1989) proposent assister la méthode par une table de recherche "table lookup" comportant des approximations de la probabilité d'acceptation. Szu & Hartely (1987) proposent à leur tour un Recuit Simulé Rapide RSR, qui permet occasionnellement des grands sauts afin d'accélérer la convergence [Jai & Mee, 99(2)].

Yamada & al. (1994) ont formulé une méthode appelée "Block Critique Recuit Simulé" utilisant une fonction de voisinage dérivée du block critique. La température initiale et finale sont définies en fonction d'un taux d'acceptation, et des réintensifications sont appliquées pour améliorer la recherche. Les auteurs [Yam & Nak, 96] réimplémentent leur BCRS avec un générateur d'ordonnancements de Giffler & Thompson. Krishna & al. [Kris & al., 95]. Un autre algorithme de Recuit Simulé connu par *Focused Simulated Annealing* FSA, procédant au gonflement du coût des solutions inefficaces, est proposé par Sadeh & Nakakuki [Sad & Nak, 96].

Kolonko [Kol, 99] de son part, a montré que le Recuit Simulé appliqué au Job Shop n'est plus un processus convergent, et le voisinage standard du problème n'est pas symétrique. Par conséquent, il a présenté une méthode hybride combinant le Recuit Simulé et un Algorithme Génétique.

7. Benchmarks

En ordonnancement d'ateliers de type Job Shop, il est souvent utile d'avoir des *benchmarks* pour tester les modèles d'ordonnancement et surtout les méthodes de résolution appropriées.

Les benchmarks sont des problèmes-types construits par plusieurs auteurs pour tester les performances des approches de résolution, en procédant surtout à des comparaisons sur les mêmes instances [Jen, 01].

Il s'agit d'instances théoriques (à l'instar des benchmarks des autres problèmes), formulées pour servir à l'étude du problème de Job Shop.

Un grand nombre de benchmarks de Job Shop sont proposés par plusieurs auteurs. Certains sont largement employés dans les recherches, alors que, d'autres sont moins connus. Parmi ces benchmarks, les suivants [Jen, 01] [Jai, 98] :

- ft06, ft10, ft20[1] connus aussi par : mt06, mt10 et mt20 : sont des problèmes de petite à moyenne taille (6×6, 10×10, 20×5 respectivement), proposé par Fisher & Thompson ; ft10 est le problème le plus utilisé pour tester des méthodes de résolution proposées. Il est publié en 1963 et ce n'est qu'en 1989 que Carlier et Pinson ont trouvé sa solution.

  ft06 et ft20 sont de difficulté moindre, et sont résolus en 1975 par M^{c} Mahon & Florian [Jen, 01].

- la01...la40 : ce sont 40 benchmarks publiés par Lawrence (1984), et couvrent 8 tailles différentes : 10×5, 15×5, 20×5, 10×10, 15×10, 20×10, 30×10, 15×15. Lawrence les dénote respectivement F1-5, G1-5, H1-5, A1-5, B1-5, C1-5, D1-5, I1-5, toutefois la dénomination donnée par Applegate & Cook est communément la plus connue.

- abz05...abz09 : proposés par Adams, Balas et Zawack (1988). Sont cinq benchmarks de deux tailles différentes : 10×10 pour abz05 et abz06, 20×15 pour abz07-09.

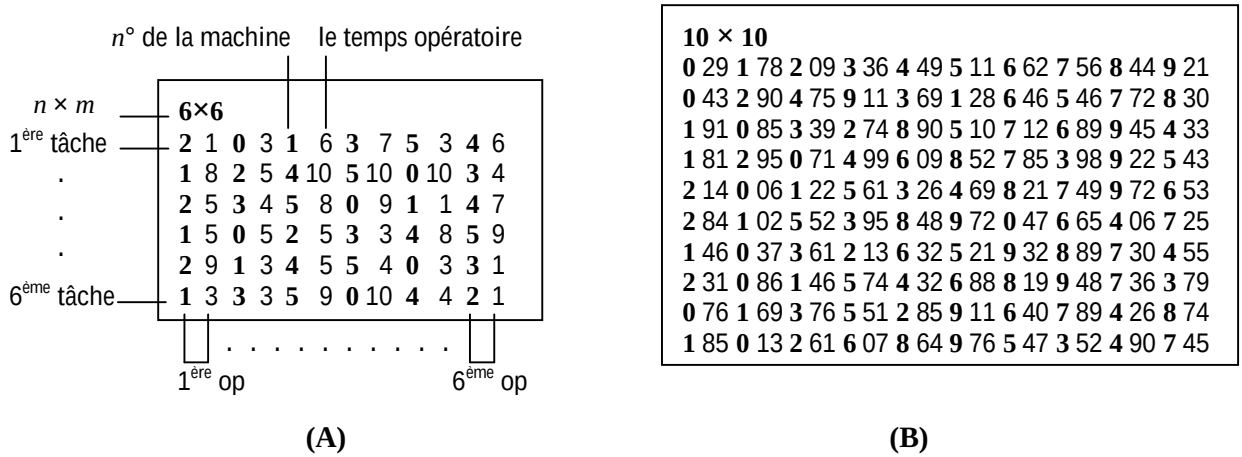
[1]- Ces dénominations ne sont pas originales des auteurs des benchmarks. En effet, ft, la, abz et orb, sont données par Applegate & Cook (1991) ; tandis que, celles de swv et yn sont données par Vaessens (1996) ; et celle de TD par Balas & Vazacopoulos 1998 [Jen, 01].

- orb01...orb10 : sont dix problèmes formulés par plusieurs auteurs : Bill Cook, Monika, Bruce Gamble, Bruce Shepherd, George Steiner ; ils sont alors appelés : bic2, mon2, brg1, brs1, ges1, bic1, mon1, brg2, brs2, ges2. Ce sont des problèmes difficiles et leur résolution nécessite des méthodes rigoureuses.
- swv01...swv20 : publiés par Storer, Wu and Vaccari (1992), vingt problèmes de quatre tailles différentes : 20×10, 20×15, 50×10, 50×10. Sont connus comme difficiles sauf pour la dernière série.
- yn01...yn04 : ces problèmes dont la taille est 20×20, sont formulés par Yamada et Nakano (1992), le temps opératoire est généré dans l'intervalle [10,50].
- td01...td80 : formulées par Taillard. Sont 80 benchmarks de différentes tailles : 15×15, 20×15, 20×20, 30×15, 30×20, 50×15, 50×20, 100×20.
- dmu01...dmu80 : sont 80 problèmes de huit tailles différentes : 20×15, 20×20, 30×15, 30×20, 40×15, 40×20, 50×15, 50×20, formulés par Demirkol, Mehta & Uzsoy (1998).

Les benchmarks présentent les caractéristiques suivantes :

- Chaque benchmark est décrit par un ensemble de n tâches et m machines. Ce sont des problèmes $n \times m$.
- Toutes les tâches comprennent exactement m opérations. Chaque tâche passe alors une et une seule fois sur chaque machine.
- Pour chaque opération, le temps d'exécution (nombre entier), ainsi que la machine sur laquelle elle doit s'exécuter, sont spécifiés.
- Les durées opératoires des opérations sont des entiers compris entre 1 et 100.
- A part le temps opératoire, il n'existe pas d'autres contraintes temporelles comme : la date d'échéance, le temps de préparation, la date de disponibilité etc.
- Les solutions optimales de ces benchmarks sont connues ; ce qui constitue un bon outil pour les expérimentations et les comparaisons.

La figure 2.9 nous montre un exemple de deux benchmarks : mt06 et mt10, sur lesquelles nous constatons les caractéristiques générales des benchmarks.



(A) **(B)**

Figure 2.9 : Exemple de deux benchmarks, (A) : mt06 et (B) : mt10.

Dans notre étude, nous utilisons un échantillon de ces benchmarks comme des problèmes-tests, afin d'étudier la performance des stratégies Métaheuristiques employées dans la résolution du problème de Job Shop.

8. Conclusion

L'ordonnancement d'ateliers de type Job Shop est considéré un problème NP-Difficile. En dépit de sa simple modélisation et définition par précision des données (tâches et machines), des contraintes et des objectifs, la résolution du problème est une tâche difficile, le classant ainsi parmi les problèmes NP-Difficiles au sens fort. Ce qui le rendant un terrain d'essai privilégié pour le test d'une multitude de méthodes notamment les nouvelles approches de résolution, comme les Métaheuristiques, qui font le sujet de notre application, et qui sont présentées dans le prochain chapitre.

CHAPITRE III

Les Métaheuristiques

Résumé : *L'objectif de ce chapitre est de présenter succinctement les principes généraux des trois Métaheuristiques qui constituent l'objet de notre étude : les Algorithmes Génétiques qui font partie des méthodes évolutives ; la Recherche Tabou et le Recuit Simulé qui sont les principales méthodes basées sur la recherche locale. Pour chaque méthode, nous présentons son principe de fonctionnement et ses ingrédients essentiels.*

CHAPITRE III

Les Métaheuristiques

1. Introduction

Les Métaheuristiques, sont un paradigme très important des méthodes approchées destinées à la résolution des problèmes combinatoires comme celui de Job Shop. Grâce à elles, on peut proposer aujourd'hui de bonnes solutions pour des problèmes de plus grande taille, et pour de très nombreuses applications qu'il était impossible de traiter auparavant.

Nous présentons dans ce chapitre les principes fondamentaux des trois Métaheuristiques qui constituent l'objet de ce mémoire : les Algorithmes Génétiques, la Recherche Tabou et le Recuit Simulé. Nous donnons d'abord, un aperçu général sur les Métaheuristiques. Ensuite, nous évoquons dans les trois sections qui suivent, les notions fondamentales relatives aux trois Méthodes retenues en précisant notamment les principes généraux et le fonctionnement de chacune.

2. Présentation des Métaheuristiques

Les problèmes d'optimisation combinatoire comme le problème de Job Shop, sont pour la majorité NP-difficiles, et ne possèdent pas à ce jour de solutions algorithmiques efficaces valables pour toutes les données [Hao & al., 99]. Le recours à des méthodes approchées ou heuristiques constitue une alternative indispensable. Depuis une vingtaine d'années, les heuristiques les plus populaires, et également les plus efficaces [Her, 05], destinées à la résolution des problèmes d'optimisation combinatoire, sont des techniques générales, appelées Métaheuristiques, qu'il s'agit d'adapter à chaque problème particulier.

2.1. Concept de Métaheuristiques

Les Métaheuristiques sont des méthodes générales destinées principalement à la résolution des problèmes d'optimisation combinatoire dont le temps de résolution croît exponentiellement.

Le terme de Métaheuristiques introduit pour la première fois par Glover (1986), vient des deux mots grecs : *heuriskein* qui veut dire *trouver* et *meta* qui signifie *au-delà* [Blu & Rol, 03].

« Les Métaheuristiques sont des stratégies de recherche itérative de haut niveau, destinées à l'exploration de l'espace de solutions par l'utilisation de différentes techniques. Ces méthodes n'essayent pas d'examiner toutes les solutions possibles, mais de trouver dans un temps raisonnable une solution satisfaisante par investigation intelligente de l'espace en s'appuyant sur deux principes : la diversification de la recherche dans tous "*les endroits*" de l'espace ; et l'intensification, en exploitant chaque point de l'espace d'état » [Blu & Rol, 03].

Toutes les Métaheuristiques s'appuient sur un équilibre entre l'intensification de la recherche et sa diversification [Col & al., 96]. Ne pas préserver cet équilibre conduit à une convergence trop rapide vers des minima locaux (manque de diversification) ou à une exploration trop longue (manque d'intensification) [Tai, 02].

2.2. Classes des Métaheuristiques

Généralement, on distingue deux grands groupes de Métaheuristiques : Les méthodes à base de population ou méthodes évolutives, et les méthodes à point de recherche unique. Cette classification est basée sur le nombre de solutions manipulées à chaque itération.

2.2.1. Les méthodes à base de population

Ces méthodes, comme leur nom l'indique, travaillent sur une population de solutions à la fois. Leur principe général consiste à combiner des solutions entre elles pour en former de nouvelles, en essayant d'hériter les bonnes caractéristiques des solutions parents. Un tel processus est répété jusqu'à ce qu'un critère d'arrêt soit satisfait. Parmi les Métaheuristiques à base de population, on trouve deux grandes classes : les Algorithmes Génétiques et les Colonies de Fourmis.

2.2.2. Les méthodes de recherche locale

S'appellent encore Métaheuristiques à trajectoire ou de voisinage [Blu & Rol, 03]. Sont des méthodes dont le principe général est de démarrer par une solution unique x, et tentent de l'améliorer à chaque pas de leur progression. L'ensemble des solutions que l'on peut atteindre à partir d'une solution x est appelé *voisinage* $N(x)$ de cette solution. Déterminer une solution voisine de x dépend bien entendu du problème traité. Cette classe regroupe plusieurs méthodes, les plus importantes sont : la méthode de Descente, le Recuit Simulé, la Recherche Tabou, *Iterated Local Search* (Recherche Locale Itérée), GRASP (*Greedy Randomized Adaptative*

Search), *Guided Local Search* (Recherche Locale Guidée). Les méthodes de Recuit Simulé et Tabou sont plus anciennes et sans doute plus populaires.

Il existe bien d'autres méthodes importantes qui combinent entre les avantages des deux paradigmes par l'approche de l'hybridation. Parmi ces méthodes, on a les Algorithmes Mémétiques (Genetic Local Search), Scatter Search, GA/PM.

3. Les Algorithmes Génétiques

Les Algorithmes Génétiques sont des méthodes de recherche de solutions approchées, basées sur des mécanismes s'inspirant des processus d'évolution naturelle (méthode évolutive).

Les premiers travaux sur les Algorithmes Génétiques ont commencé dans les années cinquante. Entre 1960 et 1970, John Holland sur la base des travaux précédents, développa les principes fondamentaux des Algorithmes Génétiques [Col & al., 96] [Hau & Hau, 04]. Cette section présente le modèle de base des Algorithmes Génétiques, puisque il existe plusieurs variantes de la méthode qui engagent des mécanismes plus complexes.

3.1. Principes généraux des Algorithmes Génétiques

Les Algorithmes Génétiques tentent de simuler le processus d'évolution naturelle. Ils utilisent un vocabulaire similaire à celui de la génétique. On parlera ainsi d'individu dans une population, l'individu est représenté par un chromosome, les chromosomes sont constitués de gènes qui contiennent les caractères héréditaires de l'individu. Les opérateurs de sélection, de croisement, de mutation simulent les processus naturels du même nom.

Pour un problème d'optimisation donné, un individu représente un point de l'espace d'états, on lui associe la valeur de critère à optimiser. On génère ensuite de façon itérative des populations d'individus sur lesquelles on applique des processus de sélection, de croisement et de mutation.

Pour utiliser un Algorithme Génétique pour un problème particulier, on doit donc disposer des cinq éléments suivants [Whi, 94] [Hau & Hau, 04] :

- **Le codage des solutions (individus) :** associe à chacun des points de l'espace d'états une structure de données. Elle vient généralement après une phase de modélisation mathématique du problème traité. La qualité du codage conditionne le succès de l'algorithme.
- **Une méthode de génération de la population initiale :** la population d'individus produite initialement qui servira de base pour les générations futures doit être non homogène.

- **Une fonction à optimiser :** ou fonction d'évaluation de l'individu. Cette fonction retourne une valeur réelle appelée *fitness*, qui va permettre de déterminer la probabilité de sélection d'un individu [Duv & al., 98].
- **Les opérateurs génétiques :** permettent de diversifier la population au cours des générations et d'explorer l'espace d'états. Le croisement recompose les gènes d'individus de la population. La mutation entraîne des altérations minimes sur les individus pour éviter la convergence rapide de la population. La sélection favorise les meilleurs individus.
- **Les paramètres de dimensionnement :** taille de la population, nombre total de générations ou critère d'arrêt, probabilités d'application des opérateurs de croisement et de mutation.

Le principe de l'Algorithme Génétique se résume par l'algorithme 3.1.

Algorithme : Algorithme Génétique ;
Début
Génération d'une population initiale de k individus ;
Répéter
1. Evaluation de la fonction objectif f de chaque individu ;
2. Sélection des meilleurs individus ;
3. Croisement entre deux individus sélectionnés : obtention de deux enfants issus de deux parents sélectionnés ;
4. Mutation : transformation aléatoire des gènes de certains individus de la population ;

Jusqu'à (condition d'arrêt**)**
Retourner la meilleure solution ;
Fin.

Algorithme 3.1 : Principe de l'Algorithme Génétique.

3.2. Le codage

Les Algorithmes Génétiques agissent sur une population d'individus. Chaque individu de la population est codé par un chromosome ou génotype. Une population est donc un ensemble de chromosomes, et chacun code pour un point de l'espace de recherche. L'étape de codage associe à chacun des points de l'espace d'état une structure de données.

Plusieurs types de codage sont utilisés ; les plus répandus sont : le codage binaire, le codage par permutations de valeurs entières et le codage par valeurs [Hau & Hau, 04] [Fon, 99].

3.2.1. Le codage binaire

C'est le codage utilisé dans les premiers Algorithmes Génétiques [Hau & Hau, 04]. Un chromosome est représenté sous forme d'une chaîne de bits contenant l'information nécessaire à la description d'un point dans l'espace d'états.

3.2.2. Le codage par permutations de valeurs entières :

Chaque gène est codé par une valeur entière. Ainsi, un chromosome se présente dans la forme standard comme une chaîne d'entiers.

Ce codage est bien adapté à l'optimisation des problèmes industriels réels [Fon, 99].

3.2.3. Le codage par valeurs :

Avec ce codage, chaque gène est codé par une valeur prise dans un ensemble fini ou infini. Ces valeurs sont bien entendu liées au problème à résoudre.

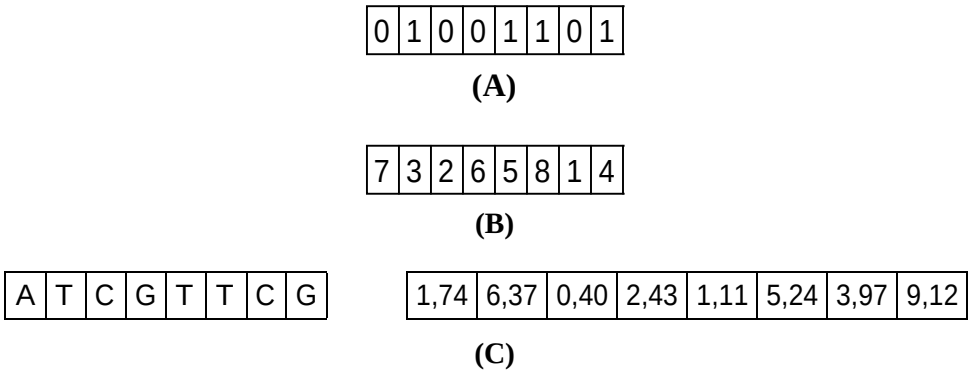


Figure.3.1 : Exemples de différents types de codage.

(A) Codage binaire.
(B) Codage par permutation de valeurs entières.
(C) Codage par valeurs.

3.3. Les opérateurs de reproduction

Les opérateurs de reproduction intervenant dans les Algorithmes Génétiques sont de trois : l'opérateur de sélection, l'opérateur de croisement et celui de mutation.

3.3.1. Le croisement

L'opérateur de croisement a pour objectif de recombiner les chromosomes d'une paire d'individus sélectionnés (*parents*), afin de créer une nouvelle paire d'individus (*enfants*) qui héritent de certaines caractéristiques de leurs parents. Le croisement est mis en place pour que les nouveaux chromosomes gardent la meilleure partie des chromosomes anciens. Ceci dans le but d'obtenir de meilleurs chromosomes [Whi, 94].

Plusieurs stratégies de croisement sont utilisées [Hau & Hau, 04] [All & Dur, 05] :

a. Le croisement à un point simple :

Dans ce type (figure 3.2), un seul site de croisement est choisi. Chaque enfant alors, se construit par l'une des deux sous-chaînes terminales de chacun des deux chromosomes.

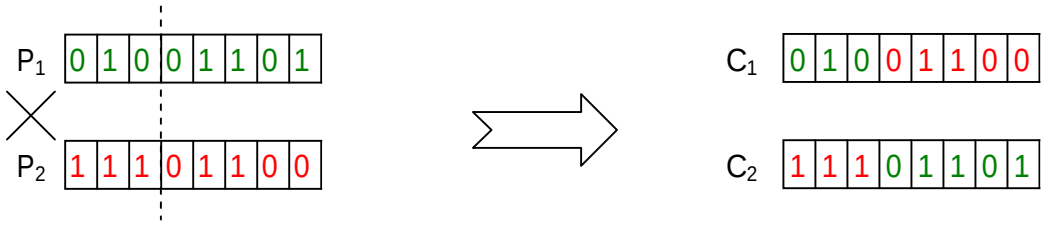


Figure 3.2 : Exemple de croisement à un point simple.

b. Le croisement multi-points :

Le découpage du chromosome peut se faire non pas en 2 sous-chaînes mais en 3, 4, etc. Ce qui permet d'échanger différents morceaux compris entre deux sites de croisement.

Un cas particulier du multi-points, est le croisement bi-points (figure 3.3) qui consiste à choisir deux points de croisement et à échanger les segments des deux parents déterminés par ces deux points.

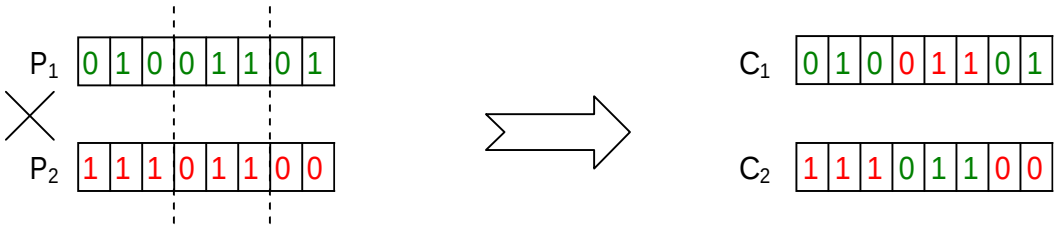


Figure 3.3 : Exemple de croisement bi-points.

Ce croisement est employé surtout dans les cas discrets. Pour les problèmes continus, un croisement dit "*barycentrique*" basé sur un principe similaire est souvent utilisé [All & Dur, 05].

c. Le croisement uniforme :

Plusieurs variantes de ce croisement sont utilisées. L'approche utilisant un vecteur masque est la plus connue. Le masque est une suite de 0 et 1 de la taille du chromosome choisie aléatoirement. Il sert à déterminer de quel parent seront repris les gènes.

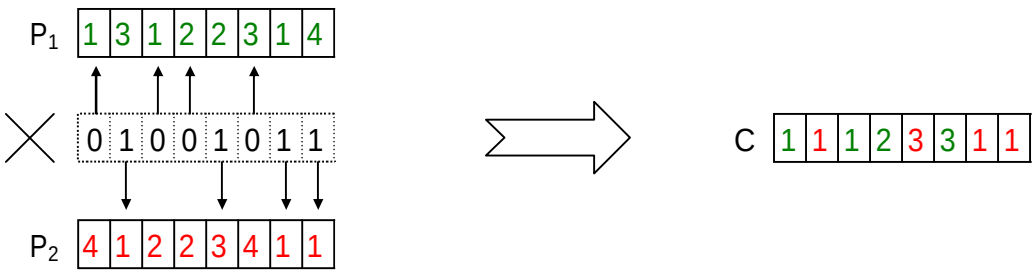


Figure 3.4 : Exemple de croisement uniforme.

Dans des cas particuliers, où les gènes sont soumis à certaines restrictions du codage, ce croisement est appliqué de la façon qui préserve dans les enfants, les restrictions imposées.

d. Le croisement multi-parental :

Consiste à combiner les gènes de plus de deux parents en utilisant par exemple la méthode du simplexe pour orienter la recombinaison [Fon, 99].

L'opérateur de croisement doit tenir compte de la structure particulière du problème traité. En d'autres termes, des opérateurs de croisement "spécialisés" s'avèrent généralement plus performants que des opérateurs généraux du type uniforme. L'idée maîtresse qui doit guider le choix d'un opérateur de combinaison est l'échange d'information pertinente [Vac, 00].

3.3.2. La mutation

La mutation permet de transformer au hasard le codage d'un individu afin d'apporter une certaine diversité dans la population et empêcher que celle-ci converge trop vite vers un seul type d'individu parfait, incapable de sortir d'un minimum local. La mutation est réalisée en modifiant un gène d'un individu pris au hasard. Dans les Algorithmes Génétiques, la mutation est considérée comme un opérateur secondaire par rapport au croisement.

Parmi les stratégies de mutation utilisées en pratique [Whi, 94] :

a. La mutation uni-point :

Cette mutation se fait par altération d'une seule valeur sur le chromosome.

b. La mutation bi-points et multi-points :

Cette mutation se fait par altération de plusieurs valeurs sur le chromosome.

c. La mutation par valeurs :

Avec ce type, la mutation se fait par transformation d'une valeur donnée en une autre valeur déterminée, sur tous les gènes de chromosome.

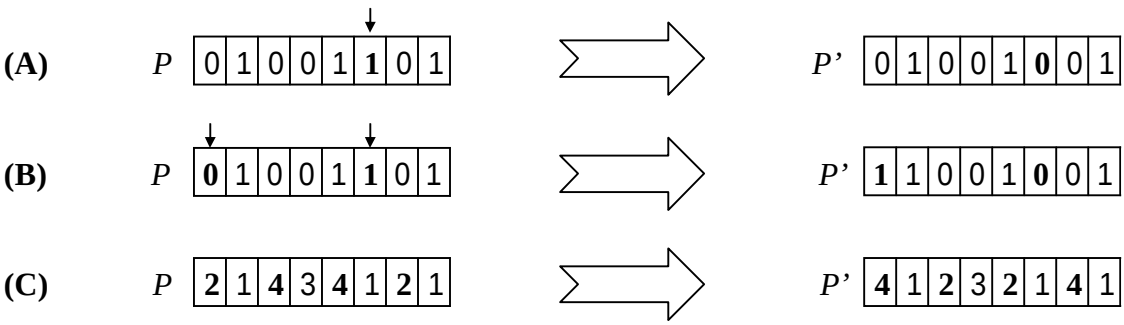


Figure 3.5 : Exemples de différents types de mutation.

(A) Mutation uni-point.
(B) Mutation bi-points.
(C) Mutation par valeurs 2 et 4.

3.3.3. La sélection

La sélection permet aux individus d'une population de survivre, de se reproduire ou de mourir, selon leur adaptation. En règle générale, la probabilité de survie d'un individu dépendra directement de son efficacité relative dans la population.

Il s'agira donc de privilégier les individus ayant un "*score*" d'adaptation le plus élevé ; et de pénaliser ceux avec une faible adaptation. On note que cette adaptation est liée directement à la fonction objectif [All & Dur, 05].

On trouve dans la littérature un grand nombre de stratégies de sélection. Les plus important parmi ces stratégies sont [Hau & Hau, 04] [Whi, 94] [All & Dur, 05] :

a. La sélection par la roue de fortune (roulette wheel) :

Il faut imaginer une sorte de "Roue de fortune" découpée en secteurs, chaque secteur correspond à un chromosome de la population. La superficie de chaque secteur est proportionnelle à la réponse de la fonction objectif. Plus un individu est adapté, plus le secteur qui lui correspond est grand. Les parents sont sélectionnés en fonction de leur performance.

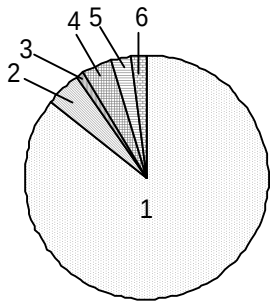


Figure 3.6 : Sélection par la "roue de fortune".

Selon cette méthode, chaque chromosome sera dupliqué dans une nouvelle population proportionnellement à sa valeur d'adaptation (sa *fitness*). La probabilité avec laquelle un individu sera sélectionné dans une population de taille N est déterminée généralement de la façon suivante [Hau & Hau, 04] :

$$Pr(i) = \frac{f(d(c_i))}{\sum_{j=1}^{N} f(d(c_j))}$$

b. La sélection par rang :

La sélection par rang trie d'abord la population par fitness de 1 à N. Ainsi le plus mauvais chromosome aura le rang 1, le meilleur chromosome aura le rang N. La sélection d'un chromosome est la même que par "roulette", mais les proportions sont en relation avec le rang plutôt qu'avec la valeur de l'évaluation. L'exemple du tableau 3.1 illustre cette méthode.

Chromosomes	**1**	**2**	**3**	**4**	**5**	**6**	**Total**
Probabilités de fitness	89 %	5 %	1 %	4 %	3 %	2 %	100 %
Rangs	6	5	1	4	3	2	21
Probabilités de rang	29 %	24 %	5 %	19 %	14 %	9 %	9 %

Tableau 3.1 : Exemple de sélection par rang.

c. La sélection "*steady-state*":

Avec cette stratégie, l'Algorithme Génétique évolue de la manière suivante : à chaque génération, sont sélectionnés quelques chromosomes parmi ceux qui ont le meilleur coût, pour créer des chromosomes fils. Ensuite, les chromosomes les plus mauvais sont retirés et remplacés par les nouveaux. Le reste de la population survie à la nouvelle génération.

d. La sélection par tournoi :

Sur une population de N chromosomes, la sélection par tournoi se fait en comparant des paires d'individus tirées de la population. De chaque paire un seul sera sélectionné selon une probabilité dite de *victoire* du plus fort. Cette probabilité qui doit être grande (entre 70% et 100%), représente la chance qu'a le meilleur chromosome de chaque paire d'être sélectionné. Les tournois peuvent aussi se réaliser par sélection de trois individus (tournoi à trois), le meilleur des trois sera sélectionné avec la probabilité de victoire.

e. L'élitisme

A la création d'une nouvelle population, il y a de grandes chances que les meilleurs chromosomes soient perdus après les opérations de croisement et de mutation. Pour éviter ce problème, on utilise la stratégie d'élitisme. Elle consiste à copier un ou plusieurs des meilleurs chromosomes dans la nouvelle génération. Ensuite, on génère le reste de la population selon l'algorithme de reproduction usuel.

3.4. Les paramètres de dimensionnement

Le processus de l'Algorithme Génétique est guidé par un certain nombre de paramètres fixés à l'avance. La valeur de ces paramètres influence la réussite ou non de l'algorithme. Ces paramètres sont les suivants [All & Dur, 05] [Hüe, 97] :

- La taille de la population N, et la longueur du codage de chaque chromosome l (dans le cas du codage binaire). Si N est trop grand, le temps de recherche par l'algorithme devient

important. Si N est trop petit, la population peut converger trop rapidement vers un mauvais individu.

- La probabilité de croisement p_c : elle dépend de la forme de la fonction de fitness. Plus elle est élevée, plus la population subit des changements importants. Les valeurs généralement admises sont comprises entre 0,5 et 0,9.
- La probabilité de mutation p_m : ce taux est généralement faible puisqu'un taux élevé risque de conduire à une solution sous-optimale, et à la perte de la population originale.
- Le nombre de générations peut également être défini à priori comme critère d'arrêt.

Les paramètres décrits ici sont des paramètres communs, il peut exister plusieurs d'autres selon le modèle de l'Algorithme Génétique utilisé.

4. La Recherche Tabou

Le terme de Recherche avec Tabou ou simplement Recherche Tabou (*Tabu Search*) est apparu pour la première fois dans un article présenté par Glover en 1986, où apparaît également le terme de Métaheuristique [Col & al., 96].

4.1. Le principe général de la méthode

La Recherche Tabou est une méthode de recherche locale. Elle procède à explorer pour une solution courante s, tout le voisinage $N(s)$ à chaque itération. La meilleure solution s' de ce voisinage est retenu comme nouvelle solution même si sa qualité est plus mauvaise que celle de la solution courante s.

Cependant, cette stratégie peut entraîner des cycles, par exemple la séquence de solutions : $s_1 \bullet s_2 \bullet s_1 \bullet s_2$... détermine un cycle de longueur 2.

Pour éviter ce type de cycle, on mémorise les k dernières configurations visitées dans une mémoire à court terme, et on interdit tout mouvement qui aboutit à l'une de ces configurations. Cette mémoire appelée la *mémoire Tabou* ou la *liste des tabous* (qui a donné le nom à la méthode), est l'une des composantes essentielles de cette méthode. Elle permet d'éviter tous les cycles de longueur inférieure ou égale à k. La valeur de k dépend du problème à résoudre et peut éventuellement évoluer au cours de la recherche [Hao & al., 99].

En conservant la liste des tabous, il est possible alors, qu'une solution de meilleure qualité ait un statut tabou. Dans ce cas, on peut accepter tout de même cette solution en négligeant son caractère tabou, c'est l'application du *critère d'aspiration*.

Le principe de la Recherche Tabou est présenté par l'algorithme 3.2.

Algorithme : Recherche Tabou ;
Début
1. Trouver une solution initiale s_0 , et poser $s^* = s_0$, $i = 0$ et $TL = \phi$; (TL : liste taboue)
2. Poser $i = i + 1$;
3. Déterminer le sous-ensemble de voisinage $N^*(s_i) \subset N(s_i)$ tel que :
   $\forall\ s_{i+1} \in N^*(s_i) : (s_i, s_{i+1}) \notin TL$; ((s_i, s_{i+1}): le mouvement de s_i à s_{i+1}) ;
4. Remplacer s_i par s_{i+1} tel que s_{i+1} est la meilleure solution de $N^*(s_i)$;
5. Mettre à jour TL ;
6. Si la condition d'arrêt n'est pas vérifiée, alors aller à 3 ;

Fin.

Algorithme 3.2 : Pseudo-code de la Recherche Tabou.

4.2. Les mécanismes principaux de la méthode

Dans la Recherche Tabou, la mémorisation des solutions récemment visitées dans une liste des tabous, afin d'éviter le cyclage est le mécanisme essentiel de la méthode. Mais d'autres mécanismes [Glo & al., 96] [Sch, 01] [Hao & al., 99] sont employés aussi, pour améliorer le processus de recherche.

4.2.1. La liste des tabous

La liste des tabous est une mémoire à court terme, qui évite le retour à une solution récemment visitée. Sa mise à jour se fait à chaque itération. La mémorisation de configurations entières serait trop coûteuse en temps de calcul et en place mémoire et ne serait sans doute pas la plus efficace. Il est courant de ne garder dans cette liste que des informations, soit sur les caractéristiques des solutions, soit sur les mouvements, au lieu de configurations complètes.

4.2.2. Redimensionnement de la liste des tabous

Dans certains cas, il est nécessaire d'ajuster la longueur de la liste taboue selon la qualité de voisinage. Si la liste est trop courte, la recherche finit par explorer un optimum local de rayon légèrement grand. Les différentes solutions explorées forment un cycle qui va se répéter indéfiniment. A l'inverse, si la liste est trop longue, tous les mouvements peuvent devenir tabous, et la recherche se bloque. Le réglage de la longueur de la liste dépend essentiellement de la topologie de l'espace des solutions, donc de la qualité de voisinage.

4.2.3. Le critère d'aspiration

L'aspiration, est mis en place afin de permettre à certaines solutions d'être acceptées malgré leur interdiction. Ce mécanisme permet de lever le statut tabou d'une configuration, sans pour

autant introduire un risque de cyclage dans le processus de recherche. La fonction d'aspiration peut être définie de plusieurs manières. La plus simple consiste à révoquer le statut tabou d'un mouvement si ce dernier permet d'atteindre une solution de qualité supérieure à celle de la meilleure solution trouvée.

4.2.4. Détection de cycle

Parfois, la recherche peut tomber dans un cycle de longueur plus grande que la taille de la liste taboue qui ne permet pas de l'éviter. Ainsi, la détection de ce cycle se fait en mémorisant l'historique des solutions par le biais d'une mémoire à long terme. On recherche dans cet historique deux sous-chaînes de configurations identiques (mêmes valeurs, même longueur) et consécutives. Si un tel cycle est détecté, il doit être évité. Cette mémoire permet d'éviter de rester dans une seule région de l'espace de recherche et d'étendre la recherche vers des zones plus intéressantes.

4.2.5. Liste des solutions élites

Permet de mémoriser des solutions de bonne qualité afin de les utiliser comme nouveaux points de départ quand la recherche devient improductive pendant plusieurs itérations consécutives.

L'algorithme et les mécanismes évoqués ici de la méthode de Recherche Tabou ne sont qu'une présentation basale de la méthode. Evidement, il existe d'autres mécanismes et d'autres versions plus compliquées qui sont généralement adaptés selon le problème traité.

5. Le Recuit Simulé

Le Recuit Simulé (Simulated Annealing) est l'une des méthodes de voisinage les plus anciennes. Il a acquis son succès essentiellement grâce à des résultats pratiques obtenus sur de nombreux problèmes NP-difficiles [Col & al., 96].

La méthode du Recuit Simulé a été introduite en 1983 par les travaux de Kirkpatrick & al, et indépendamment par V. Cerny en 1985. Mais, les origines de la méthode remontent aux expériences réalisées par Metropolis & al. dans les années 50.

5.1. Le principe général de la méthode

Le principe de la méthode s'inspire du processus d'amélioration de la qualité d'un métal solide par recherche d'un état d'énergie minimum correspondant à une structure stable de ce métal. L'état optimal correspondrait à une structure moléculaire régulière parfaite.

En partant d'une température élevée à laquelle le solide est devenu liquide, la phase de refroidissement conduit la matière liquide à retrouver sa forme solide par une diminution progressive de la température. Chaque température est maintenue jusqu'à ce que la matière trouve un équilibre thermodynamique.

L'application de ce principe à l'optimisation commence par générer une solution initiale. A chaque itération, on calcule l'énergie de l'état (de la solution), et on diminue la température. Le passage de la solution actuelle à une nouvelle (même plus mauvaise) se fait avec une certaine probabilité qui dépend de la nature des deux solutions et l'avancement de la méthode.

5.2. Processus et ingrédients de la méthode

Le processus du Recuit Simulé répète une procédure itérative qui cherche des configurations de coût plus faible tout en acceptant de manière *contrôlée* des configurations qui dégradent la fonction de coût.

On utilise une méthode stochastique pour générer une suite d'états successifs du système en partant d'un état initial donné. Tout nouvel état est obtenu en faisant subir un déplacement (une perturbation) aléatoire au système.

Soit :

- $\Delta E = E_2 - E_1$: la différence d'énergie occasionnée par une telle perturbation,
- T : la température absolue du système,
- k : une constante physique appelée : *constante de Boltzmann*, $k = 1{,}380510^{-23}\ J/K$;

Le nouvel état est accepté si l'énergie du système diminue ($\Delta E \leq 0$) ; sinon, il est accepté avec une probabilité définie par : $p = e^{-\Delta E / kT}$

A chaque nouvelle itération, un voisin $s' \in N(s)$ de la configuration courante s est généré de manière aléatoire. Selon les cas, ce voisin sera soit retenu pour remplacer celle-ci, soit rejeté.

- Si ce voisin est de performance supérieure ou égale à celle de la configuration courante, *i.e.* $f(s') \leq f(s)$, il est systématiquement retenu ;
- Dans le cas contraire, s' est accepté avec une probabilité p qui dépend de deux facteurs :

  - De l'importance de la dégradation $\Delta f = f(s') - f(s)$; les dégradations plus faibles sont plus facilement acceptées.

  - D'un paramètre de contrôle T (la température), une température élevée correspond à une probabilité plus grande d'accepter des dégradations [Hao & al., 99].

La température est contrôlée par une fonction décroissante qui définit un schéma de refroidissement. Le schéma de refroidissement de la température est l'une des paramètres les plus difficiles à régler. Ce schéma est crucial pour l'obtention d'une implémentation efficace.

Sans être exhaustif, on rencontre habituellement trois grandes classes de schémas [Hao & al., 99] :

- Réduction par paliers : la température est maintenue constante pendant un certain nombre d'itérations, et décroît ainsi par paliers.
- Réduction continue : la température est modifiée à chaque itération.
- Réduction non-monotone : la température décroît à chaque itération avec des augmentations occasionnelles.

Parmi les difficultés de la méthode, la détermination de certains paramètres : la valeur initiale de la température (T_0) et le coefficient de décroissance de la température (γ). Le réglage de ces paramètres est assez délicat et repose sur des essais. Certains utilisateurs de cet algorithme posent $\gamma \in$ [0,85, 0,95]. Quant à la température initiale, celle-ci est déterminée empiriquement ou par des méthodes plus sophistiquées [Fon, 99].

L'algorithme 3.3 présente la version simplifiée du Recuit Simulé.

Algorithme : Recuit Simulé ;
Début
1. Déterminer la solution initiale s_0 et la température initiale T_0 ;
2. Poser $s = s_0$ et $T_i = T_0$;
3. Calculer $f(s_0)$;
4. Répéter pour chaque itération i / i = 1,...,n (ou jusqu'à ce que T est proche de 0) :
   a. Choisir $s' \in N(s_i)$;
   b. Calculer $\Delta f = f(s') - f(s_i)$;
   c. **Si** $\Delta f < 0$ **Alors** $s = s_i$;
   d. **Sinon**
      Début
         tirer p dans [0, 1] suivant une distribution uniforme ;
         Si $p \leq e^{(-\Delta f / T)}$: **Alors** $s = s_i$;
         Sinon s_i est rejetée ;
      Fin ;
   e. Calculer $T = \gamma T$;
5. Retourner s.

Fin.

Algorithme 3.3 : Pseudo-code du Recuit Simulé.

6. L'hybridation des Métaheuristiques

Que ce soit, pour élargir le spectre d'application d'une méthode, ou augmenter ces performances, le recours au principe d'hybridation est envisagé depuis longtemps, ce qui permet de créer de nouvelles méthodes hybrides plus sophistiquées et plus performantes.

Une méthode hybride est une méthode de recherche constituée d'au moins deux méthodes de recherche distinctes. Il est possible d'hybrider toutes les méthodes, y compris heuristiques et méthodes exactes [Duv, 00], mais il faut être prudent dans le choix des méthodes à utiliser pour obtenir une bonne coopération. Par exemple, l'hybridation d'une méthode évolutionnaire qui renforce la diversification, et une méthode de recherche locale qui assure l'intensification de la recherche.

L'hybridation peut se faire par plusieurs stratégies (figure 3.7) [Duv, 00] :

- **Hybridation séquentielle :** consiste à exécuter séquentiellement différentes méthodes de recherche, de telle manière que les résultats d'une méthode servent de solutions initiales à la suivante.
- **Hybridation parallèle synchrone :** est obtenue par incorporation d'une méthode de recherche particulière dans un opérateur d'une autre. C'est à dire, une méthode est englobée dans une autre. C'est une technique plus complexe que la précédente.
- **Hybridation parallèle asynchrone :** consiste à faire évoluer en parallèle plusieurs méthodes de recherche qui peuvent être identiques. Cette coévolution nécessite un mécanisme coordinateur qui permet d'assurer une bonne coopération des méthodes.

Il est bien entendu possible de combiner plusieurs stratégies d'hybridation au sein de la même méthode.

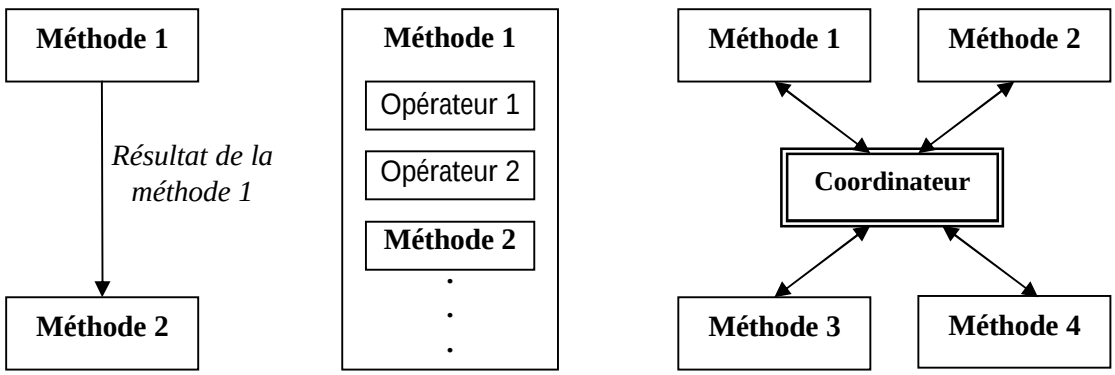


Figure 3.7 : Différentes stratégies d'hybridation [Duv, 00].

7. Conclusion

Les Métaheuristiques sont représentées essentiellement par les *méthodes de voisinage* comme le Recuit Simulé et la Recherche Tabou, et les *méthodes évolutives* comme les Algorithmes Génétiques. Ces trois Métaheuristiques étaient et restent l'objet d'une recherche intense pour la résolution de divers problèmes NP-difficiles dont le problème d'ordonnancement de Job Shop fait partie. Elles ont révélé leur grande efficacité de fournir des solutions approchées de bonne qualité pour un grand nombre de problèmes [Fin & Vo•, 03].

La présentation générale menée au cours de ce chapitre sur les Algorithmes Génétiques, le Recuit Simulé et la Recherche Tabou portant sur leur définition et l'explication de leur principe de fonctionnement a permis de faire ressortir l'idée de base de chacune de ces Métaheuristiques pour conduire ensuite à la description de leur adaptation et application au problème de Job Shop dans le chapitre suivant.

CHAPITRE IV

Application des Métaheuristiques au problème d'ordonnancement de Job Shop

Résumé : *Ce chapitre vise à mettre en application les trois métaheuristiques retenues : les Algorithmes Génétiques, la Recherche Tabou et le Recuit Simulé pour la résolution du problème d'ordonnancement des Job-Shops. Pour chacune de ces méthodes, en se basant sur leur forme basale, plusieurs stratégies ont été implémentées. Le résultat est un outil logiciel "Meta-JSS" sur lequel nous avons effectué des séries d'expériences, dans le but de valider notre approche d'implémentation, et de démontrer l'utilité de ces Métaheuristiques.*

CHAPITRE IV

Application des Métaheuristiques au problème d'ordonnancement de Job Shop

1. Introduction

Dans le but de résoudre le problème complexe d'ordonnancement des ateliers de type Job Shop par les trois Métaheuristiques retenues : les Algorithmes Génétiques, la Recherche Tabou et le Recuit Simulé, nous présentons dans ce chapitre, de manière synthétique l'implantation et la justification des différents choix adoptés pour notre application qui concernent les techniques, ingrédients et paramètres de ces trois méthodes adaptées au problème de Job Shop.

Ce chapitre est subdivisé en quatre sections. Nous consacrons les trois premières à l'implémentation des trois Métaheuristiques retenues. Nous présentons respectivement la résolution du problème par les Algorithmes Génétiques, par la Recherche Tabou et par le Recuit Simulé. Pour chaque méthode, nous discutons les différents choix implémentés, et surtout les algorithmes d'implémentation. Nous terminons le chapitre par une quatrième section qui décrit succinctement le logiciel développé "Meta-JSS" comme résultat de notre implémentation, et qui sert à l'expérimentation sur des problèmes de test.

2. Application des Algorithmes Génétiques

Les Algorithmes Génétiques sont considérés par plusieurs chercheurs comme une méthode bien adaptée au problème de Job Shop, même si elle ne peut pas arriver à l'optimum dans certains cas difficiles [Fan & al., 93].

Notre application porte uniquement sur l'Algorithme Génétique standard, c-à-d, la version basale de la méthode. Des versions plus évoluées et plus sophistiquées ne sont pas appliquées. Cependant, à la place des opérateurs binaires classiques, nous avons utilisé des codages et des

opérateurs améliorés, basés sur des connaissances spécifiques du problème. L'algorithme implémenté est détaillé par l'organigramme de la figure 4.1.

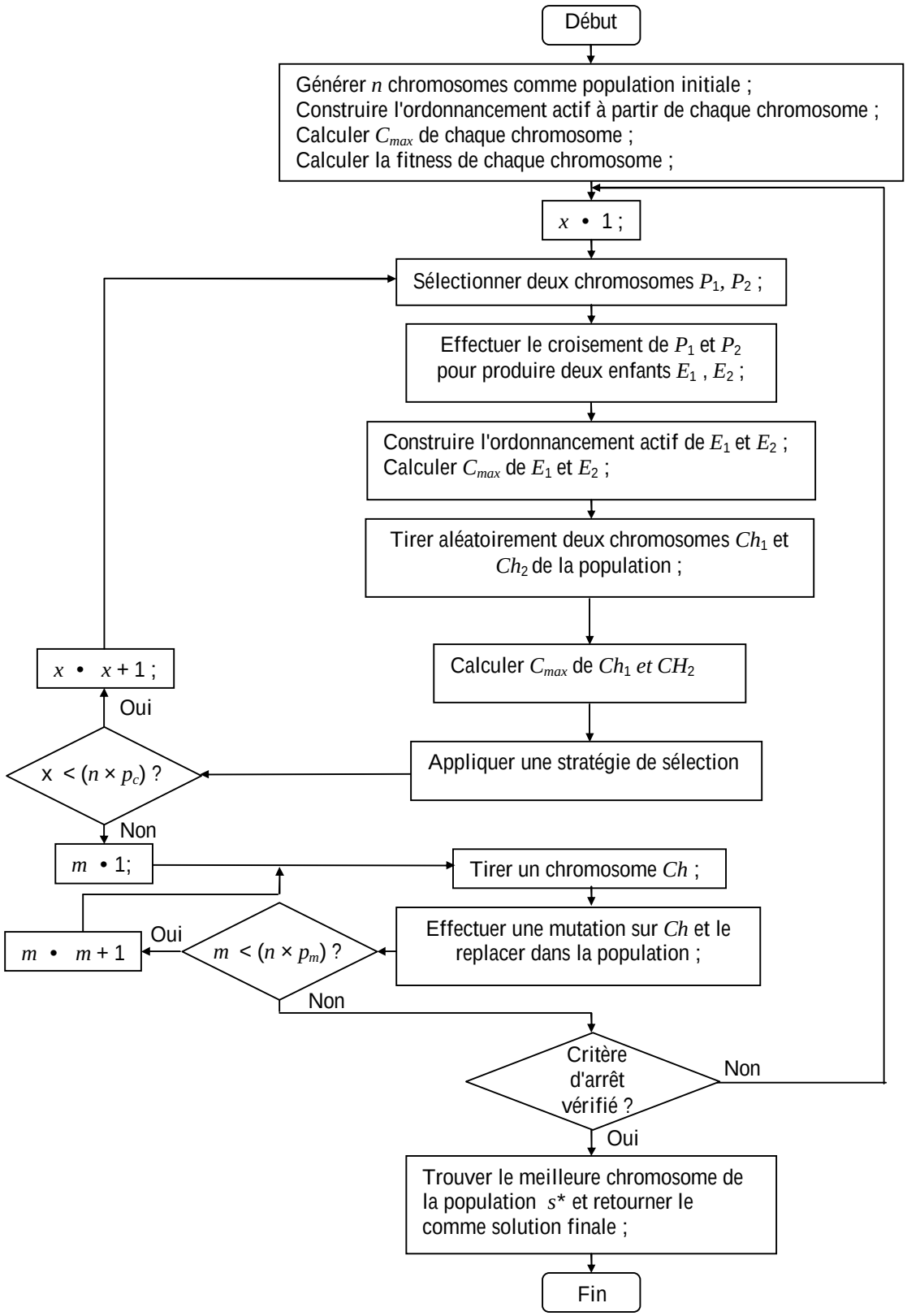


Figure 4.1 : Organigramme général de l'Algorithme Génétique implémenté.

La mise en œuvre de notre Algorithme Génétique passe par :

- L'implémentation de la représentation (codage) convenable au problème ;
- L'implémentation des opérateurs génétiques : croisement, mutation, sélection ;
- La détermination des différents paramètres de la méthode : taille de la population, nombre de générations, taux de croisement, taux de mutation,…

2.1. Le codage des solutions

Le codage est le déterminant important de l'efficacité de la méthode [K•s & al., 99] [Bie, 95]. Il signifie la transcription d'un ordonnancement réel en représentation adéquate permettant la réalisation des différents opérateurs génétiques. Par conséquent, les deux mots "codage" et "représentation" seront employés dans cette section dans le même sens.

Dans notre approche, un mécanisme de réparation de l'ordonnancement est inclus "implicitement" dans tous les algorithmes de codage. Ceci évite l'utilisation d'un module séparé de réparation. En conséquence, chaque chromosome généré correspond impérativement à un ordonnancement réel actif, donné par l'algorithme de construction de l'ordonnancement correspondant. L'avantage de cette approche est d'alléger les algorithmes d'implémentation ; et d'éviter le remaniement d'ordonnancements non admissibles. Pour cette implantation, comme le montre l'organigramme de la figure 4.1, le passage de l'ordonnancement réel à sa représentation n'existe pas ; tous les algorithmes de codage n'intéressent que le passage d'un chromosome à l'ordonnancement actif.

Dans ce travail, nous adoptons quatre stratégies de codage qui sont : le codage basé sur les tâches (Job Based), le codage basé sur les opérations (Operation Based), le codage basé sur les règles de priorité (Priority Rules Based) et le codage basé sur les listes de préférences (Preference List Based). Notons que les deux premiers codages sont directs, alors que ces deux derniers sont indirects.

2.1.1. Le codage basé sur les tâches

Un ordonnancement avec cette représentation utilisée par Holsapple & al. [Hol & al., 93] consiste en une liste ordonnée de tâches. La construction de l'ordonnancement se fait en fonction de la séquence des tâches sur le chromosome : toutes les opérations de la première tâche s'ordonnancent d'abord, puis celles de la deuxième tâche, et ainsi de suite jusqu'à le dernier gène (dernière tâche). Chaque fois que le rôle d'une tâche arrive, toutes ses opérations seront ordonnancées suivant leur ordre dans la gamme opératoire. Donc, la longueur du chromosome est n gènes, où n est le nombre de tâches.

Ce principe de construction de l'ordonnancement est implanté suivant l'algorithme 4.1.

Algorithme codage basé sur les tâches ;
Entrée : 1. $Ch[1,...,n]$: un chromosome de n gènes constitué de permutation de $\{J_1,...,J_n\}$;
2. J_1:$[O_{1,1}$: $p_{1,1},...,O_{1,m}$: $p_{1,m}]$,..., J_n:$[O_{n,1}$: $p_{n,1},...,O_{n,m}$: $p_{n,m}]$: les gammes opératoires des tâches ;
Sortie : • : un ordonnancement actif ;
Début
De $j \leftarrow 1$ **jusqu'à** n
Faire
Début
De $k \leftarrow 1$ **jusqu'à** m
Faire Insertion($O_{Ch[j],k}$) ;
Fin ;
Fin.

Algorithme 4.1 : Construction d'ordonnancement avec le codage basé sur les tâches.

La procédure "insertion" que nous avons utilisée à l'intérieur de cet algorithme, est détaillée par l'algorithme 4.2. Cette procédure est également utilisée par d'autres algorithmes de codage.

Algorithme Insertion($O_{i,j}$) ;
Entrée : 1. $O_{i,j}$: l'opération à insérer ;
2. M_k / $R(O_{i,j}) = M_k$: La machine demandée par $O_{i,j}$;
3. J_1:$[O_{1,1}$: $p_{1,1},...,O_{1,m}$: $p_{1,m}]$,..., J_n:$[O_{n,1}$: $p_{n,1},...,O_{n,m}$: $p_{n,m}]$: la gamme opératoire.
Sortie : $M_k \oplus O_{i,j}$; où $\oplus$ désigne l'opérateur d'ordonnancement de l'opération $O_{i,j}$ sur M_k ;
Début
1. Déterminer $c_{i,j-1}$: fin d'exécution de $O_{i,j-1}$;
2. Trouver sur M_k une période vide p de début t / $p \geq p_{i,j}$, et $t \geq c_{i,j-1}$;
3. $t_{i,j} \neg \max(t, c_{i,j-1})$; $c_{i,j} \neg t_{i,j} + p_{i,j}$;
4. Mise à jour des intervalles vides et occupées de M_k ;
Fin ;

Algorithme 4.2 : Procédure d'insertion d'une opération dans un ordonnancement partiel.

En guise d'exemple, pour le problème défini par les gammes opératoires suivantes :

$J_1[M_1:4 ; M_2:3 ; M_3:3]$; $J_2[M_1:1 ; M_3:5 ; M_2:3]$; $J_3[M_2:2 ; M_1:4 ; M_3:1]$;

La construction de l'ordonnancement pour le chromosome [2,1,3], se fait en ordonnançant d'abord les trois opérations de J_2, puis celles de J_1, ensuite celles de J_3 (figure 4.2).

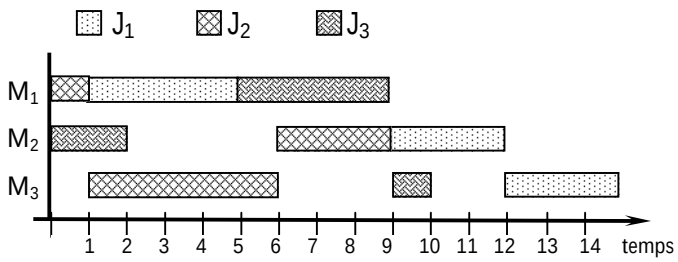


Figure 4.2 : Ordonnancement construit à partir du chromosome [2 , 1 , 3] codé à base de tâches.

2.1.2. Le codage basé sur les opérations

Le codage basé sur les opérations représente un ordonnancement comme une suite d'opérations. Chaque opération est codée par un gène. La désignation de chaque opération peut se faire de deux façons : soit l'opération est désignée par un symbole qui la distingue de toutes les autres, comme le codage utilisé dans le problème du voyageur de commerce (TSP), mais qui est moins fiable pour le Job Shop [Yam & al., 96]. Soit, toutes les opérations d'une tâche donnée portent le même symbole (le numéro de la tâche). L'interprétation de ce symbole se fait suivant l'ordre de son occurrence dans la liste. C'est cette dernière approche utilisée par Gen & al. et Bierwirth [Bie, 95], que nous avons implémentée pour notre application.

Avec ce codage, pour un problème $n \times m$, le chromosome comprend $n \times m$ gènes. Chaque tâche apparaît dans le chromosome exactement m fois (nombre d'opérations). L'occurrence k^{ieme} d'une tâche désigne sa k^{ieme} opération.

L'avantage de cette représentation est qu'elle n'exige qu'un simple constructeur d'ordonnancement [Bie, 95], comme celui de notre application, qui est montré par l'algorithme 4.3. Son autre avantage, est qu'elle couvre l'espace de tous les ordonnancements actifs possibles, même si elle génère une certaine redondance.

Algorithme codage basé sur les opérations ;
Entrée : 1. $Ch[1,\dots,n\times m]$: un chromosome constitué de permutation de $\{J_1,\dots,J_n\}^m$;
2. J_1:$[O_{1,1}$: $p_{1,1},\dots,O_{1,m}$: $p_{1,m}]$,..., J_n:$[O_{n,1}$: $p_{n,1},\dots,O_{n,m}$: $p_{n,m}]$: la gamme opératoire.
Sortie : • : un ordonnancement actif ;
Début
De $j \leftarrow 1$ **jusqu'à** $n\times m$
Faire
1. Déterminer la dernière opération ordonnancée de $Ch[j] = J_i$, soit $O_{i,k-1}$;
2. Insertion($O_{i,k}$) ;
Fin ;
Fin ;

Algorithme 4.3 : Construction d'ordonnancement par codage basé sur les opérations.

A titre d'exemple, considérons le problème précédent (§ 2.1.1), la figure 4.3 présente le principe de construction de l'ordonnancement à partir du chromosome : CH = [2,3,1,2,1,3,1,2,3], codé à base d'opérations.

Pour construire l'ordonnancement représenté par ce chromosome, on ordonnance les neuf opérations dans l'ordre déterminé par le chromosome qui correspond à : $O_{2,1}$ sur M_1, $O_{3,1}$ sur M_2, $O_{1,1}$ sur M_1, $O_{2,1}$ sur M_3, $O_{1,2}$ sur M_2, $O_{3,2}$ sur M_1, $O_{1,3}$ sur M_3, $O_{2,3}$ sur M_2, $O_{3,3}$ sur M_3.

Où, ($O_{i,k}$ sur M_r : signifie que la $k^{ème}$ opération de la tâche i s'ordonnance sur la machine M_r)

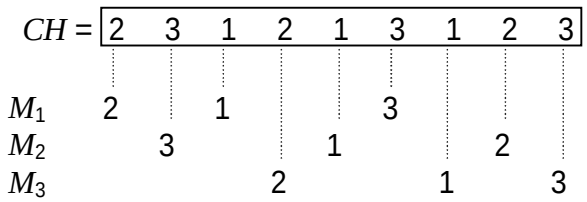


Figure 4.3 : Construction d'ordonnancement à partir d'un chromosome codé à base d'opérations.

2.1.3. Le codage basé sur les règles de priorité

Les règles de priorité sont une sorte d'heuristiques qui tentent à guider le processus de recherche de solutions. Dans ce codage utilisé d'abord par Dorndorf & al. [Dor & Pes, 95], le chromosome est une suite de gènes qui représentent des règles de priorité. La construction de l'ordonnancement dans notre approche à partir d'un chromosome ainsi codé, se fait selon l'algorithme Giffler & Thompson (GT) [Dor & Pes, 95] [Yam & Nak, 92] [T'ki & Bil, 06], dont la décision sur la prochaine opération à chaque étape est guidée par les gènes du chromosome. Ce principe est détaillé par l'algorithme 4.4.

Algorithme codage basé sur les règles de priorité ;

Entrée : 1. Un chromosome $Ch[1,\dots, n\times m-1]$ / $Ch[j] \in \{R_1,\dots,R_r\}$: R_i règle de priorité ;
2. J_1:[$O_{1,1}$: $p_{1,1}$,...,$O_{1,m}$: $p_{1,m}$],..., J_n:[$O_{n,1}$: $p_{n,1}$,...,$O_{n,m}$: $p_{n,m}$] : la gamme opératoire.

Sortie : • : un ordonnancement actif ;

Début

Initialiser C comme l'ensemble des premières opérations de toutes les tâches : C={$O_{i,1}$,...$O_{n,1}$} ; et poser pour chaque opération de C, $ES(O_{ij})$ =0, et $EC(O_{ij}) = p_{ij}$;

De $i \leftarrow 1$ **jusqu'à** $n\times m-1$ **Faire**

Début

1. Déterminer l'opération ordonnançable au plus tôt : $O_{ij}\in C$, et la machine M_r correspondante ;
2. Déterminer l'ensemble de conflit $C[M_r,i] \subset C$, où i-1 est le nombre d'opérations déjà ordonnancées sur M_r ;
3. Choisir selon $Ch[i]$ une opération de $C[M_r,i]$, et ordonnancer la sur la machine M_r ;
4. Mettre à jour C : enlever la dernière opération ordonnancée de C, ajouter l'opération suivante dans C et calculer ES et EC pour les opérations de C ;

Fin ;

Ordonnancer la dernière opération ;

Fin.

Algorithme 4.4 : Principe d'ordonnancement par codage basé sur les règles de priorité.

> C (appelé coupe) est l'ensemble de toutes les opérations ordonnançables pour chaque itération. ES : le temps de début au plus tôt. EC : le temps de fin au plus tôt. $C[M_r,i]$: l'ensemble de conflits qui comprend toutes les opérations susceptibles d'être ordonnancées sur M_r à l'étape i.

Les règles de priorité sont nombreuses. Nous avons utilisé dans notre application dix (tableau 4.1) qui sont déjà employées par plusieurs auteurs comme [K•s & al., 99].

EST	*Earliest Starting Time* : La prochaine opération ordonnancée est celle dont la date de début est antérieure à celles des autres opérations ;
LST	*Latest Starting Time* : La prochaine opération ordonnancée est celle dont la date de début est postérieure à celles des autres opérations ;
EFT	*Earliest Finish Time* : La prochaine opération ordonnancée est celle dont la date de fin est antérieure à celles des autres opérations ;
LFT	*Latest Finish Time* : La prochaine opération ordonnancée est celle dont la date de fin est postérieure à celles des autres opérations ;
SPT	*Shortest Processing Time* : La prochaine opération ordonnancée est celle dont la durée est inférieure à celles des autres opérations non encore ordonnancées ;
LPT	*Longest Processing Time* : La prochaine opération ordonnancée est celle dont la durée est supérieure à celles des autres opérations non encore ordonnancées ;
MOPNR	*Most OperatioNs Remaining ou MTR Most Task Remaining* : La prochaine opération ordonnancée est la première opération non encore ordonnancée de la tâche contenant le plus grand nombre d'opérations non encore ordonnancées.
MWKR	*Most WorK Remaining ou LRT Longest remaining processing Time* : La prochaine opération ordonnancée est la première opération non encore ordonnancée de la tâche dont la somme des durées des opérations non encore ordonnancées est la plus grande.
LWKR	*Least WorK Remaining ou SRT Shortest Remaining processing Time* : La prochaine opération ordonnancée est la première opération non encore ordonnancée de la tâche dont la somme des durées des opérations non encore ordonnancées est la plus courte.
RANDOM	La prochaine opération ordonnancée est choisie aléatoirement parmi les opérations non encore ordonnancées.

Tableau 4.1 : Les dix règles de priorité utilisées pour le codage à base de règles de priorité.

Dans ce codage, la longueur de chromosome est égale au nombre d'opérations moins un, puisque la dernière opération est prise automatiquement, soit $n \times m$-1 pour un problème $n \times m$.

Par exemple, le chromosome suivant : CH = [LPT, LFT, EFT, SPT, LST, EST, SPT] représente un ordonnancement pour un problème de huit opérations.

2.1.4. Le codage basé sur les listes de préférence

Ce codage proposé d'abord par Davis est utilisé par Falkenauer & al. [Fal & Bou, 91] et Croce & al. [DCr & al., 95] pour la résolution des problèmes de Job Shop simples.

Ce type, appelé aussi "codage avec permutations réparties" (Partitionned Permutation Encoding), utilise pour un problème de Job Shop $n \times m$, des chromosomes constitués de m sous-chromosomes. Chaque sous-chromosome est constitué de n gènes qui sont des listes de préférences de chaque machine. Ces listes, ne décrivent pas la séquence des opérations sur les machines mais l'ordre dans lequel les machines "préfèrent" exécuter les tâches. Ainsi, à chaque étape, devant chaque machine, il existe une file d'opérations qui attendent l'exécution, la machine doit prendre une d'elles suivant sa liste de préférence (algorithme 4.5).

Algorithme codage basé sur les listes de préférences ;
Entrée : 1. $Ch[SCh_1,\dots, SCh_m]$ où $SCh_i = [1,\dots, n]$ est une permutation de $\{J_1,\dots,J_n\}$;
2. J_1:$[O_{1,1}: p_{1,1},\dots,O_{1,m}: p_{1,m}],\dots, J_n$:$[O_{n,1}: p_{n,1},\dots,O_{n,m}: p_{n,m}]$: la gamme opératoire.
Sortie : • : un ordonnancement actif ;
Début
Tant que il y a des opérations non ordonnancées **Faire**
Début
Déterminer les opérations actuelles préférées pour chaque machine $Pref[1,\dots,m]$;
De $j \leftarrow 1$ **jusqu'à** m
Faire
Si $Pref[j] = O_{j,k}$ est ordonnançable i.e $O_{j,k-1}$ est déjà ordonnancée
Alors insertion($O_{j,k}$) ;
Sinon Passer à $Pref[j]$ +1 sur la liste des préférences;
Fin ;
Fin ;

Algorithme 4.5 : Principe du codage basé sur les listes de préférences.

Ce type de codage est indirect et nécessite par conséquent un constructeur robuste d'ordonnancement afin de décoder les chromosomes en ordonnancement réels.

Pour illustrer le principe de ce codage, considérons le problème de 3-tâches 3-machines précédent (§ 2.1.1) et la représentation suivante : CH = [(2,3,1) (1,3,2) (2,1,3)]. (2,3,1) est la liste de préférence de la machine M_1, (1,3,2) de M_2, (2,1,3) de M_3.

Le tableau 4.2 détaille les étapes de construction de l'ordonnancement pour ce problème et la figure 4.4 donne le diagramme de Gantt de son ordonnancement final.

Etape	Opérations préférées	Opérations disponibles	Opérations effectuées
1	$J_2 \bullet M_1, J_1 \bullet M_2, J_2 \bullet M_3$	$J_1 \bullet M_1, J_2 \bullet M_1, J_3 \bullet M_2$	$J_2 \bullet M_1$
2	$J_3 \bullet M_1, J_1 \bullet M_2, J_2 \bullet M_3$	$J_1 \bullet M_1, J_2 \bullet M_3, J_3 \bullet M_2$	$J_2 \bullet M_3$
3	$J_3 \bullet M_1, J_1 \bullet M_2, J_1 \bullet M_3$	$J_1 \bullet M_1, J_3 \bullet M_2, J_2 \bullet M_2$	-
4	$J_1 \bullet M_1, J_1 \bullet M_2, J_1 \bullet M_3$	$J_1 \bullet M_1, J_3 \bullet M_2, J_2 \bullet M_2$	$J_1 \bullet M_1$
5	$J_3 \bullet M_1, J_1 \bullet M_2, J_1 \bullet M_3$	$J_1 \bullet M_2, J_3 \bullet M_2, J_2 \bullet M_2$	$J_1 \bullet M_2$
6	$J_3 \bullet M_1, J_3 \bullet M_2, J_1 \bullet M_3$	$J_1 \bullet M_3, J_3 \bullet M_2, J_2 \bullet M_2$	$J_3 \bullet M_2$
7	$J_3 \bullet M_1, J_2 \bullet M_2, J_1 \bullet M_3$	$J_1 \bullet M_3, J_2 \bullet M_2, J_3 \bullet M_1$	$J_3 \bullet M_1, J_2 \bullet M_2, J_1 \bullet M_3$
8	$J_3 \bullet M_3$	$J_3 \bullet M_3$	$J_3 \bullet M_3$

Tableau 4.2 : Exemple de construction d'ordonnancement avec un codage basé sur les listes de préférences.

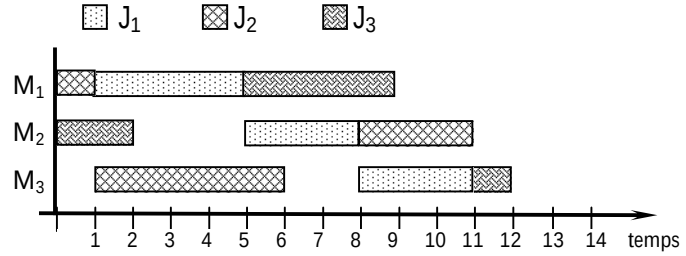


Figure 4.4 : Ordonnancement final donné par les étapes du tableau 4.2.

2.2. Le croisement

Le croisement est un opérateur très important des Algorithmes Génétiques. L'emploi de croisements simples (uni-point, multi-points) ne semble pas fiable pour le problème de Job Shop. Ainsi, plusieurs types de croisement spécifiques du domaine sont proposés. Dans notre application, nous avons implémenté cinq opérateurs de croisement convenables aux codages utilisés :

- Croisement par échange de sous-séquences, (Subsequence Exchange Crossover SXX).
- Croisement de l'ordre des tâches, (Job Order Crossover JOX).
- Croisement de l'ordre linéaire, (Linear Order Crossover LOX).
- Croisement de l'ordre généralisé (Generalized Order Crossover GOX).
- Croisement par préservation de la précédence, (Precedence Preservation Crossover PPX).

2.2.1. Croisement par échange de sous-séquences (SXX)

Le SXX, utilisé par Kobayashi & al. [Kob & al., 95] est une extension du "croisement avec échange de sous-tours" utilisé dans le problème de TSP. La dénomination de *sous-séquences* vient du fait que des sous-séquences sont sélectionnées et échangées entre les chromosomes parents.

Chaque deux séquences échangeables doivent être *symétriques*, c-à-d, elles comprennent des gènes identiques, et ne diffèrent qu'en leur arrangement. Ce croisement consiste à effectuer un échange de sous séquences entre les chromosomes parents pour obtenir des chromosomes fils. Nous avons implanté, comme [Kob & al., 95], ce croisement pour la représentation basée sur les listes de préférences suivant l'algorithme 4.6.

Algorithme le croisement SXX
Entrée : $P_1 = [SCh_{1,1},\dots , SCh_{1,m}]$, $P_2 = [SCh_{2,1},\dots , SCh_{2,m}]$: deux chromosomes parents de m sous-chromosomes $SCh_{i,j}$, où $SCh_{i,j}$ est une permutation de $\{J_1,\dots,J_n\}$;
Sortie : E_1, E_2 : deux chromosomes enfants **;**
Début
  $E_1 \leftarrow P_1$; $E_2 \leftarrow P_2$;
  De $i \leftarrow 1$ **jusqu'à** m
  Faire
  Début
    Trouver dans $E_1[SCh_{1,i}]$ et $E_2[SCh_{2,i}]$ deux sous-séquences échangeables SS_1 et SS_2 ;
    Permuter SS_1 et SS_2 dans $SCh_{1,i}$ et $SCh_{2,i}$;
  Fin
Fin

Algorithme 4.6 : Principe de croisement SXX.

L'exemple présenté sur la figure 4.5, montre la réalisation du croisement SXX sur deux chromosomes parents P_1, P_2, pour obtenir deux fils E_1 ,E_2. Le croisement s'effectue par sélection de sous séquences "symétriques" (en gras sur l'exemple), et les échanger entre les chromosomes fils.

P_1 :	**1**	**2**	**3**	6	4	5
P_2 :	6	**2**	**1**	**3**	4	5
E_1 :	**2**	**1**	**3**	6	4	5
E_2 :	6	**1**	**2**	**3**	4	5

Figure 4.5 : Exemple de croisement SXX.

Outre le codage basé sur les listes des préférences, ce croisement peut être appliqué au codage basé sur les tâches et celui basé sur les opérations.

2.2.2. Croisement de l'ordre des tâches (JOX)

Le croisement JOX est proposé par Yamamura & al. [Yam & al., 96] pour le codage par listes des préférences. Des tâches sont choisies aléatoirement. Les deux fils héritent les gènes correspondant à ces tâches en mêmes positions dans les parents. Le reste des gènes de chaque fils est transféré de l'autre parent, en insérant les gènes ne correspondant pas aux tâches préchoisies dans leur ordre, aux emplacements vides successifs du fils.

En outre de son application à la représentation basée sur les listes de préférences à l'instar de [Yam & al., 96]. Nous l'avons appliqué également au codage basé sur les tâches, et celui basé sur les opérations, suivant l'algorithme 4.7.

Algorithme le croisement JOX
Entrée : P_1, P_2 : deux chromosomes parents constitué de permutations de $\{J_1,...,J_n\}$;
g : le taux génétique ;
Sortie : E : un chromosome enfant ;
Début
1. Poser $l = n \times g$;
2. Choisir aléatoirement un nombre de tâches égale à $random(l/2, l)$;
3. Insérer les tâches choisies dans E aux mêmes emplacements de P_2 ;
4. **Pour** tous les gènes de P_1 **Faire**

Début
Si un gène ne correspond pas à une tâche choisie, **Alors**
poser la dans le premier emplacement vide de E ;
Fin ;
Fin

Algorithme 4.7 : Procédure de croisement JOX.

Pour illustrer ce principe, la figure 4.6 présente un exemple de croisement JOX de deux chromosomes parents P_1 et P_2, qui préserve les tâches J_2 et J_4.

P_1 :	1	**2**	3	6	**4**	5
P_2 :	6	1	**2**	3	**4**	5

Chaque fils hérite les tâches préchoisies, en mêmes emplacements :

E_1 :	.	**2**	.	.	**4**	.
E_2 :	.	.	**2**	.	**4**	.

Ensuite, les gènes vides des fils sont remplis par les tâches non choisies (en ordre) de l'autre parent :

E_1 :	6	**2**	1	3	**4**	5
E_2 :	1	3	**2**	6	**4**	5

Figure 4.6 : Exemple de l'application de JOX.

2.2.3. Croisement de l'ordre linéaire (LOX)

Le croisement de l'ordre linéaire LOX**,** développé par Falkenuaer & al. [Fal & Bou, 91] est une extension du "croisement de l'ordre OX" appliqué au TSP.

Le principe de ce croisement consiste à prélever une sous-séquence sur un parent donneur, puis l'insérer au récepteur, exactement au même endroit de prélèvement. La longueur de la sous-séquence est déterminée par un paramètre que nous avons appelé "le taux génétique". Il détermine la distance entre deux sites qui sont choisis aléatoirement dans [Fal & Bou, 91]. Sachant que les redondances de gènes ne sont pas permises, il faut supprimer d'abord tous les gènes du récepteur identiques à celles de la sous-séquence, puis réarranger le reste des gènes.

Nous avons employé le croisement LOX dans deux cas : Pour le codage basé sur les tâches, et pour le codage basé sur les listes de préférences. Le principe d'implémentation exprimé par l'algorithme 4.8, est identique pour les deux codages.

Algorithme le croisement LOX

Entrée : P_1, P_2 : deux chromosomes parents constitués de permutations de $\{J_1,...,J_n\}$;
g : le taux génétique ;
Sortie : E : un chromosome enfant ;
Début

1. $E \leftarrow P_2$;
2. Poser $l = n \times g$;
3. Déterminer aléatoirement sur E_1 un site d_1 / $d_1 \leq n - l$;
4. Déterminer un deuxième site à $d_2 = d_1 + random(l/2, l)$: "*random*" est une fonction qui génère un entier aléatoire entre deux valeurs $l/2$ et l ;
5. Prélever la sous-séquence SS entre d_1 et d_2 ;
6. Supprimer de E tous les gènes identiques à ceux de SS ;
7. Réarranger les gènes dans E de façon à laisser l'emplacement d_1d_2 vide ;
8. Insérer SS dans E à l'emplacement vide ;

Fin

Algorithme 4.8 : Principe de croisement LOX.

La figure 4.7 présente un exemple d'application de LOX.

P_1 : 1 **2 3 6** 4 5
P_2 : 6 2 5 3 4 1

E hérite de P_1 la sous-séquence sélectionnée :
E_2 : . **2 3 6** . .

Ensuite E se complète avec les tâches non sélectionnées dans l'ordre de P_2 :
E_1 : 5 **2 3 6** 4 1

Figure 4.7 : Exemple d'application de LOX.

2.2.4. Croisement de l'ordre généralisé (GOX)

Le croisement de l'ordre généralisé (GOX), est une autre extension de OX. Il est utilisé par Olivier & al. [Bie, 95] [Sha & Yao, 04] pour le codage basé sur les opérations.

Dans ce croisement, un parent donneur contribue par une chaîne dont la longueur est normalement de la moitié à deux tiers de celle du chromosome pour garantir que le fils hérite l'important de l'information portée par les deux parents. Pour notre application, cette longueur est déterminée par le "taux génétique". La position d'insertion de cette chaîne dans le récepteur est déterminée par une méthode empruntée au problème de TSP, où on emploie un index des gènes des chromosomes, cet index détermine l'ordre de chaque opération dans sa gamme.

L'algorithme 4.9 décrit le principe de ce croisement que nous avons implanté au codage basé sur les opérations. En effet, c'est le seul codage admettant le GOX.

Algorithme le croisement GOX
Entrée : P_1, P_2 : deux chromosomes parents constitués de permutations de $\{J_1,...,J_n\}^m$;
g : le taux génétique ;
Sortie : E : un chromosome enfant ;
Début
1. Poser $l = n \times g$;
2. Construire deux chaînes index I_1 de P_1 et I_2 de P_2 de longueur $n \times m$. La chaîne index détermine l'ordre relatif des occurrences de chaque tâche depuis le début de la chaîne;
3. $E \leftarrow P_2$;
4. Déterminer aléatoirement sur P_1 un site d_1 / $d_1 \leq n - l$;
5. Déterminer un deuxième site à $d_2 = d_1 + random(l/2, l)$;
6. Prélever la sous-séquence SS entre d_1 et d_2 ;
7. Supprimer de E tous les gènes identiques à ceux de SS en valeur et en index ;
8. Réarranger les gènes dans E ;
9. Déterminer un site d'insertion s sur E dont la valeur du gène et son index correspondent à celles du premier gène de SS ;
10. Décaler dans E les gènes à partir de s à la fin de E ;
11. Insérer SS dans E à s ;
Fin

Algorithme 4.9 : Principe de croisement GOX.

L'exemple donné à la figure 4.8 illustre l'application de GOX, sur deux chromosomes parents codés à base d'opérations, P_1: le donneur et P_2 : le récepteur.

P_1:	3	3	1	2	1	3	1	2	2
P_2:	1	2	2	1	3	1	3	2	3
P_1:	3	**3**	**1**	**2**	**1**	**3**	1	2	2
Index :	1	2	1	1	2	3	3	2	3
P_2:	1	2	2	1	3	1	3	2	3
Index :	1	1	2	2	1	3	2	3	3
E:	2	3	**3**	**1**	**2**	**1**	**3**	1	2

Figure 4.8 : Exemple d'application de GOX.

2.2.5. Croisement par préservation de précédence (PPX)

Le PPX proposé par Bierwirth & al. [Bie & al., 96], respecte aussi comme le GOX l'ordre des gènes dans le chromosome. Son principe consiste à utiliser une chaîne (de longueur $n \times m$) de 0 et 1 construite aléatoirement, servant comme "modèle" pour le croisement. Elle détermine l'ordre dans lequel les gènes seront tirés des deux chromosomes parents pour construire le fils. Un 0 signifie que le gène sera tiré du premier parent, un 1 veut dire qu'il viendra du deuxième. Quand un gène est tiré d'un parent, le gène correspondant (en valeur et en index) sur l'autre parent est supprimé. L'algorithme 4.10 présente le principe de l'implémentation de PPX.

Algorithme le croisement PPX
Entrée : P_1, P_2 : deux chromosomes parents constitués de permutations de $\{J_1,...,J_n\}^m$;
g : le taux génétique ;
Sortie : E : un chromosome enfant ;
Début
1. Construire aléatoirement une chaîne T : $[t_1,...,t_{n\times m}]$ / $t_i \in \{0,1\}$;
2. **De** $i \leftarrow 1$ **jusqu'à** $n \times m$ **Faire**
Début
Si $t_i = 0$ **Alors**
Début
2.1. Le i^{eme} gène de E vient du premier gène non marqué de P_1 ;
2.2. Marquer ce gène dans P_1 ;
2.3. Marquer le premier gène non marqué codant pour la même tâche dans P_2 ;
Fin Si ;
Sinon Début
2.4. Le i^{eme} gène de E vient du premier gène non marqué de P_2 ;
2.5. Marquer ce gène dans P_2 ;
2.6. Marquer le premier gène non marqué codant pour la même tâche dans P_1 ;
Fin Sinon ;
Fin De ;
Fin ;

Algorithme 4.10 : Principe de croisement PPX.

La figure 4.9 en donne un exemple d'application sur deux parents P_1 et P_2 selon la chaîne binaire *T*.

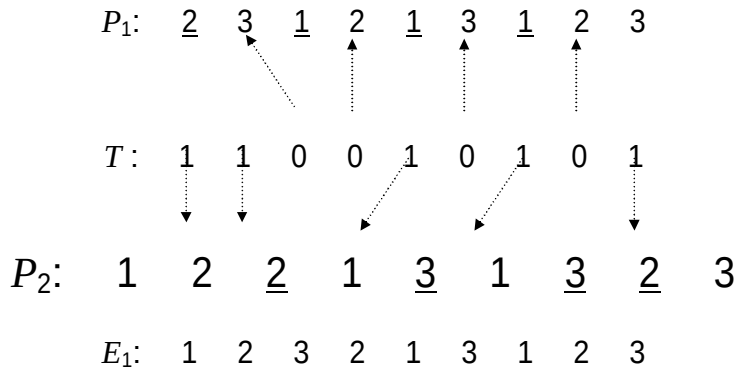


Figure 4.9 : Exemple d'application de PPX.

2.3. La mutation

Le rôle de la mutation est d'apporter une certaine diversité à la population et d'empêcher que celle-ci converge trop vite vers un seul type d'individu. Généralement, la mutation suit deux grands principes : mutation de valeur et mutation de position.

Pour notre problème, plusieurs stratégies de mutation sont proposées dans la littérature, comme : *Multi-step mutation* (MSM), *Multi-Step Mutation Fusion* (MSMF), *Time horizon exchange mutation* (THX-M), *Neighbor Search Mutation* (NSM) [V•z & Whi, 00] [Vac, 00] [Yam, 97]. Puisque l'effet de la mutation est limité sur l'évolution, nous avons utilisé deux stratégies simples et représentatives des grand principes de mutation, il s'agit de : mutation de position (position-based mutation) et mutation à base de tâche (Job-based Mutation).

2.3.1. Mutation de Position

Le principe de la mutation de position [K•s & al., 99] consiste à choisir deux emplacements sur le chromosome en question, puis permuter les deux gènes. Ce processus est répété jusqu'à atteindre l'amplitude (degré) de la mutation, comme cela est présenté par l'algorithme 4.11.

Algorithme La mutation de position
Entrée : *Ch* : un chromosome de longueur *l* ; *amp* : amplitude de mutation ;
Sortie : *Ch* muté ;
Début
De $i \leftarrow 1$ **jusqu'à** $l \times amp$ **Faire**
Début
1. *Choisir aléatoirement deux sites* d_1, d_2 *sur* Ch / $d_1, d_2 \pounds l$ *;*
2. Permuter $Ch[d_1]$ et $Ch[d_2]$;

Fin ;
Fin ;

Algorithme 4.11 : Principe de "Position-Based Mutation".

A titre d'exemple, la mutation par permutation de positions 3 et 8 du chromosome suivant :

[2 3 **1** 2 1 3 1 **2** 3]

lui confère la configuration suivante :

[2 3 **2** 2 1 3 1 **1** 3]

2.3.2. Job-based Mutation

La mutation basée sur la tâche se voit comme une altération du chromosome, portant sur les gènes codant pour une tâche donnée. Deux modèles de cette mutation sont utilisés, mais seulement le premier est retenu pour notre application :

- **Mutation par échange de tâches**

La réalisation de cette mutation se fait par le choix aléatoire de deux tâches, tous les gènes correspondant à l'une des tâches seront remplacés par l'autre.

Algorithme Mutation par permutation de tâches ;
Entrée : *Ch* : un chromosome de longueur *l* ;
Sortie : *Ch* muté ;
Début
1. *Choisir aléatoirement deux tâches à permuter* J_j *et* J_k *;*
2. **De** $i \leftarrow 1$ **jusqu'à** l **Faire**
   Début
   2.1. **Si** $Ch[i] = J_j$ **Alors** $Ch[i] \leftarrow J_k$;
   2.2. **Si** $Ch[i] = J_k$ **Alors** $Ch[i] \leftarrow J_j$;
   Fin ;

Fin ;

Algorithme 4.12 : Principe de mutation par permutation de tâches.

Pour illustrer ce principe, prenons par exemple le chromosome suivant :

2 **3** **1** 2 **1** **3** **1** 2 **3**

La mutation par permutation des tâches 1 et 3 lui apporte la configuration suivante :

2 **1** **3** 2 **3** **1** **3** 2 **1**

- **Mutation par décalage de tâches**

La réalisation de cette mutation se fait par le choix aléatoire d'une tâche. Tous les gènes correspondant à cette tâche seront décalés d'un nombre déterminé de pas.

2.4. La sélection

La sélection est appliquée afin de favoriser au cours du temps les individus les mieux adaptés, à les faire se produire (duplication). Dans notre application, trois stratégies de sélection sont utilisées :

2.4.1. Sélection par la roue de fortune

Notre implémentation de cette stratégie porte uniquement sur la moitié meilleure de la population. Le principe consiste à calculer la probabilité de sélection en fonction de l'écart du makespan de l'individu à la moyenne des makespans de la population avec la formule suivante :

$$Pr(i) = \frac{\overline{C_{max}} - C_{max}(i)}{\sum_{j=1}^{n} (\overline{C_{max}} - C_{max}(j))} \text{ pour toutes les tâches } J_i \text{ , où : } C_{max}(i) < \overline{C_{max}}$$

Selon cette formule, les individus ayant le makespan supérieur à la moyenne ont une probabilité nulle à être sélectionnés. Alors que, ceux avec makespan inférieur à la moyenne, ont une probabilité proportionnelle à l'écart entre leurs makespans et la moyenne.

2.4.2. Sélection "*steady-state*"

Le principe de cette stratégie consiste à sélectionner aléatoirement des chromosomes parmi les $n \times el$ meilleurs chromosomes, où : n est la taille de la population et el est le taux d'*éliticité* (le taux d'éliticité est introduit pour désigner le pourcentage de la population qu'on qualifie d'élite). Le croisement se fait uniquement entre ces meilleurs individus. Les individus les plus mauvais sont retirés et remplacés par les nouveaux. En résumé, ce principe consiste à faire reproduire les meilleurs, et exclure les mauvais, en ne donnant aucune chance aux mauvais de survivre.

2.4.3. Sélection par tournoi à 2 et à 3

La sélection par "tournoi à 2" se fait en comparant des paires d'individus tirées aléatoirement de la population, le meilleur est sélectionné ensuite.

La stratégie de "tournoi à 3" se réalise par sélection de 3 individus de la population, le meilleur des trois est sélectionné.

2.4.4. La stratégie *élitiste*

Du fait de la nature stochastique de la sélection, et le risque de perte des meilleurs individus, nous avons adopté la stratégie élitiste pour toute les implémentation de l'Algorithme Génétique. Ceci consiste à conserver le meilleur individu de la génération courante et l'insérer systématiquement dans la suivante.

2.5. La population initiale

Avant de lancer l'Algorithme Génétique, une population initiale doit être générée. Cet ensemble d'individus qui servira de base pour les générations ultérieures doit être non

homogène. Afin de satisfaire cette non homogénéité, la génération de la population initiale dans notre application se fait de deux façons :

- Soit, en générant des chromosomes aléatoires au lieu d'ordonnancements réels. Cette approche est exploitable pour tous les codages implantés.
- Soit, en utilisant l'algorithme de GT : cette approche n'est implantée qu'avec le codage basé sur les opérations et celui basé sur les règles de priorité.

2.6. L'évaluation des individus

Le seul critère que nous avons utilisé pour l'évaluation des individus est le makespan de l'ordonnancement. Sur ce critère, s'appuie l'estimation de *fitness* des individus. L'évaluation de *fitness* des chromosomes se fait en trois étapes :

- La construction de l'ordonnancement correspondant au chromosome,
- Le calcul du makespan de cet ordonnancement,
- L'attribution au chromosome de son makespan comme estimation de son *fitness*.

Ensuite, le tri des chromosomes et leur sélection se fait selon cette valeur de fitness.

3. Application de la Recherche Tabou

Cette méthode a montré son efficacité pour la résolution de plusieurs problèmes difficiles, parmi lesquels figure le problème de Job Shop [Wat & al., 06].

Notre approche appliquée ici et détaillée par l'organigramme de la figure 4.10, suit l'algorithme standard de la Recherche Tabou. Commençant par une solution (ordonnancement) initiale, le processus de la recherche consiste, à chaque itération, à choisir la meilleure solution qui n'est pas taboue dans le voisinage de la solution courante, même si cette solution n'entraîne pas une amélioration.

Dans le même souci de vouloir exploiter les différents ingrédients de la méthode, nous avons procédé à son implémentation suivant plusieurs stratégies.

Nous présentons dans cette section la mise en œuvre des éléments suivants nécessaires à l'implémentation de la méthode Tabou :

- La génération de la solution initiale,
- La fonction de génération de voisinage,
- L'évaluation de voisinage,
- L'implantation de la liste des tabous,
- L'intégration de quelques mécanismes "standards" renforçant la méthode.

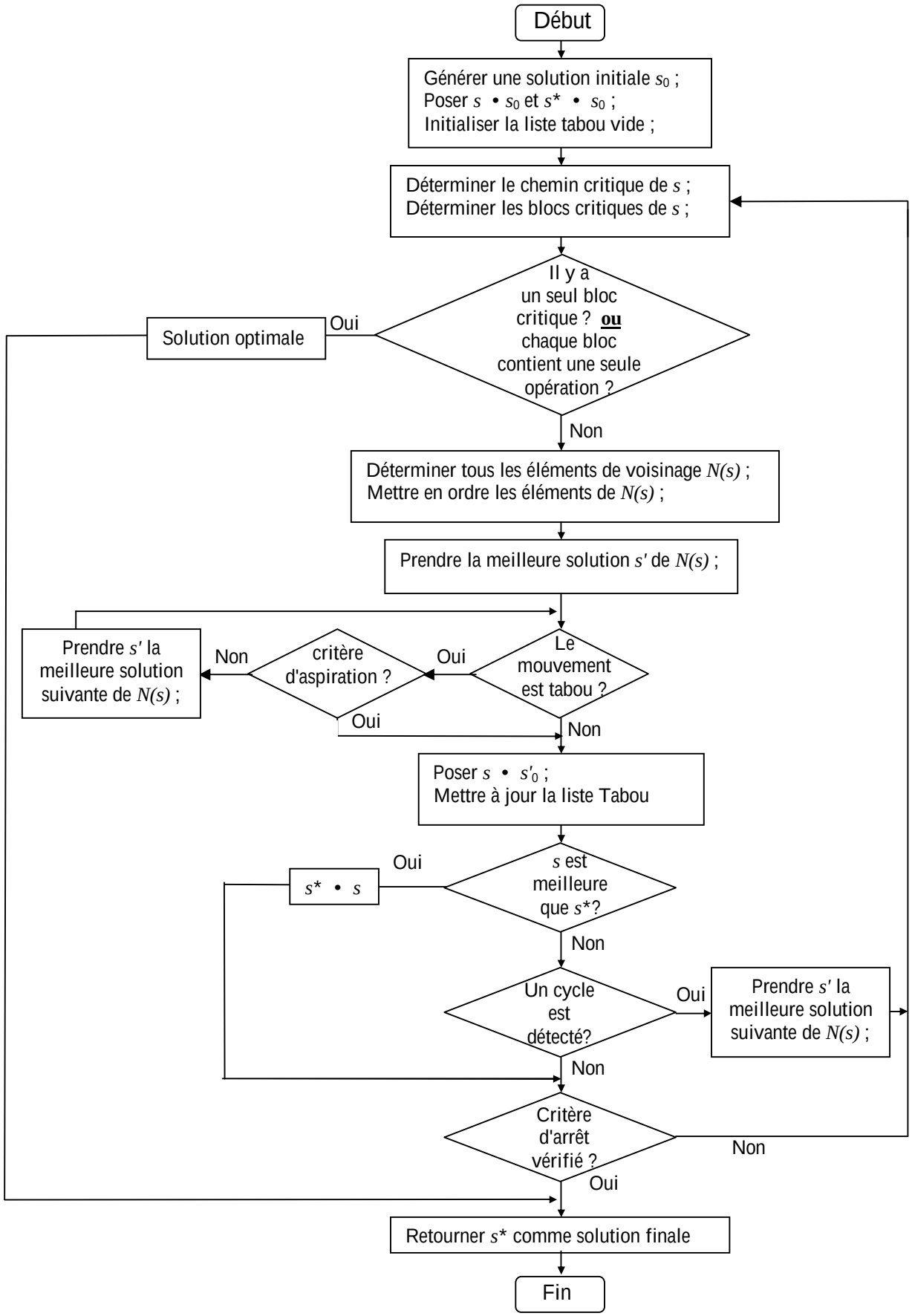


Figure 4.10 : Organigramme général de la méthode de Recherche Tabou implémentée.

3.1. La génération de la solution initiale

Il a été démontré que l'efficacité des méthodes basées sur le principe de la recherche locale dépend étroitement de la qualité de la solution initiale choisie [Jai & Mee, 99(1)]. Dans notre application, trois possibilités de génération de la solution initiale sont exploitées selon la qualité voulue :

- Initialisation par Algorithme Génétique avec un nombre de générations de l'ordre d'une centaine afin d'obtenir une solution initiale de bonne qualité ;
- Initialisation par Algorithme Génétique avec un nombre de générations plus réduit, la solution initiale est alors de qualité moyenne ;
- Initialisation par l'algorithme de Giffler & Thompson aléatoire, la solution initiale générée est de qualité inférieure.

Dans les deux premières possibilités, L'Algorithme Génétique utilisé est codé à base d'opérations.

Le recours à l'Algorithme Génétique pour initialiser la Recherche Tabou (et le Recuit Simulé) est considéré comme une phase d'initialisation de la méthode de recherche locale et non plus une hybridation proprement dite de Métaheuristiques. Effectivement, cette initialisation ne fait pas partie du processus de recherche et peut se faire éventuellement par d'autres méthodes.

3.2. Les fonctions de voisinage

Le voisinage d'une solution courante est constitué de toutes les solutions obtenues en effectuant un *mouvement élémentaire* sur cette solution. Ce mouvement est défini par la fonction de voisinage qui génère pour une solution donnée, un ou plusieurs voisins.

Toute fonction de génération de voisinage doit, dans certaines limites, respecter deux conditions qui sont : la faisabilité et la connectivité :

- La faisabilité signifie que la fonction de voisinage doit générer des solutions admissibles. Toutes les approches que nous avons utilisées garantissent cette condition, puisque toutes les fonctions ne génèrent que des voisins actifs. Si un voisin n'est pas admissible, il est automatiquement omis, et aucun mécanisme de réparation ne sera appliqué.
- La connectivité ou l'accessibilité d'un optimum global est garantie si l'on peut atteindre une solution optimale depuis n'importe quelle état initial admissible. La majorité des voisinages utilisés satisfont également cette condition. De plus, ne satisfaire pas à cette condition dans

notre application ne pose pas de problèmes, puisque le but n'est pas d'arriver à la solution optimale à tout prix, mais plutôt à une solution proche de l'optimale.

Dans notre application, six fonctions de voisinage sont implémentées. Tous ces voisinages sont basés sur la notion de *bloc critique*. Rappelons-nous, que le bloc critique est l'ensemble d'opérations critiques s'exécutant consécutivement sur une même machine.

Ces voisinages, issus des travaux de plusieurs auteurs sont : $N1$, $N2$, NA, RNA, NB et NS.

En s'aidant de quelques exemples, nous présentons dans ce qui suit ces différentes fonctions de voisinage.

3.2.1. Les voisinages $N1$ et $N2$

Le voisinage dénoté $N1$ est un simple voisinage obtenu par inversion de deux opérations consécutives sur le chemin critique. Plus précisément, considérons une solution s, sur laquelle sont déterminées les opérations critiques O_i et $SM[O_i]$. L'application de $N1$ ici se fait par inversion de l'arc $(O_i, SM[O_i])$ qui consiste en réarrangement local des opérations :

$$(PM[O_i], \boldsymbol{O_i}, \boldsymbol{SM[O_i]}, SM[SM[O_i]])$$

en nouvelle séquence :

$$(PM[O_i], \boldsymbol{SM[O_i]}, \boldsymbol{O_i}, SM[SM[O_i]]).$$

(Les notations $PM[O_i]$ et $SM[O_i]$ représentent respectivement le prédécesseur et le successeur de O_i sur la même machine).

Ce principe est illustré par l'exemple de la figure 4.11.

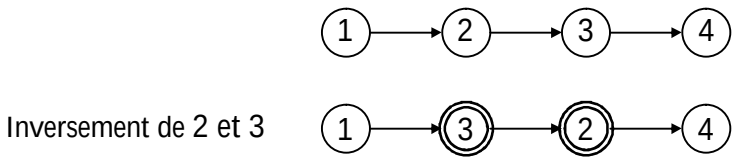


Figure 4.11 : Illustration du principe de voisinage $N1$.

$N1$ a été introduit pour la première fois par Van Laarhoven en 1988 [Sch, 01]. Il est démontré que $N1$ vérifie la connectivité [Hur & Knu, 05], ce qui signifie que l'optimal est joignable après un certain nombre d'itérations. Mais, des expérimentations ont montré de leur part que ce voisinage ne donne pas assez de chances à l'état actuel pour se changer, ce qui peut priver la recherche de visiter les différentes régions de l'espace [Sch, 01].

Le voisinage $N2$ introduit par Matsuo & al. (1988) [Jai, 98], est dérivé de $N1$, mais il restreint le nombre de voisinage. $N2$ utilise le même principe de $N1$, à l'exception qu'il ne

considère pas un arc (O_i, $SM[O_i]$) si les arcs : ($PM[O_i]$, O_i) et ($SM[O_i]$, $SM[SM[O_i]]$) s'étendent tous les deux sur le chemin critique, puisque le reversement de (O_i, $SM[O_i]$) ne peut pas améliorer la solution [Jai, 98]. Cet avantage de réduction de voisinage vient au détriment de la propriété de connectivité, en effet $N2$ ne vérifie pas cette propriété [Sch, 01].

3.2.2. Les voisinages *NA* et *RNA*

Les voisinages *NA* et *RNA* sont proposés par Dell'Amico & Trubian [Del & Tru, 93]. Le voisinage *NA* peut être vu originalement comme une suite de $N1$. Mais, au lieu d'inverser un seul arc dans un bloc, NA considère plus de trois opérations à la fois.

Dans l'exemple de la figure 4.12, le voisinage est obtenu en inversant successivement trois arcs à partir de la solution initiale.

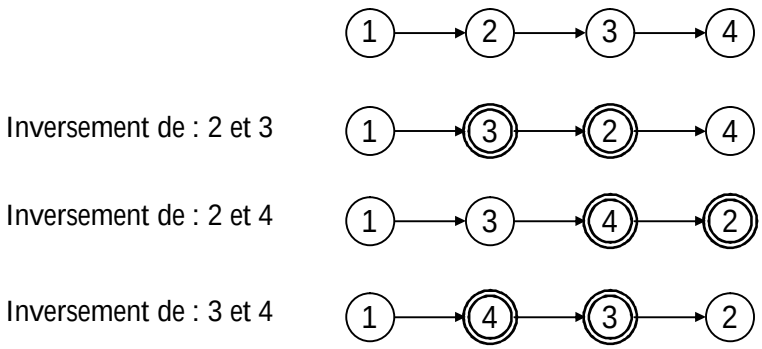


Figure 4.12 : Illustration du principe de voisinage *NA*.

Ce voisinage respecte la connectivité, puisque il se résume en application successive de $N1$, qui vérifie cette propriété.

RNA est une variante de *NA* dans le même sens de $N2$ avec $N1$. *RNA* ne considère pas un arc (O_i, $SM[O_i]$) si les opérations $PM[O_i]$ et $SM[SM[O_i]]$ sont aussi critiques.

3.2.3. Le voisinage *NB*

Ce voisinage utilisé par Grabowski & al. [Jai, 98], et aussi par Dell'Amico & Trubian [Del & Tru, 93] est basé sur la structure de bloc critique.

Le principe de *NB* consiste à déplacer une opération, soit au début, soit à la fin de son bloc critique. D'une autre façon, une opération critique O_i sera déplacée vers l'arrière (respectivement vers l'avant) tant que ce déplacement génère des ordonnancements faisables, jusqu'à atteindre la fin (respectivement le début) de son bloc.

La vérification de la connectivité est démontrée pour ce voisinage. Mais l'inconvénient majeur, c'est la taille considérable du voisinage généré à chaque itération, ce qui rend son

application lourde. L'illustration de ce principe est donnée à l'exemple de la figure 4.13. Si l'on prend par exemple l'opération 2, deux voisins peuvent être générés en la décalant au début ou à la fin du bloc.

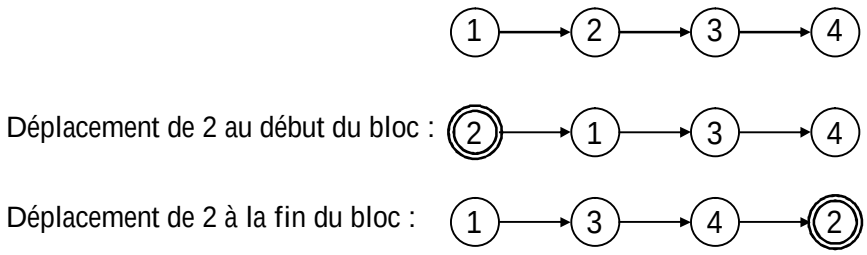


Figure 4.13 : Illustration du principe de *NB*.

3.2.4. Le voisinage *NS*

L'inconvénient majeur des voisinages précédents, est le grand nombre de voisins générés pour une solution donnée. L'impact de leur évaluation sera important sur la performance de la méthode de résolution.

Le voisinage *NS* proposé par Nowicki & Smutnicki est connu comme étant le voisinage le plus réduit. Il permet d'obtenir les résultats en temps relativement court [Wat & al., 06].

L'approche de *NS* génère d'abord un seul chemin critique : si $PM[O_i]$ et $PJ[O_i]$ dénotant le prédécesseur de l'opération O_i respectivement sur la machine et sur la tâche (gamme), sont tous les deux critiques, alors le prédécesseur qui apparaît le premier dans la séquence des opérations sera choisi. Sur ce chemin critique qui comprend b blocs, *NS* inverse le premier et le dernier arc de chaque bloc, sauf le premier et le dernier bloc où le reversement concerne respectivement le dernier et le premier arc seulement.

Sur l'exemple de la figure 4.14, le bloc est constitué de six opérations, trois voisinages sont possibles selon la localisation du bloc : au début, au milieu ou à la fin de l'ordonnancement.

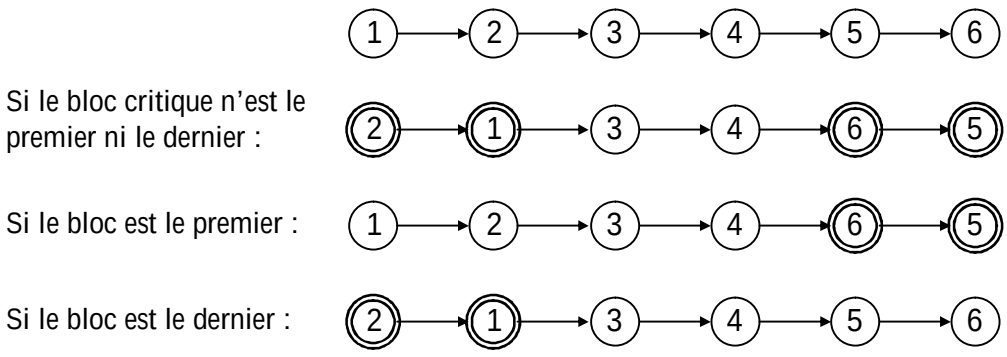


Figure 4.14 : Exemple d'application du voisinage *NS*.

3.3. L'évaluation de voisinage

La complexité d'une approche de résolution basée sur la recherche locale dépend étroitement de la méthode d'évaluation de voisinage afin de déterminer le meilleur voisin selon le critère retenu. Pour cela, il faudrait pouvoir évaluer tous les voisins. Cependant, l'évaluation complète, *i.e.* calcul des dates de début de toutes les opérations, de chaque voisin prend un temps considérable. Il a été démontré que près de 90% du temps de résolution est pris par l'évaluation des voisinages [Duv & al., 98].

Par conséquent, il est intéressant d'utiliser des voisinages aussi restreints que possible afin de diminuer au maximum la complexité de résolution.

Mais, de l'autre côté, la taille de voisinage influe sur le nombre de choix qui s'offrent à la méthode de recherche pour passer de la solution courante à la voisine, plus ce nombre est élevé, plus elle a de chances pour s'échapper de l'optimum local.

Dans notre travail, le compromis entre ces deux situations est obtenu par les considérations suivantes :

- Les voisinages choisis pour notre application sont de taille raisonnable, permettant ainsi de réduire l'effort de leur évaluation, et aussi d'exploiter suffisamment le voisinage.
- Pour évaluer le voisinage nous avons implémenté l'algorithme 4.13, qui vise à calculer seulement les dates de début d'un sous-ensemble d'opérations qui est effectivement concerné par le mouvement et non pas toutes les opérations.

Algorithme Mettre à jour O_j ;
Initialisation : • = [O_j]: • : Liste d'opérations à mettre à jour leurs dates de début ;
Début
Tant que • • • **Faire**
Début
O_i • • [1] ; • • • – • [1] ;
Mettre à jour la date de début de O_i dans l'ordonnancement ;
Si la date de début de O_i est changée **Alors**
Début
Si $SJ(O_i)$ existe **Alors** : • • • + $SJ(O_i)$;
Si $SM(O_i)$ existe **Alors** : • • • + $SM(O_i)$;
Fin ;
Fin ;
Fin ;

Algorithme 4.13 : Mise à jour de l'ordonnancement après mouvement sur O_j.

$SJ(O_i)$ *:* l'opération suivante de O_i dans la gamme opératoire ;
$SM(O_i)$ *:* l'opération suivante de O_i sur la même machine.

Cet algorithme réduit considérablement le temps alloué à l'évaluation de voisinage, puisque seulement un sous-ensemble d'opérations est concerné par la mise à jour des dates de début.

3.4. La mémoire Tabou

Afin d'éviter le piège des optima locaux dans lequel le processus de recherche peut être facilement attrapé, la Recherche Tabou utilise la mémoire des tabous.

3.4.1. Gestion de la mémoire des tabous

La structure de la mémoire implantée est dans sa version élémentaire. Elle se réalise à l'aide d'une liste circulaire de taille k qui peut être constante ou variable. La gestion de la liste suit la stratégie FIFO, l'ordre de dégagement des éléments de la liste est celui de leur insertion. La mise à jour de la liste des tabous se fait à chaque itération de la recherche. A chaque itération un élément nouveau est introduit et un autre le plus ancien est dégagé de la liste.

Les éléments de la liste doivent comporter suffisamment d'informations pour mémoriser fidèlement la solution visitée. La mémorisation de configurations entières serait trop coûteuse en temps de calcul et en place mémoire et ne serait sans doute pas la plus efficace. Par conséquent, elle n'est pas applicable pour la majorité des problèmes. Couramment, ce sont des caractéristiques des solutions, ou des mouvements, qui se gardent dans la liste, au lieu de solutions complètes.

Pour notre application, les éléments de la liste Tabou sont des mouvements : un mouvement marque la transition faite sur une solution pour passer à son voisin.

- Dans le cas de : $N1$, $N2$ et NS : le mouvement est constitué des deux opérations échangées ;
- Dans le cas de : NA, RNA et NB : le mouvement est constitué de l'opération déplacée et sa nouvelle position.

Un mouvement tabou est le mouvement inverse d'un mouvement déjà mémorisé dans la liste des tabous.

3.4.2. Redimensionnement de la mémoire des tabous

Dans notre implémentation, la taille de la liste des tabous peut avoir deux états : soit fixe, soit dynamique. Dans le cas de la taille dynamique, le redimensionnement de la liste se fait suivant la qualité des solutions découvertes. Ainsi, la liste Tabou raccourcit quand l'itération est amélioratrice, et s'allonge si le contraire. La justification de ce choix est que, lorsque une solution meilleure est découverte, alors il est possible de découvrir des meilleures davantage. Il

faut donner donc, plus de chances au voisinage, en réduisant les mouvements tabous. Mais lorsque la solution découverte est pire, la suivante a de grande chance d'être de même. Il faut augmenter le nombre de mouvements tabous afin d'obliger la recherche de sortir de la zone actuelle.

Le raccourcissement et l'allongement se fait d'un élément à chaque fois en ne dépassant pas les limites supérieure et inférieure de la liste. Ces limites sont fixées préalablement à l'instar de [Sch, 01] comme suit :

- La limite inférieure *min* est choisie dans l'intervalle [2, 2 + (*n*+*m*) / 3] ;
- La limite supérieure *max* est choisie dans l'intervalle [*min*+6, *min*+6 + (*n*+*m*) / 3] ;

Où *n* et *m* désignent respectivement le nombre de tâches et le nombre de machines.

3.5. Mécanismes de renforcement

Bien que l'objet de notre travail ne concerne que les approches standards de chaque Métaheuristique, nous avons employé deux mécanismes couramment utilisés, pour renforcer la recherche par la méthode Tabou. Ces deux mécanismes sont le critère d'aspiration et la détection de cycle.

3.5.1. Le critère d'aspiration

Le critère d'aspiration est « une condition nécessaire et satisfaisante pour qu'un mouvement tabou soit accepté malgré son statut tabou » [Sch, 01].

Le critère d'aspiration appliqué ici est celui adopté dans plusieurs applications de la Recherche Tabou au problème de Job Shop, et qui semble plus performant. Ce critère consiste à révoquer le statut tabou d'un mouvement si ce dernier conduira à une meilleure solution obtenue jusqu'à lors.

3.5.2. Détection de cycle

Dans certaines circonstances, en revanche de l'existence de la liste des tabous, la recherche peut tomber dans un cycle de longueur supérieure à la taille de la liste Tabou qui ne permet pas de l'éviter. La détection de ce cycle est importante, afin de permettre à la recherche de sortir de ce bouclage. Dans notre application cela se fait par le biais de deux mécanismes :

- La technique de la mémoire Tabou dynamique, qui est efficace pour l'évasion de ce cyclage, même si la longueur de cycle est plus grande que la limite maximale de la liste des tabous.

- L'intégration d'un mécanisme de mémorisation à long terme [Wat & al., 06], sous forme d'une liste de petite taille (ne dépassant pas une vingtaine d'éléments). Dans cette liste, sont mémorisé par intervalle fixe une petite suite de solutions (en effet les makespans de ces solutions). Lorsque le makespan de la solution actuelle fait partie de cette liste mémorisée, on vérifie le deuxième, si celui-ci existe également, on vérifie le troisième et ainsi de suite. Si enfin, la liste mémorisée est identique à la suite des solutions visitées, alors un cycle est détecté. Son évitement se fait en déposant avec les tabous, le mouvement faisant passer du premier élément de la liste au deuxième. Ce mécanisme peut détecter et éliminer efficacement les cyclages francs où la séquence qui se répète est constante.

4. Application du Recuit Simulé

Le Recuit Simulé est une variante de la recherche locale utilisant la température T pour guider la recherche en passant d'une solution à sa voisine.

L'algorithme de Recuit Simulé que nous avons appliqué ici pour la résolution du problème de Job Shop est également standard, et suit le schéma général montré par l'organigramme de la figure 4.15.

Tout comme pour la Recherche Tabou, l'application du Recuit Simulé au problème de Job Shop, nous a amené à déterminer plusieurs choix pour implanter les stratégies de la méthode. Ces choix concernent :

- La génération de la solution initiale,
- La fonction de génération de voisinage,
- L'évaluation de voisinage,
- La fonction de température,
- La fonction "Energie",
- L'acceptation de transition.

Les choix adoptés pour les trois premiers éléments sont exactement ceux présentés pour le cas de la Recherche Tabou. Les mêmes voisinages utilisés auparavant sont réutilisés ici. La génération de la solution initiale et l'évaluation de voisinage se font également de la même manière. Nous allons présenter dans ce qui suit le reste des mécanismes qui sont spécifiques de la méthode : la température, l'énergie et l'acceptation de transition.

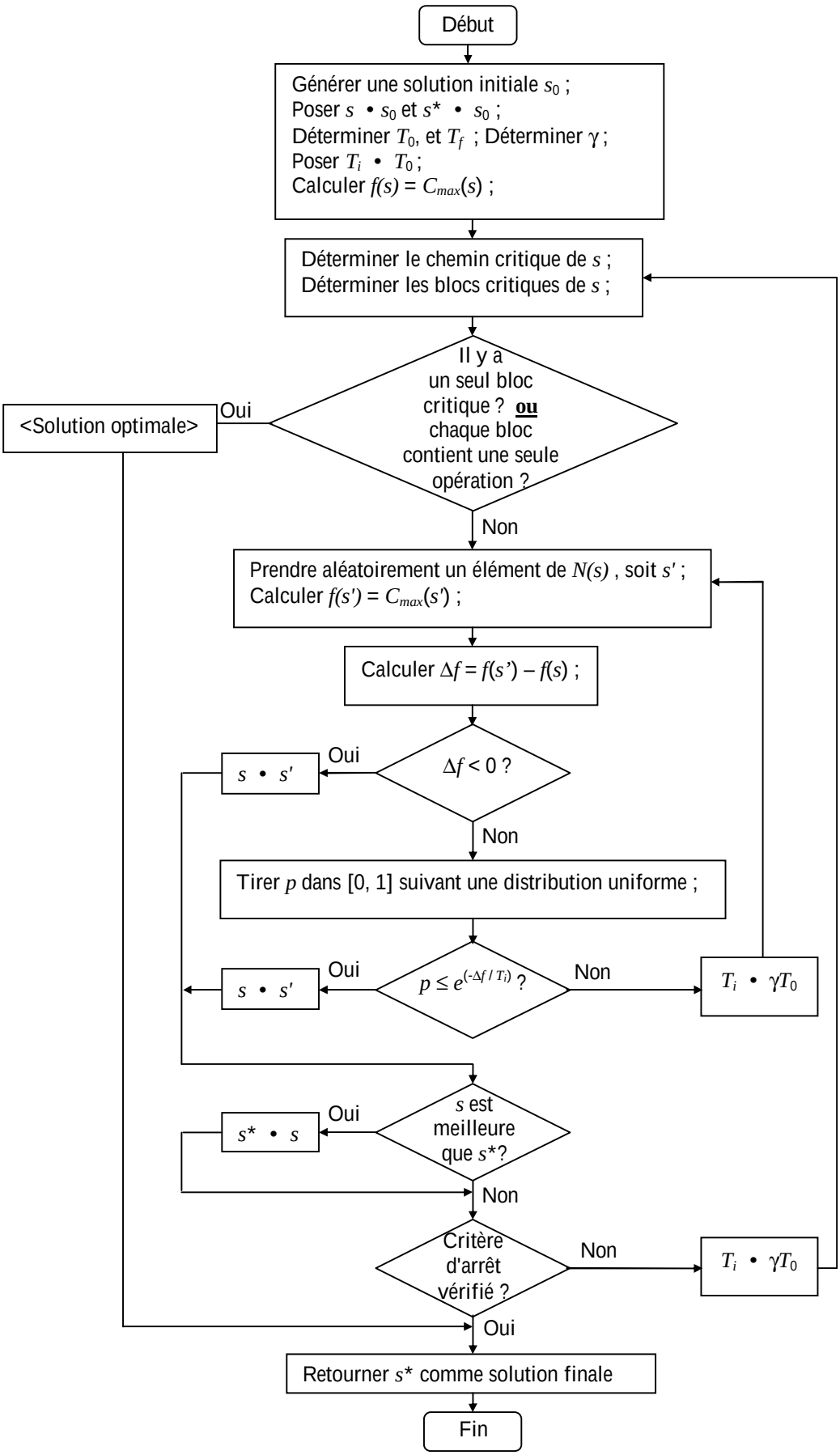


Figure 4.15 : Organigramme général du Recuit Simulé implémenté.

4.1. La fonction de température

La température est le plus important ingrédient de Recuit Simulé. C'est le mécanisme contrôleur, au cours du temps, de transition d'une solution courante à sa voisine. Dans l'algorithme de Recuit Simulé, la fonction de température s'implémente au travers la détermination de : La Température Initiale, la Température Finale et le schéma de Refroidissement.

4.1.1. La Température Initiale

Certains auteurs font intervenir des procédures complexes pour déterminer la Température Initiale [Yam & Nak, 96]. Dans notre application, l'initialisation de la Température peut se faire, soit par des valeurs arbitraires, soit empiriquement, après une série de tests.

4.1.2. La Température Finale

Quant à la Température Finale, Nous avons utilisé deux possibilités de détermination. Soit, elle est fixée, comme la valeur initiale (c'est la procédure couramment employée). Soit, la méthode se déroule avec un facteur de décroissance de température arbitraire, la Température Finale dépendra alors du nombre d'itérations.

4.1.3. Le schéma de Refroidissement

La décroissance de la température se fait graduellement suivant un schéma de Refroidissement. La fonction régissant la température est décroissante. Dans notre application, elle est déterminée par l'une des formules suivantes :

- $T_k = T_0 e^{-ck}$ où :
  - T_k : est la température calculée à l'itération k ;
  - T_0 : est la Températures Initiale ;
  - c : est une constante définie au début par la relation : $c = -\log(T_f / T_0)/k$; T_f : est la Température Finale.
- $T_k = \gamma\, T_{k-1}$ avec : $\gamma \in [0.85, 0.99]$. Dans le cas où la Température Finale est laissée indéterminée [Yam & Nak, 96].

Le Refroidissement avec ces deux fonctions, peut se faire de deux manières :

- Continue : la nouvelle valeur de la température est recalculée à chaque itération k.
- Par paliers : la température est recalculée au début de chaque palier (qui correspond à un certain nombre d'itérations), et demeure inchangée jusqu'à un nouveau palier.

4.2. La fonction Energie

A chaque itération, une solution voisine s' de la solution courante s est générée suivant une distribution uniforme aléatoire, la probabilité de choisir la solution s', $g(s')$ est [Yam & Nak, 96]:

$$g(s') = 1/n \text{ où } s' \in N(s), n \text{ : nombre de voisinage de } s.$$

Une valeur appelée "énergie" est associée à chaque état (solution). Cette énergie correspond à l'évaluation de la solution. Dans notre application, c'est le makespan qui est retenu comme énergie de l'état. La différence d'énergie occasionnée par une transition est donnée par :

$$\Delta E = E_{s'} - E_s = C_{max}(s') - C_{max}(s).$$

4.3. L'acceptation de voisinage

Pour passer d'une solution courante s à une solution voisine s', deux situations sont rencontrées :

- Soit le mouvement améliore la qualité de la solution courante, *i.e.*, $C_{max}(s') \leq C_{max}(s)$. L'acceptation de passage est alors triviale.
- Soit le mouvement détériore la qualité de la solution courante, la probabilité p d'accepter un tel mouvement dépend :
  - D'une part, de l'importance de la dégradation : $C_{max}(s') - C_{max}(s)$, les dégradations les plus faibles sont plus facilement acceptées,
  - D'autre part, de la température courante T_k : une température élevée correspond à une probabilité plus grande d'accepter les dégradations.

  Cette probabilité d'acceptation est définie par : $p = e^{-\Delta E / kT}$

5. Description de l'application informatique

Le résultat de l'implantation des différentes approches et stratégies des trois Métaheuristiques retenues pour la résolution du problème d'ordonnancement de Job Shop, nous a conduit à développer un logiciel simplifié. Il s'agit de Meta-JSS (Metaheuristics for Job Shop Scheduling) construit avec le langage de programmation visuelle Delphi (Pascal Objet).

Bien qu'il existe des bibliothèques de code préconstruites et des logiciels puissants (comme Matlab® par exemple) permettant l'implémentation de ces Métaheuristiques, nous avons préféré le développement d'un outil simple, autonome, dédié au problème traité. Cet outil offre par son interface simple (figure 4.16) un banc d'essai important pour expérimenter et explorer les différentes stratégies.

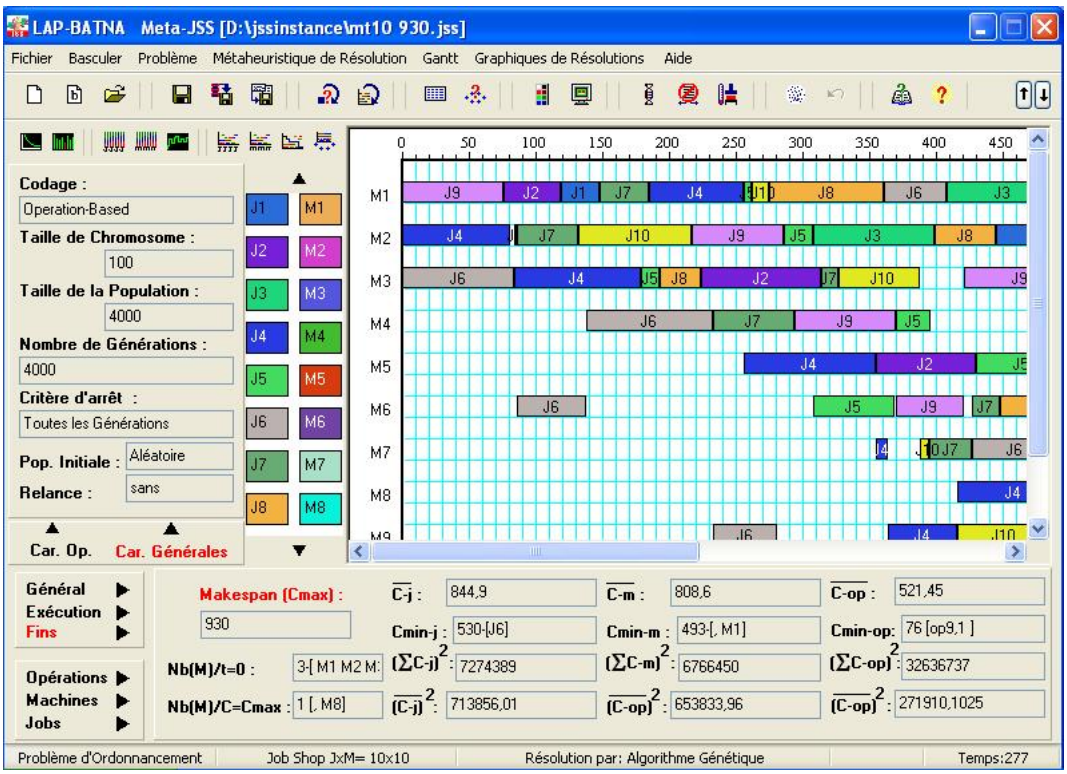


Figure 4.16 : L'interface graphique de Meta-JSS.

Sans s'enfoncer dans la description détaillée de cet outil (qui se trouve dans son fichier d'aide), nous décrivons seulement les étapes essentielles permettant son exploitation : la spécification des instances, le paramétrage des méthodes et la présentation des résultats.

5.1. Spécification des instances (problèmes)

La première étape d'utilisation de Meta-JSS, est la détermination de l'instance de Job Shop constituant le problème à résoudre. Les instances doivent être de type Job Shop simple constitué de m machines et n tâches. Les instances sont de taille petite à moyenne (n et m varient entre 3 et 50). Deux alternatives de détermination des instances sont possibles. L'une d'elles est l'utilisation des benchmarks connus, fournis par le logiciel. L'autre, c'est la construction de nouvelles instances à l'aide du logiciel. Chaque instance doit vérifier un certain format pour qu'elle soit identifiée et résolue par le logiciel. Ce format schématisé par la figure 4.17, se caractérise par :

- La définition de la taille de l'instance à la première ligne : $n \times m$,
- Chaque ligne des n lignes suivantes décrit une tâche,
- La description d'une tâche se fait par précision des opérations successives de la gamme en décrivant chaque opération par la machine demandée suivie de son temps opératoire.

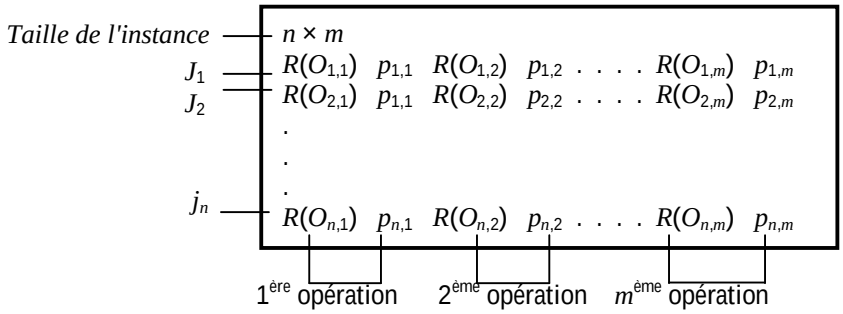


Figure 4.17 : Format général des instances traitées par Meta-JSS.

$O_{i,j}$: $j^{ème}$ opération de la tâche i ; $R(O_{i,j})$: la machine demandée par l'opération $O_{i,j}$; $p_{i,j}$: la durée opératoire de $O_{i,j}$.

La figure 4.18 présente un exemple du benchmark mt10 (ft10) fourni par MetaJSS.

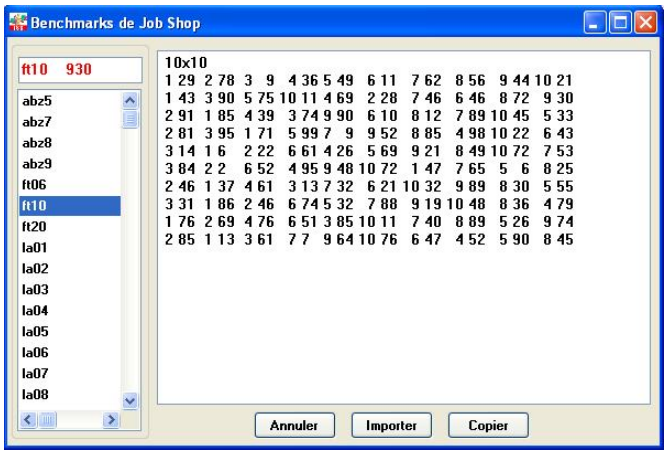


Figure 4.18 : Exemple d'un benchmark présenté par Meta-JSS.

5.2. Paramétrage et mise en œuvre des méthodes

Après détermination de l'instance-problème, vient l'étape de résolution qui commence par l'appel de la méthode choisie. La fenêtre de paramétrage de la méthode s'affiche (figure 4.19). Sur cette fenêtre, on doit déterminer les paramètres et les opérateurs de la méthode. Ces paramètres diffèrent d'une méthode à l'autre, et leur détermination se fait via certaines étapes. Les figures 4.20, 4.21 et 4.22, présentent les plans généraux de l'exploitation de chacune des trois méthodes.

Figure 4.19 : Fenêtre de paramétrage des Algorithmes Génétiques.

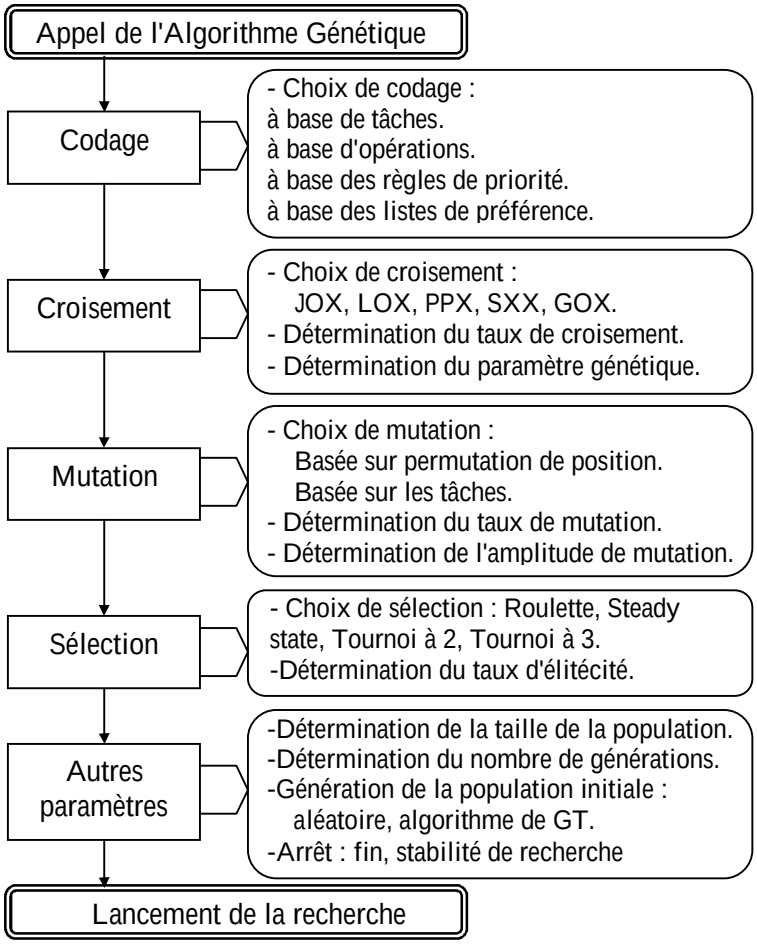


Figure 4.20 : Plan général de l'utilisation des Algorithmes Génétiques dans Meta-JSS.

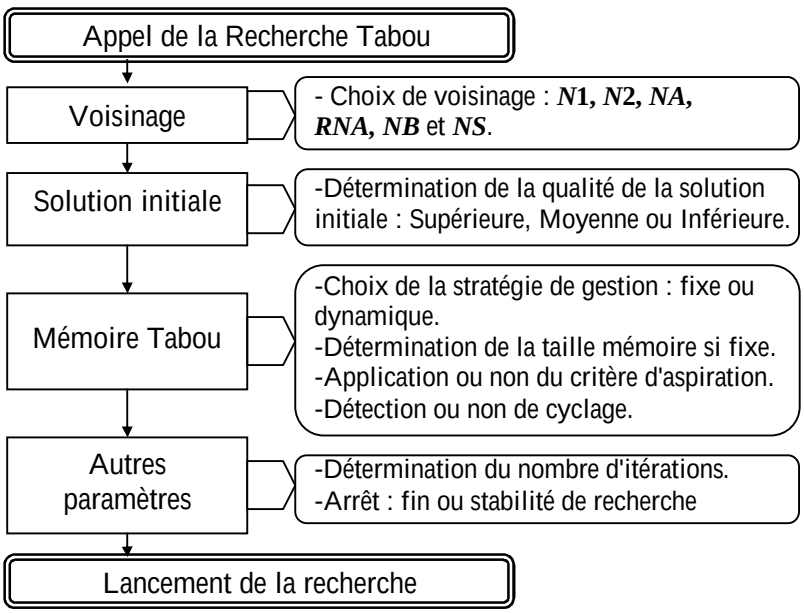


Figure 4.21 : Plan général de l'utilisation de la Recherche Tabou dans Meta-JSS.

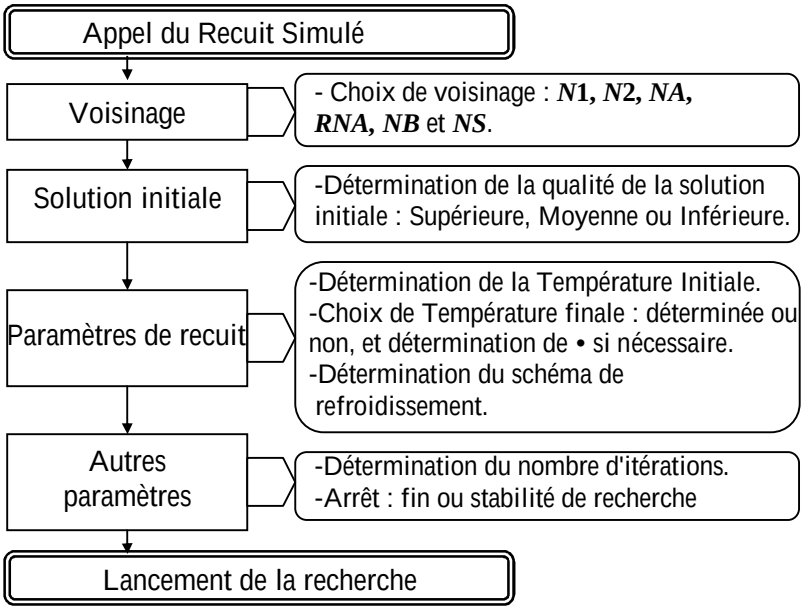


Figure 4.22 : Plan général de l'utilisation du Recuit Simulé dans Meta-JSS.

5.3. Présentation des résultats

Après le lancement de la recherche avec une méthode, la courbe d'évolution de la résolution s'affiche durant le processus de recherche (figure 4.23). Cette courbe indique à chaque moment (itération) le makespan de la solution actuelle.

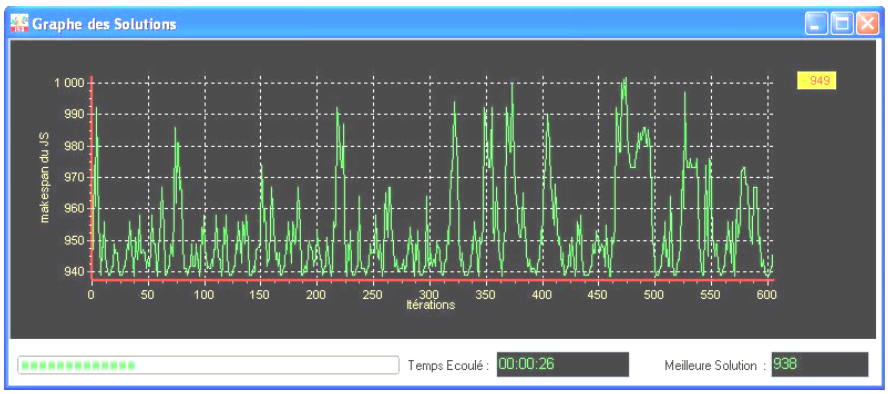


Figure 4.23 : La courbe de l'évolution de la résolution avec la Recherche Tabou.

Quand la recherche termine, le logiciel affiche les résultats de résolution sur des graphiques et des tableaux (voir figure 4.16).

Le diagramme de Gantt est le graphique le plus représentatif de la solution. Deux types de Gantt sont affichés pour chaque solution, Gantt-machine et Gantt-tâche.

La figure 4.24 présente le diagramme de Gantt-machine d'une solution optimale du benchmark mt10.

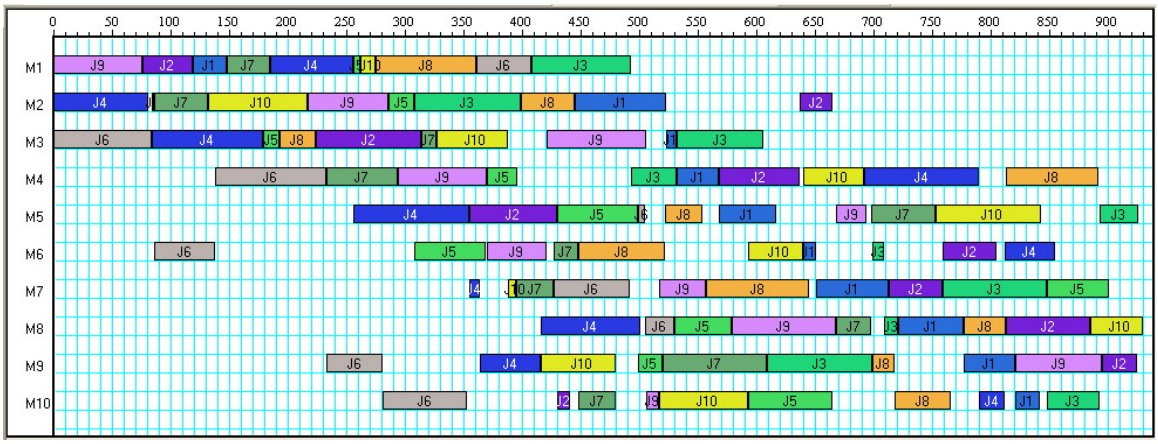


Figure 4.24 : Diagramme de Gantt représentant la solution optimale de mt10.

6. Conclusion

Au travers ce chapitre, nous avons présenté et expliqué l'implémentation des différentes Métaheuristiques retenues, en les appliquant à la résolution du problème de Job Shop. Pour chacune, plusieurs choix ont été implantés dans le but d'explorer la fiabilité des différentes stratégies et l'impact des différents opérateurs et paramètres sur l'efficacité des méthodes. Notre application ne porte que sur la version standard de chaque méthode. Ceci exclut plusieurs approches plus avancées dont la recherche ne cesse de les proposer. Le résultat de l'implémentation informatique de nos travaux est un outil (Meta-JSS) qui servira à expérimenter les différentes méthodes étudiées dans le chapitre suivant.

CHAPITRE V

Expérimentations, résultats et discussion

Résumé : *Dans le but d'explorer les performances des Métaheuristiques implémentées, et surtout de leurs différents ingrédients, opérateurs et paramètres, nous présentons dans ce chapitre, les résultats obtenus par plusieurs séries d'expérimentations. Ces expérimentations portent sur un ensemble de benchmarks couramment utilisés dans les recherches. L'objectif est de dégager des résultats importants concernant la résolution du problème de Job Shop à l'aide des Métaheuristiques standards.*

CHAPITRE V

Expérimentations, résultats et discussion

1. Introduction

L'objectif de ce présent chapitre est de donner une synthèse des résultats obtenus par application des trois Métaheuristiques à la résolution d'un ensemble de problèmes tests "benchmarks" de Job Shop simple, afin d'évaluer les performances des différents choix implémentés de ces Métaheuristiques. Contrairement à certaines études, nous avons choisi d'évaluer les différents opérateurs et de comparer leurs performances au sein des méthodes, et pas sur des opérateurs isolés. En effet, il ne s'agit pas de comparaisons formelles, fondées sur une analyse statistique rigoureuse. Cependant, elles suffisent à explorer l'efficacité des différentes stratégies, opérateurs et paramètres de ces méthodes, et à valider notre implémentation informatique.

Ainsi, dans la deuxième section, nous présentons des résultats généraux exprimant les performances des Métaheuristiques employées, en s'intéressant à leur aptitude générale de résolution et aux meilleurs résultats trouvés. Dans les trois sections suivantes, l'expérimentation intéresse les différentes composantes de chaque méthode. L'estimation de l'influence des différents paramètres et ingrédients sur le processus de résolution est faite via l'analyse des résultats obtenus. Notons que dans tous les essais, l'objectif est la minimisation du makespan, et pour chaque cas, les expérimentations sont réalisées dans les mêmes circonstances techniques.

2. Résultats généraux

Dans le but d'explorer les performances relatives de chacune des Métaheuristiques en résolution du problème de Job Shop, une série d'essais a été effectuée sur 25 problèmes de tailles et de complexité diverses. Pour chaque problème, cinq expériences ont été réalisées, et nous prenons leur moyenne arithmétique. Le tableau 5.1 et le graphique de la figure 5.1 rapportent les

résultats obtenus. Notons que le paramétrage des méthodes est pris par défaut. En plus des trois Métaheuristiques étudiées, s'ajoute à l'analyse, la génération de solutions aléatoires.

problème	taille	Opt.	AG	RT	RS	Solution Aléatoire
abz5	10x10	1234	1239.2	1240.4	1241.0	1842.6
abz6	10x10	943	945.0	944.2	944.8	1348.4
abz7	20x15	655	678.4	679.2	680.0	980.6
abz8	20x15	638	692.4	688.8	688.2	960.0
abz9	20x15	661	699.6	670.0	670.4	975.2
ft06	6x6	55	55.0	55.0	55.0	72.8
ft10	10x10	930	937.8	938.4	938.0	1340.4
ft20	20x5	1165	1167.8	1170.2	1168.0	1479.0
la11	20x5	1222	1222.0	1222.0	1222.8	1445.6
la12	20x5	1039	1039.0	1039.2	1039.0	1270.8
la13	20x5	1150	1150.0	1150.0	1150.0	1358.4
la14	20x5	1292	1292.0	1292.0	1292.0	1501.6
la15	20x5	1207	1207.0	1207.0	1207.0	1497.2
la16	10x10	945	948.8	952.4	947.6	1305.6
la17	10x10	784	787.8	786.0	785.2	1039.0
la18	10x10	848	861.0	858.2	855.2	1276.8
la19	10x10	842	851.2	846.6	847.2	1245.4
la20	10x10	902	904.0	902.8	907.4	1330.2
orb01	10x10	1059	1064.2	1062.0	1066.4	1571.6
orb02	10x10	888	898.4	892.4	894.0	1196.0
orb03	10x10	1005	1014.6	1008.8	1016.4	1711.2
orb04	10x10	1005	1020.2	1012.0	1014.6	1728.8
orb05	10x10	887	888.8	892.8	892.2	1156.4
orb06	10x10	1010	1030.2	1016.6	1013.8	1440.0
orb08	10x10	899	922.6	914.4	920.8	1294.2
	La moyenne		940,68	938,256	937,28	1294,712

Tableau 5.1 : Résultats généraux obtenus par application des trois Métaheuristiques.

Opt. : solution optimale, AG : Algorithmes Génétiques,
RT : Recherche Tabou, RS : Recuit Simulé.

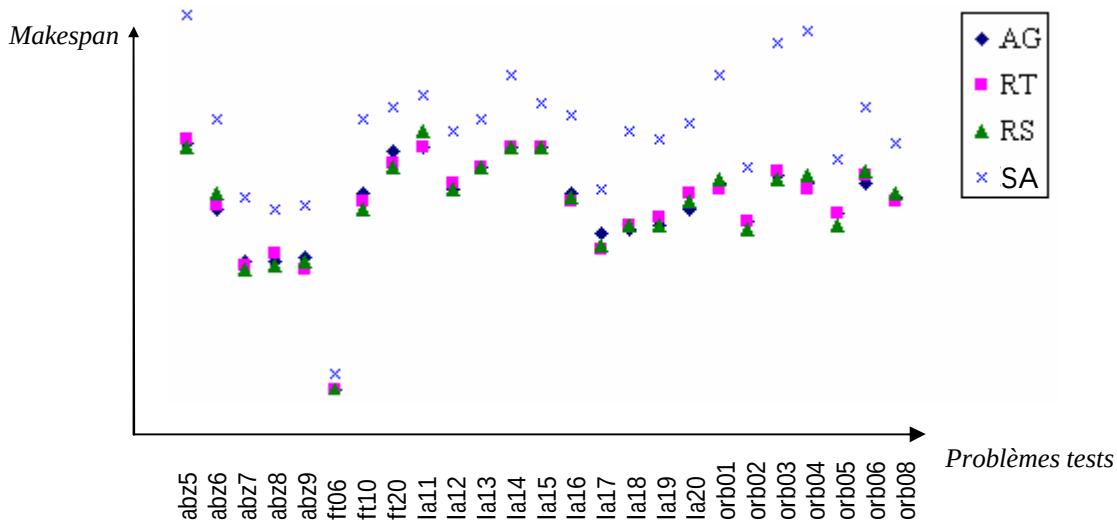


Figure 5.1 : Graphique des résultats de résolution de 25 problèmes par les trois Métaheuristiques.

Il peut être intéressant d'observer les relations de dominance empirique afin de déduire un classement des méthodes. On dit qu'une Métaheuristique domine empiriquement une autre, si pour tous les problèmes tests, les solutions données par la première sont égales ou meilleures que celles obtenues par la deuxième, avec au moins une solution strictement meilleure [Duv, 00].

D'après ces résultats, apparaît l'intérêt majeur des Métaheuristiques en les comparant avec la solution générée aléatoirement. Cette dernière reste toujours très loin de l'optimal, alors que les autres méthodes produisent de très bonnes solutions, en terme du makespan. On observe que la génération aléatoire est dominée empiriquement par les trois Métaheuristiques. Et aucune autre relation de dominance ne peut être observée. Le premier résultat à souligner donc est le grand intérêt des Métaheuristiques dans la résolution du problème de Job Shop.

L'autre résultat à annoncer, est qu'il semble difficile, étant donné la faible différence entre les solutions générées, de déterminer un classement certain entre les trois méthodes. Cette même constatation a été soulignée par plusieurs d'autres études [Duv, 00]. Toutefois les résultats obtenus, et dans la limite des tests effectués, nous permettent d'établir grossièrement le classement décroissant suivant des méthodes selon la qualité de résolution : RS > RT > AG.

Ces résultats permettent aussi de valider notre implémentation des différentes méthodes. En effet, pour les problèmes faciles, les trois méthodes ont permis de trouver l'optimum dans la plupart des expériences. Pour des problèmes reconnus comme difficiles, les méthodes arrivent à des solutions très proches de l'optimum dans la majorité des cas, et même elles produisent, dans certains cas des solutions optimales.

Le temps de résolution est un autre facteur qui mérite d'être examiné. Nous ne nous intéressons pas ici à comparer les méthodes en terme du temps de résolution, mais uniquement à valider notre implémentation en ce qui concerne ce facteur. Le temps de trouver la solution optimale de six problèmes de tailles différentes, par les trois méthodes est rapporté au tableau 5.2

Problème	**Taille**	C_{max}	**AG**		**RT**		**RS**	
			générations	temps	itérations	temps	itérations	temps
ft06	6x6	55	1000	00:00:19	6000	00:00:11	6000	00:00:09
la01	10x5	666	500	00:00:13	3000	00:00:19	3000	00:00:22
abz6	10x10	943	3000	00:00:34	60000	00:00:25	60000	00:00:31
ft20	20x5	1165	2000	00:00:54	80000	00:00:29	80000	00:00:37
la15	20x5	1207	500	00:00:21	5000	00:00:20	6000	00:00:24
la11	20x5	1222	500	00:00:20	5000	00:00:21	6000	00:00:24

Tableau 5.2 : Le temps pris par les trois méthodes pour trouver l'optimale de six problèmes.

D'après les résultats du tableau 5.2, nous pouvons conclure que le facteur de temps n'est plus un problème pour notre application. Effectivement, la durée de résolution des différentes

instances est de l'ordre de quelques secondes à quelques dizaines de minutes pour une recherche de plusieurs millions d'itérations. Par conséquent, le temps de résolution marque un point de force pour l'application et témoigne de l'efficacité de l'implémentation proposée de ces Métaheuristiques.

3. Résultats de l'application des Algorithmes Génétiques

L'efficacité des Algorithmes Génétiques est fonction des différents éléments et paramètres de la méthode qui se résument en : codage, opérateurs génétiques et paramètres de la méthode. Afin d'investiguer l'influence de ces différents éléments sur le processus de recherche, nous avons effectué une série d'expérimentations sur chaque élément, visant ainsi à explorer comparativement l'importance des différentes stratégies et approches implantées.

3.1. Le codage

Le codage est sans doute, l'élément ayant le rôle le plus important dans l'efficacité de l'Algorithme Génétique. Le tableau 5.3 et le graphique de la figure 5.2 présentent les résultats obtenus par l'emploi des quatre stratégies de codage (voir le chapitre 4. § 2.1). Pour chaque problème et codage, la moyenne arithmétique de cinq essais réalisés est donnée.

Pour toutes les expérimentations, les autres paramètres de la méthode sont fixés à 1000 individus, 4000 générations, le reste est pris par défaut.

problème	taille	Opt.	O-based	J-based	PL-based	PR-based
abz5	10x10	1234	1238.2	1411.0	1243.4	1246.6
abz6	10x10	943	946.8	1047.2	948.6	944.0
abz7	20x15	655	679.6	717.8	681.4	680.4
abz8	20x15	638	694.4	822.4	699.0	698.2
abz9	20x15	661	699.0	798.8	706.2	701.4
ft06	6x6	55	55.0	61.0	55.0	55.0
ft10	10x10	930	938.4	1046.0	937.6	940.2
ft20	20x5	1165	1168.4	1198.2	1170.8	1171.6
la16	10x10	945	974.8	1126.6	979.4	975.0
la17	10x10	784	784.0	868.0	785.0	784.2
la18	10x10	848	856.6	960.2	857.4	858.8
la19	10x10	842	844.8	954.0	848.0	845.6
la20	10x10	902	911.2	980.0	907.6	911.4
orb01	10x10	1059	1065.4	1168.8	1070.6	1068.2
orb02	10x10	888	889.2	970.4	888.8	890.0
orb03	10x10	1005	1006.8	1140.8	1011.6	1017.4
orb04	10x10	1005	1032.0	1186.4	1023.0	1032.8
orb05	10x10	887	888.8	975.2	893.0	894.4
swv1	20x10	1407	1442.4	1558.6	1458.2	1450.8
swv2	20x10	1475	1510.6	1655.0	1526.8	1516.2
	La moyenne		931,32	1032,32	934,57	934,11

Tableau 5.3 : Résultats obtenus par application des quatre stratégies de codage.

La figure 5.2 nous montre sous forme graphique les relations d'ordre entre les codages adoptés.

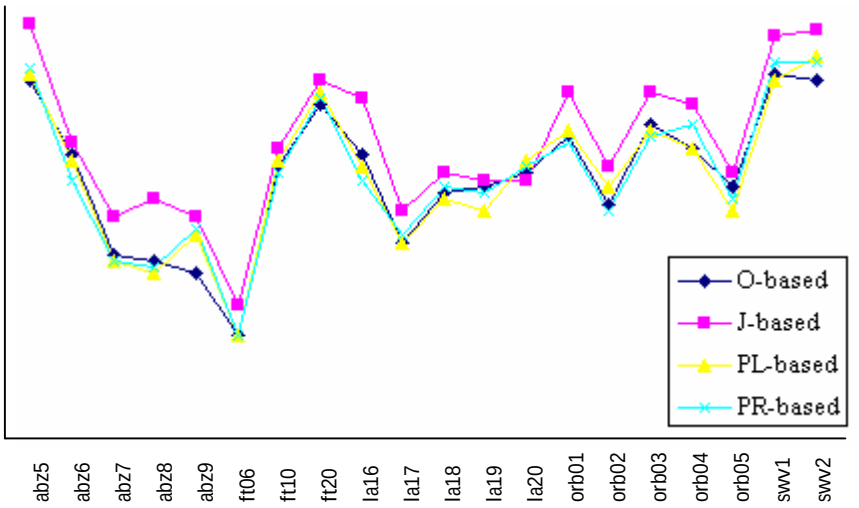


Figure 5.2 : Graphique des résultats obtenus par l'utilisation des quatre types de codage.

Ces résultats nous permettent de déduire les deux constatations suivantes :

- Le codage basé sur les tâches (job based representation), est nettement surclassé par les autres codages. Ce fait s'explique par la définition de ce codage lui même. Puisque, avec ce codage, l'ordonnancement d'une opération est lié directement à sa tâche, ceci limitera sa chance à situer dans plusieurs emplacements de l'ordonnancement. Par conséquent, le processus de recherche n'explore qu'une partie très réduite de l'espace d'états.
- Pour les autres codages, il n'existe pas de différences remarquables en terme de qualité des solutions trouvées exprimée par le makespan. Pourtant, les résultats obtenus dans le cas du codage basé sur les opérations sont légèrement meilleurs que celles des deux autres codages.

3.2. Les opérateurs génétiques

Les résultats obtenus par les Algorithmes Génétiques sont largement influencés par les opérateurs génétiques utilisés : croisement, mutation et sélection.

3.2.1. Le croisement

Dans le but de mettre en évidence d'éventuelles différences pouvant exister entre les différentes stratégies de croisement employées, une série d'expériences a été effectuée. Pour chaque cas, la moyenne arithmétique de cinq expériences est calculée. Les résultats sont donnés au tableau 5.4. Les problèmes tests sont choisis difficiles. Le nombre de générations et la taille de la population sont peu élevés. Tout cela, dans le but d'explorer clairement l'efficacité des opérateurs de croisement.

Représentation basée sur les opérations				
Problème	**Taille**	**JOX**	**PPX**	**GOX**
abz7	20x15	679.2	682.4	680.0
abz8	20x15	690.8	692.8	688.6
abz9	20x15	705.0	711.6	708.2
la36	15x15	1296.6	1301.2	1294.8
la37	15x15	1425.4	1428.8	1433.0
la38	15x15	1228.6	1221.0	1220.8
la39	15x15	1259.8	1264.4	1264.2
la40	15x15	1256.6	1256.2	1251.6
swv1	20x10	1441.2	1459.0	1442.4
swv2	20x10	1502.0	1494.4	1513.6

Représentation basée sur les listes de préférences				
Problème	**Taille**	**JOX**	**LOX**	**SXX**
abz7	20x15	679.2	680.0	679.8
abz8	20x15	694.6	691.6	689.4
abz9	20x15	703.2	709.0	721.8
la36	15x15	1298.2	1300.6	1299.8
la37	15x15	1431.6	1421.4	1429.4
la38	15x15	1221.0	1230.2	1229.0
la39	15x15	1254.2	1258.4	1262.0
la40	15x15	1255.4	1261.0	1251.6
swv1	20x10	1451.2	1442.4	1446.8
swv2	20x10	1500.4	1505.8	1499.6

Représentation basée sur les tâches			
Problème	**Taille**	**JOX**	**LOX**
abz7	20x15	715.2	721.8
abz8	20x15	825.0	828.4
abz9	20x15	796.6	991.2
la36	15x15	1491.8	1498.6
la37	15x15	1606.2	1592.8
la38	15x15	1443.6	1433.0
la39	15x15	1498.4	1506.2
la40	15x15	1455.4	1459.0
swv1	20x10	1556.0	1556.8
swv2	20x10	1650.2	1651.6

Tableau 5.4 : Résultats obtenus par application des différentes stratégies de croisement.

Au travers les résultats du tableau 5.4 représentés par les trois graphiques de la figure 5.3, nous pouvons déduire que l'impact du codage employé est plus important que celui du croisement. En effet, dans toutes ces expériences, le makespan ne change pas de façon significative d'un type de croisement à l'autre pour les trois codages testés. Donc, il n'y a pas de différences strictes entre ces types de codage.

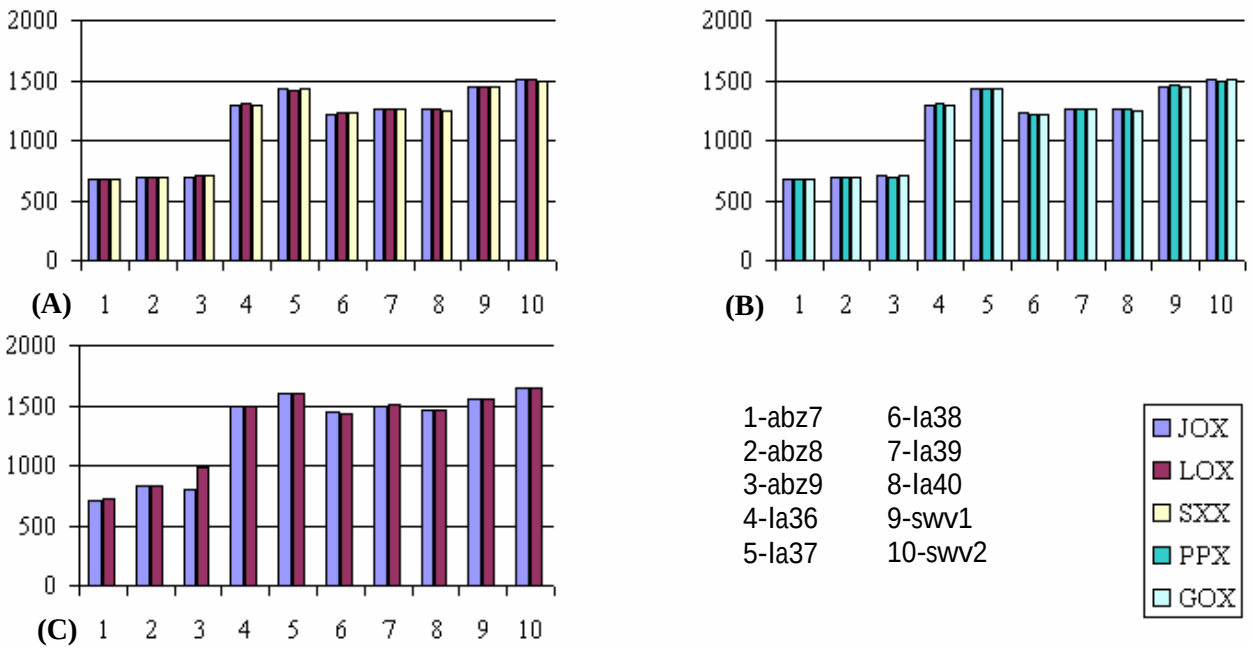
Même, si de légères différences peuvent être remarquées entre les types de croisement, en faveur :

- De JOX (4/10) et GOX (4/10) sur PPX (2/10) pour le codage basé sur les opérations.

- De JOX (5/10) sur SXX (3/10) et LOX (2/10) pour le codage basé sur les listes de préférences.
- Et de JOX (7/10) sur LOX (3/10) pour le codage à base de tâches.

Néanmoins, ces différences ne sont pas en mesure de classer les croisements appliqués.

Ces remarques conduisent à dire que les opérateurs basés sur l'héritage de l'ordre tels que : JOX, GOX et SXX, sont mieux adaptés que ceux basés sur l'héritage d'adjacence comme LOX et PPX. Ce résultat même a été souligné par [Duv, 00].



(A) Codage basé sur les listes de préférences
(B) Codage basé sur les opérations
(C) Codage basé sur les tâches

Figure. 5.3 : Représentation graphique des résultats du tableau 5.4.

Pour explorer l'effet du taux de croisement, nous avons effectué des essais avec des codages différents, en faisant varier le taux de croisement de 5% (taux faible) à 95% (la majorité de la population est concernée).

Le tableau 5.5 donne pour chaque cas le nombre de générations où le meilleur makespan est apparaît, ce makespan est égal à 3000 pour le codage à base de tâche, et à 2823 pour les autres codages. L'ensemble des essais prend sur un seul problème facile : swv20, constitué de 500 opérations, et la résolution se fait une seule fois. Ceci permet de découvrir aisément l'effet de variation du taux de croisement. Tous les autres paramètres de la méthode sont fixés.

Représentation basée sur les opérations					
croisement	5%	20%	50%	75%	95%
JOX	88	13	28	9	6
PPX	69	21	12	13	8
GOX	104	18	13	10	11
Représentation basée sur les tâches					
croisement	5%	20%	50%	75%	95%
JOX	651	324	197	112	86
LOX	701	416	110	201	57
Représentation basée sur les règles de priorité					
croisement	5%	20%	50%	75%	95%
PPX	55	18	12	26	8
Représentation basée sur les listes de préférence					
croisement	5%	20%	50%	75%	95%
JOX	57	41	21	17	10
LOX	100	38	18	15	11
SXX	73	29	19	24	7

Tableau 5.5 : Influence du taux de croisement sur le processus de recherche.

La représentation graphique est identique pour les quatre codages, nous donnons alors celle du codage basé sur les opérations à la figure 5.4.

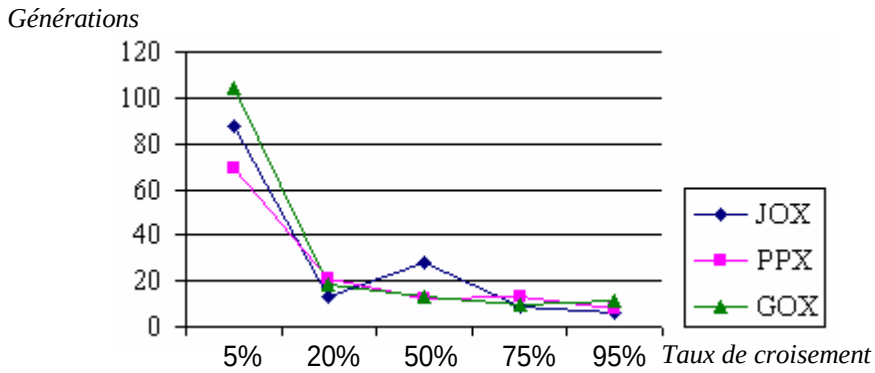


Figure 5.4 : Représentation graphique de l'influence du taux de croisement sur la recherche.

D'après ces résultats concernant le taux de croisement, deux remarques peuvent être faites :

- L'influence du taux de croisement sur le makespan n'est plus importante pour tous les problèmes testés, à condition que le nombre de générations soit suffisamment élevé. Un taux de croisement faible peut causer une convergence prématurée de la recherche si le nombre de générations est trop réduit.
- L'effet d'un taux de croisement élevé est l'accélération de la convergence de l'algorithme, au moins pour notre échantillon de test. Ainsi pour un taux de 95% la recherche nécessite seulement entre 20% à 30% de générations comparativement à une recherche avec un taux de 5% pour donner la même solution.

3.2.2. La mutation

Pour la mutation, deux types sont implantés : mutation de position et mutation de tâches. Afin de les explorer, nous avons testé trois problèmes difficiles (abz7, abz8, abz9) avec six taux différents : 5%, 15%, 30%, 50%, 75% et 90%. Le nombre de générations (500) et la taille de population (500) ne sont pas trop élevés. Le codage choisi est basé sur les opérations, et le croisement est GOX. Tous les autres paramètres sont pris par défaut. Les résultats sont donnés au tableau 5.6.

	Mutation	**5%**	**15%**	**30%**	**50%**	**75%**	**90%**
abz7	De position	717	714	711	724	699	724
	De tâche	722	712	723	733	729	731
abz8	De position	735	722	736	725	719	738
	De tâche	722	716	734	726	724	742
abz9	De position	741	726	712	742	729	720
	De tâche	708	718	724	719	738	721

Tableau 5.6 : Résultats obtenus par application de deux types de mutation avec taux différents.

Selon ces expériences, le type de mutation semble n'avoir pas d'effet sur le processus de recherche au moins pour les deux types étudiés. L'impact du type de mutation est moins important que son taux. Pour des taux faibles, elle participe à garder un certain degré de diversité dans la population, empêchant ainsi la convergence prématurée de l'algorithme. Alors que, pour des taux élevés, elle risque d'altérer les meilleurs individus, en les rendant plus mauvais. Mais nos expériences ici ne démontrent pas cet effet, ceci revient à deux raisons : premièrement la stratégie élitiste de la sélection qui garantit la conservation des bons éléments de la population. Deuxièmement, la difficulté des instances testées, qui influence sur la recherche nécessitant en réalité des paramètres plus élevés.

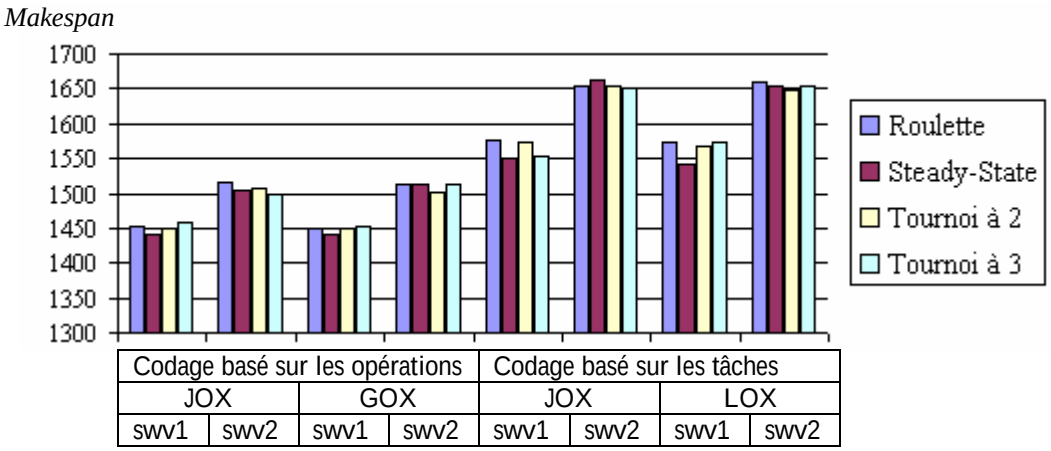
3.2.3. La sélection

Pour la sélection, nous avons également effectué un ensemble d'essais sur deux problèmes difficiles swv10 et swv11, en vue d'explorer l'impact des différents types de sélection implémentés sur la résolution par la méthode. Deux types de codage sont employés : le codage à base d'opérations avec les croisement JOX et GOX, et le codage à base de tâches avec les croisement JOX et LOX.

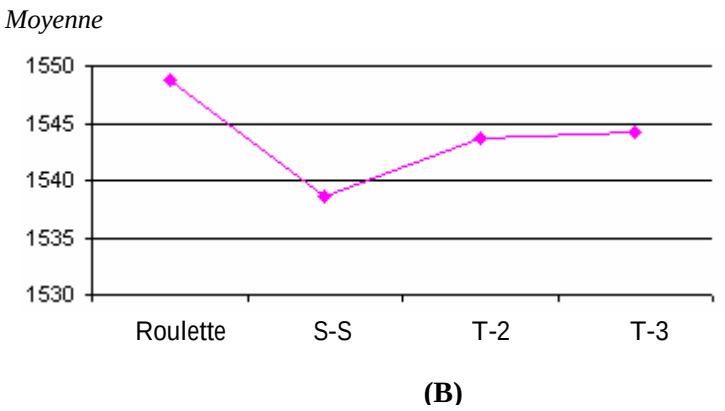
Nous rapportons au tableau 5.7 la moyenne arithmétique des cinq makespans trouvés pour chaque cas.

	Codage basé sur les opérations				**Codage basé sur les tâches**			
	JOX		GOX		JOX		LOX	
Sélection	swv1	swv2	swv1	swv2	swv1	swv2	swv1	swv2
Roulette	1451.2	1515.0	1449.6	1512.8	1575.6	1653.2	1573.8	1659.0
S-S	1441.6	1504.6	1440.2	1514.2	1550.2	1662.2	1543.0	1652.6
T-2	1449.8	1506.8	1450.8	1501.0	1573.4	1652.8	1567.4	1646.8
T-3	1459.0	1498.2	1451.4	1514.2	1554.0	1650.6	1572.8	1654.0

Tableau 5.7 : Résultats obtenus par application des quatre types de sélection.
Roulette, Steady State (S-S), Tournoi à 2 (T-2) et Tournoi à 3 (T-3).



(A)



(B)

(A) : Histogrammes des données du tableau 5.7.

(B) : Les moyennes générales des résultats obtenus pour chaque type de sélection.

Figure 5.5 : Représentation graphique de l'influence du type de sélection sur la recherche.

La remarque la plus importante à faire ici est que les différences existant entre les résultats obtenus par les différentes stratégies de sélection sont faibles, et ne sont pas en mesure d'établir un classement rigoureux de ces quatre stratégies. Effectivement, toutes ces stratégies, même différentes dans leurs algorithmes, sont basées sur le même principe général, qui vise la favorisation des individus ayant le meilleur score d'adaptation en les faisant se produire.

Toutefois, si nous prenons ces petites différences en considération, nous pouvons constater, au moins pour nos expériences (figure 5.5 (B)), que la stratégie de Steady-State donne les meilleurs résultats, alors que la stratégie de roulette semble être moins fiable. Ceci est dû peut être au nombre d'itérations réduit favorisant ainsi la stratégie Steady-State qui converge plus rapidement.

3.3. Les paramètres de dimensionnement de l'Algorithme Génétique

La taille de la population et le nombre de générations sont deux autres paramètres importants dans le processus de recherche par l'Algorithme Génétique, nous analysons dans ce qui suit leur impact via deux séries d'expériences.

3.3.1. La taille de la population

La première série d'essais est réalisée sur deux problèmes moyennement difficiles (la39 et abz5) avec la taille de population varie de 100 à 20000 et le nombre de générations est fixé à 100.

Les essais portent sur trois stratégies différentes :

- Codage basé sur les opérations avec croisement GOX
- Codage basé sur les tâches avec croisement JOX
- Codage basé sur les listes de préférences avec croisement SXX

Les résultats sont donnés au tableau 5.8.

Codage basé sur les opérations - croisement GOX										
problème	100	200	500	1000	3000	5000	7500	10000	15000	20000
la39	1467	1398	1389	1283	1251	1274	1275	1271	1259	1262
abz5	1328	1313	1290	1252	1249	1250	1242	1245	1244	1242
Codage basé sur les tâches - croisement JOX										
problème	100	200	500	1000	3000	5000	7500	10000	15000	20000
la39	1590	1585	1527	1493	1510	1462	1488	1462	1507	1498
abz5	1450	1450	1437	1424	1400	1408	1411	1411	1375	1375
Codage basé sur les listes de préférences - croisement SXX										
problème	100	200	500	1000	3000	5000	7500	10000	15000	20000
la39	1474	1411	1356	1292	1260	1272	1280	1273	1260	1266
abz5	1345	1326	1281	1250	1249	1244	1250	1249	1250	1245

Tableau 5.8 : Résultats obtenus avec tailles différentes de la population.

En traçant la courbe de la moyenne arithmétique de chaque problème, nous obtenons le graphique de la figure 5.6.

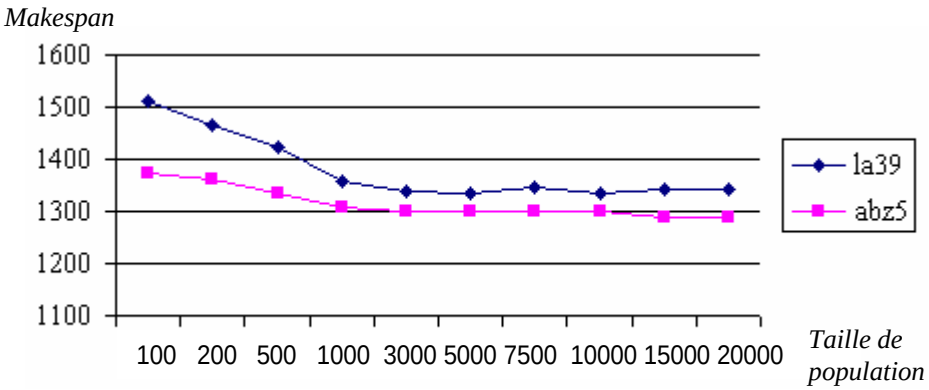


Figure 5.6 : Effet de la taille de population sur la recherche par l'Algorithme Génétique.

Sur la courbe de la figure 5.6, nous remarquons l'existence de deux phases :

- Une première phase objective l'effet positif de l'augmentation de la taille de la population sur la résolution, l'ajout de plus d'individus à la population favorise l'apparition d'une plus bonne solution.
- Après une certaine limite, commence la deuxième phase ; l'augmentation de la taille de la population n'influence plus sur la résolution.

3.3.2. Le nombre de générations

Le processus de recherche par les Algorithmes Génétiques pour tous les problèmes testé suit une allure générale illustrée par la courbe de la figure 5.7. Cette courbe montre l'existence de deux temps de recherche. Un premier temps, se caractérise par une décroissance rapide du makespan minimal des générations successives, ce qui signifie que la recherche est efficace. A un deuxième temps, la recherche commence à se stabiliser, et l'amélioration de la solution se fait très lentement jusqu'à stabilisation de la recherche. L'évolution durant les dernières générations devient inutile.

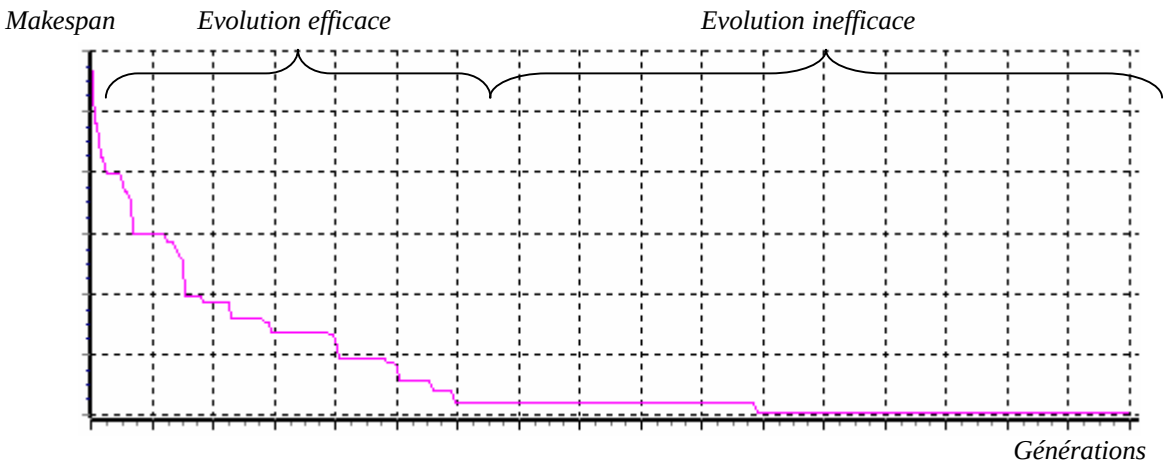


Figure 5.7 : Amélioration de la solution durant l'évolution des générations de l'Algorithme Génétique.

D'après les résultats obtenus par les séries d'expériences réalisées sur les différents opérateurs et paramètres des Algorithmes Génétiques, nous constatons que cette méthode est très intéressante pour la résolution du problème de Job Shop en raison des bonnes solutions retournées. Nous pouvons constater également que l'impact du codage est plus important que les autres opérateurs. Les paramètres de la méthode, jouent aussi un rôle important dans le processus de résolution. Ce qui fait qu'un bon choix de ces paramètres est une condition primordiale de la réussite de la recherche.

4. Résultats de l'application de la Recherche Tabou

L'efficacité de la méthode Tabou, est déterminée par ses deux composants principaux : la fonction de voisinage et la mémoire Tabou. Par conséquent, l'exploration de la méthode passe par l'expérimentation de ces éléments.

4.1. Les fonctions de voisinage

Notre implémentation de la Recherche Tabou utilise six fonctions de voisinage qui sont : *N*1, *N*2, *NA*, *RNA*, *NB* et *NS*. La comparaison des résultats obtenus par ces voisinages se fait à l'aide de 18 problèmes tests de tailles diverses et de difficulté moyenne à forte. Pour chaque voisinage, chacun des 18 problèmes a été résolu cinq fois dont la moyenne arithmétique est calculée.

Le tableau 5.9 rapporte les résultats obtenus concernant les moyennes arithmétiques. La figure 5.8 donne la représentation graphique de leurs moyennes globales.

problème	***N*1**	***N*2**	***NA***	***RNA***	***NB***	***NS***
abz5	1239.4	1236.2	1240.0	1238.6	1238.0	1236.4
abz6	943.8	946.6	944.8	946.2	948.2	943.0
abz7	678.0	679.2	676.4	683.8	680.0	675.6
abz8	690.6	692.8	693.2	689.4	691.4	688.2
abz9	704.2	701.8	706.4	711.2	702.0	707.4
ft10	938.0	938.0	936.8	939.0	937.6	938.4
ft20	1167.0	1168.2	1170.8	1167.6	1173.4	1171.4
la36	1294.6	1304.8	1305.2	1299.8	1305.0	1297.6
la37	1438.6	1433.6	1429.8	1440.2	1437.4	1429.2
la38	1208.0	1213.4	1210.8	1211.6	1207.4	1208.4
orb01	1070.2	1067.8	1070.4	1074.0	1068.6	1069.0
orb02	890.2	891.6	889.8	889.0	892.2	890.0
orb03	1007.4	1009.4	1009.2	1010.4	1006.6	1007.2
orb04	1024.8	1028.6	1035.0	1033.8	1019.4	1025.8
orb05	896.6	894.2	892.8	895.4	896.0	893.6
Moyenne	1012,76	1013,74667	1014,09333	1015,33333	1013,54667	1012,08

Tableau 5.9 : Résultats obtenus par Recherche Tabou avec différentes fonctions de voisinage.

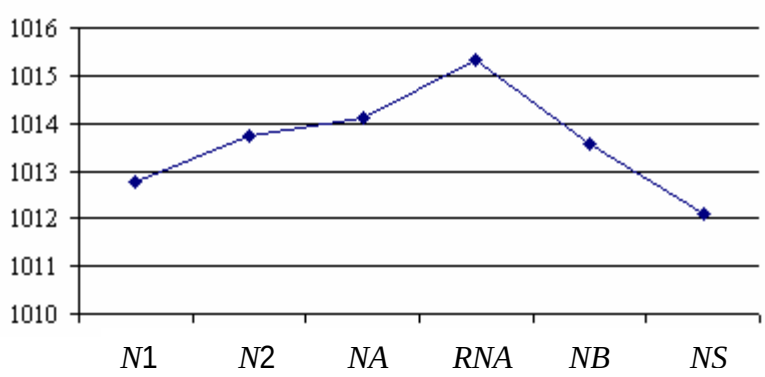


Figure 5.8 : Les moyennes des résultats obtenus par la Recherche Tabou avec différents voisinages.

D'après ces résultats, nous pouvons constater qu'il n'y a pas de prédominance empirique d'un voisinage sur les autres. Il n'existe pas de différences significatives entre ces différents voisinages. Pour la moyenne de l'ensemble des problèmes, de légères différences sont à remarquer (la plus grande différence ne dépasse pas 0.4%) entre les six voisinages. Ainsi, la classification suivante du meilleur au mauvais peut être établie $NS > N1 > NB > N2 > NA > RNA$. Les différences essentielles entre ces voisinages qui ne sont pas mises en évidence ici, et qui sont étudiées par plusieurs auteurs [Wat & al., 03] [Jai, 98], concernent surtout le coût : le temps d'exécution et le nombre d'itérations nécessaires pour trouver la même solution.

4.2. La qualité de la solution initiale

Le tableau 5.10 résume les résultats obtenus par la résolution une seule fois de dix problèmes, avec initialisations différentes. Tous les tests sont effectués dans les mêmes conditions et avec les mêmes paramètres de la méthode.

Qualité	la36	la37	la38	la39	orb01	orb02	orb03	orb04	orb05	orb06
Inférieure	1315	1439	1215	1251	1071	903	1038	1087	915	1016
Moyenne	1311	1458	1209	1241	1073	894	1010	1029	898	1013
Supérieure	1305	1434	1211	1249	1070	894	1008	1032	897	1010

Tableau 5.10 : Résultats obtenus par la Recherche Tabou avec trois niveaux d'initialisation.

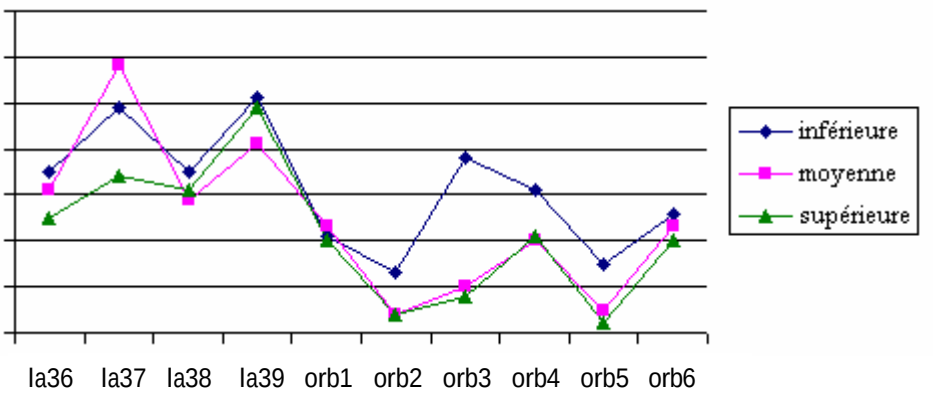


Figure 5.9 : Présentation graphique des résultats du tableau 5.10.

D'après les résultats émergés de cette série d'expériences, nous remarquons que la qualité de la solution initiale peut influencer sur la recherche. Ainsi, une mauvaise solution de départ peut entraver le processus de recherche à explorer les bons endroits de l'espace d'états. Effectivement, dans nos expériences, l'initialisation de qualité supérieure nous a fournit la meilleure solution dans la majorité des cas (7/10), contre (3/10) pour l'initialisation de qualité moyenne et (0/10) pour l'initialisation de qualité inférieure.

4.3. L'implantation de la mémoire Tabou

Afin de découvrir l'effet de la taille de la mémoire Tabou sur la résolution, nous avons effectué une série d'essais en faisant varier à chaque fois la taille de la mémoire. Nous avons utilisé pour ce faire :

- Une mémoire dynamique, et
- Une mémoire fixe avec des tailles différentes : l, l *div* 2, l×2 et l×4

(l est la racine carrée arrondie de n×m pour un problème n-tâches m-machines et *div* désigne la division naturelle).

Pour avoir des résultats clairs, la résolution se fait avec un petit nombre d'itérations (1000). Le tableau 5.11 présente les résultats trouvés.

	***l div* 2**	***l***	***l*×2**	***l*×4**	**dynamique**
la36	1315	1321	1303	1301	1307
la37	1449	1446	1446	1451	1446
la38	1219	1208	1216	1223	1210
la39	1250	1244	1252	1250	1247
la40	1264	1242	1253	1252	1243
orb01	1270	1265	1262	1276	1265
orb02	894	897	902	899	896
orb03	1005	1009	1007	1013	1012
orb04	1024	1019	1039	1028	1026
orb05	891	891	902	895	897

Tableau 5.11 : Résultats obtenus avec différentes tailles de la mémoire des taboues.

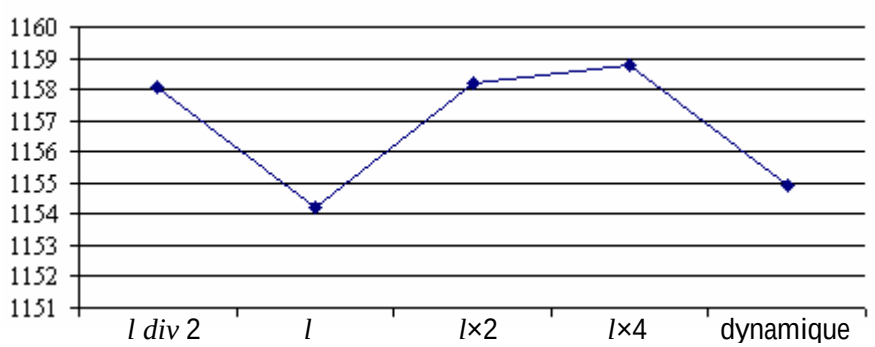


Figure 5.10 : Représentation graphique des moyennes générales des résultats du tableau 5.11.

Les résultats obtenus par l'emploi de mémoires de tailles variables permettent de constater que :

- Les résultats donnés par une mémoire dynamique sont les meilleurs en moyenne. Ceci s'explique par le fait que ce choix élimine automatiquement le cyclage qui constitue un obstacle sérieux à la méthode.
- L'emploi d'une mémoire trop courte ou trop longue donne pour plusieurs problèmes de mauvaises solutions. Une taille trop faible risque de favoriser le cyclage, alors qu'une taille trop grande risque de restreindre le voisinage, en ne laissant pas ainsi beaucoup de chances à la méthode pour sortir des minima locaux.

4.4. Le nombre d'itérations

Avant de discuter l'effet du nombre d'itérations sur la recherche, nous montrerons d'abord "l'allure" générale du processus de recherche. Ceci est visualisé via la figure 5.11. Puisque la méthode Tabou accepte même les dégradations, la courbe se présente sous forme d'oscillations entre des solutions de qualités diverses.

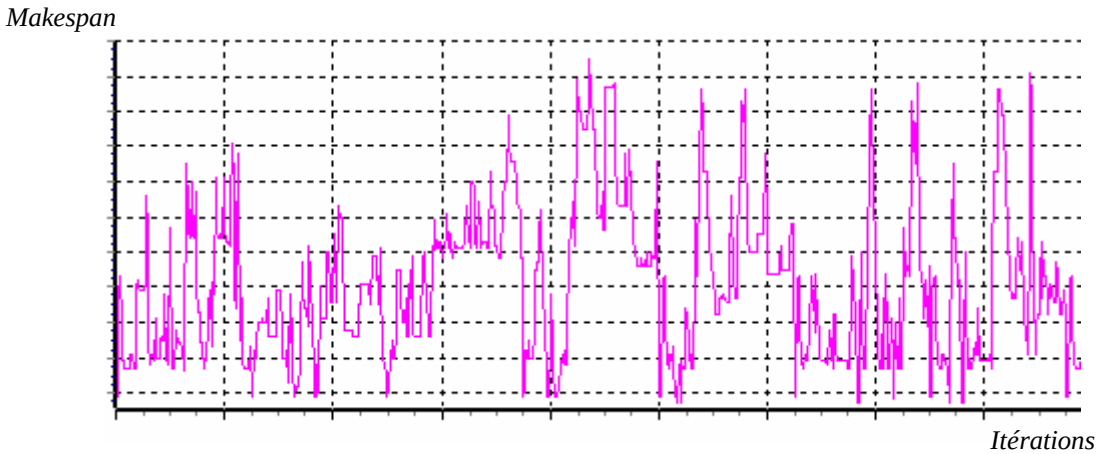


Figure 5.11 : La courbe de l'évolution de la résolution avec la Recherche Tabou.

Le tableau 5.12 rapporte les résultats issus d'une série d'expériences réalisées avec un nombre croissant d'itérations. La méthode utilisée est basée sur le voisinage *NS*, avec initialisation de qualité inférieure.

Itérations	500	1000	5000	10000	20000	50000
ft10	975	942	938	938	938	936
orb3	1026	1011	1007	1005	1005	1005
orb5	914	902	894	906	896	891

Tableau 5.12 : Résultats obtenus avec nombres croissants d'itérations.

Les résultats du tableau 5.12 sont représentés graphiquement sur la figure 5.12.

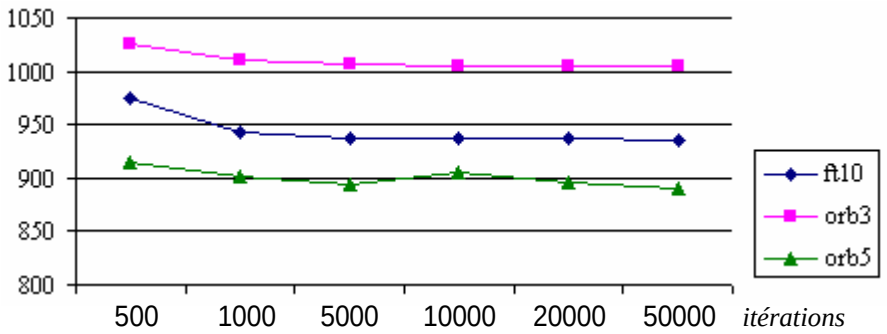


Figure 5.12 : Représentation graphique de l'influence du nombre d'itérations sur la résolution.

L'observation du graphique de la figure 5.12, et des données du tableau 5.12 met en évidence l'importance de l'ajout de plus d'itérations à la recherche. Contrairement à l'Algorithme Génétique qui arrive à une "stagnation" durant les dernières générations, du fait de la convergence de la population, la Recherche Tabou à cause de son comportement hasardeux, "profite" toujours de toute itération supplémentaire. L'absence d'une amélioration de la recherche ne signifie pas que la recherche est inutile, mais plutôt à cause de la difficulté des instances.

5. Résultats de l'application du Recuit Simulé

L'exploration des performances de Recuit Simulé se fait de la même façon que pour la Recherche Tabou, par l'exploration des ingrédients essentiels de la méthode qui sont la fonction de voisinage et la température en plus des autres paramètres de la méthode.

5.1. Les différentes fonctions de voisinage

Comme pour la Recherche Tabou, nous comparons les six fonction de voisinage qui sont : *N*1, *N*2, *NA*, *RNA*, *NB* et *NS*, avec le Recuit Simulé.

La comparaison prend sur les mêmes 18 problèmes tests. Nous effectuons cinq résolutions de chacun des problèmes par la méthode de Recuit en utilisant à chaque fois un de ces voisinages. Les expérimentations sont réalisées dans les mêmes conditions, et avec les mêmes paramètres de la méthode.

Le tableau 5.13 rapporte les résultats obtenus concernant les moyennes arithmétiques des cinq essais de chaque cas. La figure 5.13 présente la courbe des moyennes arithmétiques générales (de tous les problèmes) pour chaque fonction de voisinage.

problème	*N1*	*N2*	*NA*	*RNA*	*NB*	*NS*
abz5	1238.2	1235.2	1245.0	1238.4	1240.6	1237.4
abz6	945.2	945.6	947.6	947.4	945.2	944.8
abz7	680.4	681.0	678.8	679.8	677.3	676.0
abz8	689.0	694.8	689.4	692.6	689.6	689.0
abz9	612.6	609.6	609.2	614.2	609.8	607.2
ft10	937.8	938.4	938.2	940.4	938.0	939.4
ft20	1166.2	1167.2	1169.6	1170.8	1171.6	1171.2
la36	1295.4	1305.0	1299.4	1303.2	1297.0	1294.8
la37	1436.0	1435.2	1428.4	1435.4	1436.2	1431.6
la38	1212.2	1214.6	1209.6	1211.4	1214.4	1209.2
orb01	1070.2	1069.4	1071.0	1073.6	1067.8	1070.0
orb02	891.4	890.8	893.0	890.8	890.0	889.6
orb03	1005.6	1008.6	1007.4	1005.4	1006.0	1005.2
orb04	1020.8	1021.4	1028.0	1023.2	1022.6	1019.8
orb05	892.6	893.0	894.2	893.6	890.8	892.4
Moyenne	1006,24	1007,32	1007,18667	1008,01333	1006,46	1005,17333

Tableau 5.13 : Résultats obtenus par le Recuit Simulé avec différentes fonctions de voisinage.

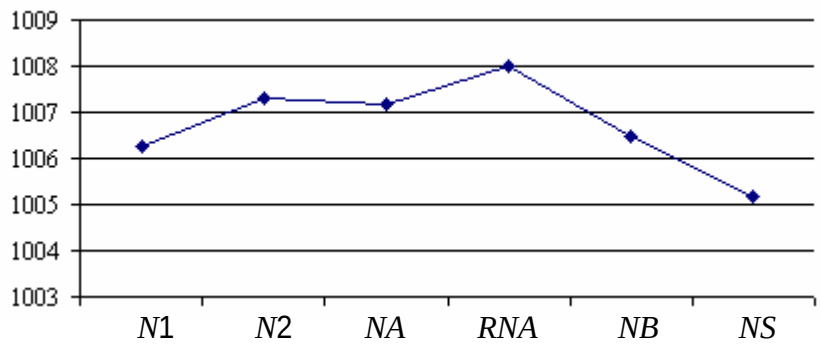


Figure 5.13 : Les moyennes générales des résultats obtenus par Recuit Simulé avec différents voisinages.

D'après les résultats présentés sur le tableau 5.13, les mêmes constations vues dans le cas de la Recherche Tabou sont à calquer ici. Pour l'ensemble des problèmes testés, il n'existe pas de différences significatives entre les différents voisinages incorporés à la méthode de Recuit en ce qui concerne le makespan, ce qui peut être démontré également par la moyenne de tous les problèmes (figure 5.13) qui met en évidence des différences minimes ne dépassant pas 0.4% pouvant établir la classification suivante de qualité décroissante : *NS* >*N*1 > *NB* > *NA* > *N*2 > *RNA*.

5.2. La qualité de la solution initiale

L'impact de la solution initiale sur la résolution par le Recuit Simulé est mis en évidence à l'aide des résultats donnée au tableau 5.14. Trois niveaux de la qualité d'initialisation (inférieure, moyenne et supérieure) sont testés avec dix problèmes. Les résultats rapportés sur le tableau sont les makespans trouvés par la résolution une seule fois de chaque problème.

Qualité	la36	la37	la38	la39	orb01	orb02	orb03	orb04	orb05	orb06
Inférieure	1323	1445	1220	1251	1088	900	1011	1044	908	1016
Moyenne	1309	1434	1207	1238	1078	889	1006	1021	894	1016
Supérieure	1294	1424	1206	1242	1068	889	1005	1013	889	1010

Tableau 5.14 : Résultats obtenus par Recuit Simulé avec trois niveaux d'initialisation.

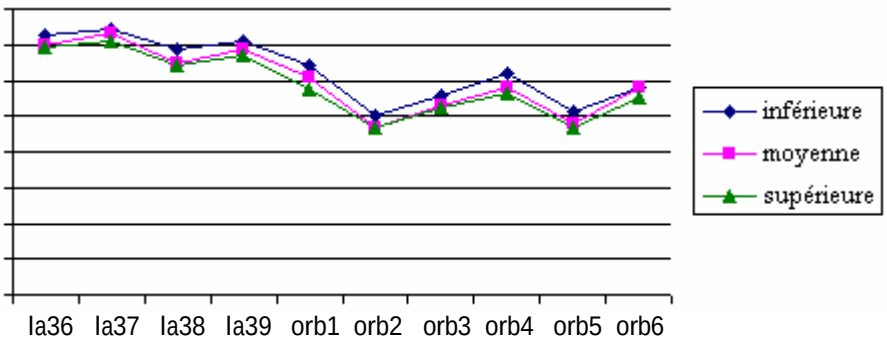


Figure 5.14 : Représentation graphique des résultats du tableau 5.14.

A la lumière des résultats trouvés, l'influence de la qualité des solutions initiales est plus importante pour le Recuit Simulé que pour la Recherche Tabou. Effectivement, le graphique de la figure 5.14, objective une dominance claire de l'initialisation de qualité supérieure sur celle de qualité inférieure. Ceci indique que la solution de départ de la méthode de Recuit est très intéressante pour l'obtention de bonnes solutions, surtout quand le nombre d'itérations est choisi réduit comme dans notre cas.

5.3. La fonction Température

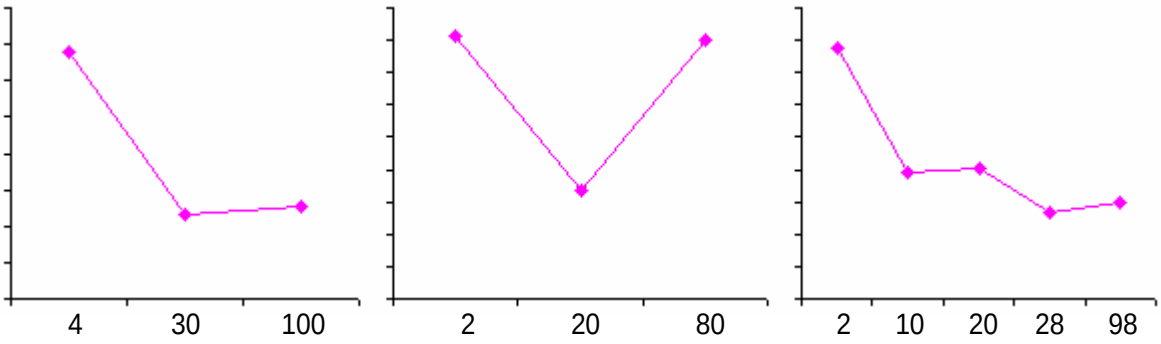
Pour évaluer l'influence des valeurs de la température sur la recherche, nous avons effectué une série d'essais sur deux problèmes la39 et orb1, avec différentes valeurs de Température Initiale et Finale. Pour chaque cas, cinq essaies ont été effectués et leur moyenne est calculée.

Le tableau 5.15 résume les résultats obtenus avec 6000 itérations.

	***T* initiale**	***T* Finale**	**Coefficient de refroidissement**	**makespan**
la39	4	2	0,000693147180559945	1269.8
	30	2	0,000451341700183702	1244.6
	100	2	0,00391202300542815	1249.2
	30	20	0,000405465108108164	1252.6
	100	80	0,00022314355131421	1248.4
orb1	4	2	0,000693147180559945	1088.0
	30	2	0,000451341700183702	1062.4
	100	2	0,00391202300542815	1060.6
	30	20	0,000405465108108164	1066.8
	100	80	0,00022314355131421	1072.2

Tableau 5.15 : Résultats obtenus avec différentes valeurs de Température.

Même si la taille de l'échantillon utilisé est insuffisante à conduire une étude sur l'impact des valeurs de Température de Recuit sur la résolution, nous procédons tous de même à effectuer des comparaisons des valeurs de Température Initiale, Finale et l'écart entre les deux, via la représentation graphique de la figure 5.15.



(A) Température Initiale **(B)** Température Finale **(C)** Ecart de Température

Figure 5.15 : Représentation graphique des résultats du tableau 5.15.

En observant les résultats du tableau 5.15, et surtout leurs représentations graphiques (figure 5.15), nous pouvons constater que des valeurs de Température Initiale ou Finale trop élevées affectent négativement la recherche. Le même effet est observé lorsque l'écart entre Température Initiale et Finale est trop réduit.

5.4. Le nombre d'itérations

La courbe de la figure 5.16 nous permet de visualiser le comportement du Recuit Simulé en cherchant la meilleure solution à notre problème. La courbe présente comme celle de la Recherche Tabou des oscillations entre solutions de diverses qualités. Mais, à l'inverse de la méthode Tabou, les oscillations tendent à être rares au fur et à mesure de l'avancement (refroidissement), et la recherche tend vers la stabilisation autour d'une solution unique. Ceci s'explique, qu'avec la recherche hasardeuse, s'ajoute l'effet de la Température qui empêche plus en plus la dégradation de solutions.

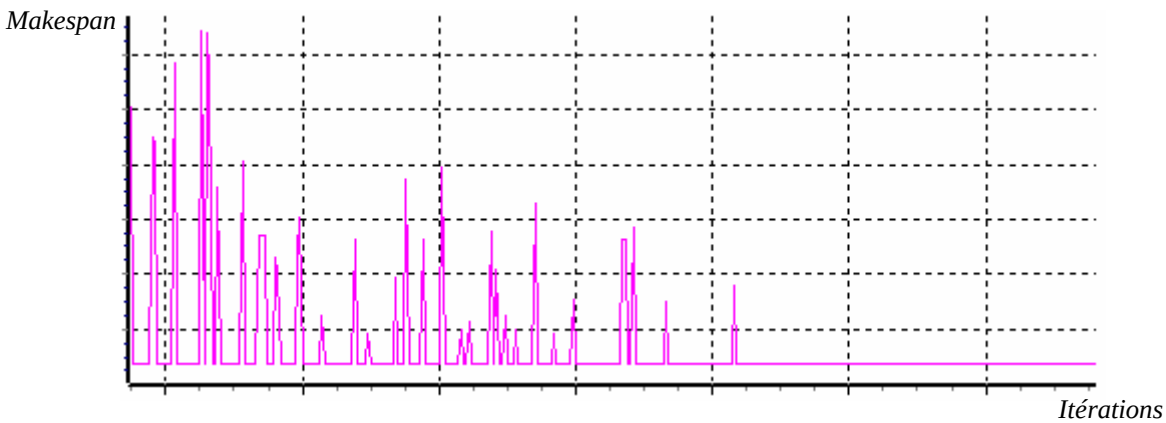


Figure 5.16 : L'évolution de la résolution avec le Recuit Simulé.

Pour ce qui concerne l'effet du nombre d'itérations sur la solution finale, le tableau 5.16 résume les résultats de résolution avec nombres différents d'itérations. Selon ces résultats, la recherche avec la méthode de Recuit est identique à la méthode de Tabou, c-à-d un nombre élevé d'itérations donne plus de chances pour arriver à une meilleure solution qu'avec moins d'itérations. L'ajout d'itérations ralentit le refroidissement qui permet à la méthode de visiter plus d'endroits dans l'espace d'état.

Itérations	**500**	**1000**	**5000**	**10000**	**20000**	**50000**
ft10	988	950	940	938	936	934
orb3	1032	1019	1006	1005	1005	1005
orb5	914	902	894	892	890	887

Tableau 5.16 : Résultats obtenus par Recuit Simulé avec nombres croissants d'itérations.

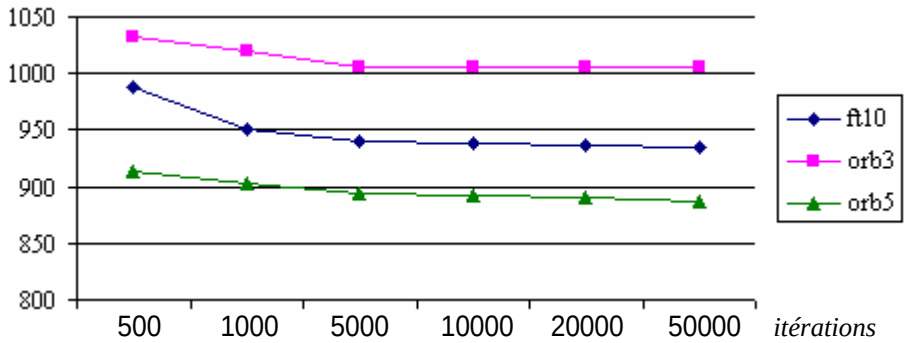


Figure 5.17 : Représentation graphique des résultats du tableau 5.16.

6. Conclusion

Nous avons présenté dans ce chapitre les résultats de différentes expérimentations réalisées sur un ensemble de problèmes tests tirés des Benchmarks. L'objectif est d'explorer les performances des différentes stratégies, d'un côté, et de valider notre implémentation des trois Métaheuristiques adaptées à la résolution du problème de Job Shop, d'un autre. Ces expérimentations nous permettent de conclure que les trois Métaheuristiques dans leur version basale s'avèrent suffisamment acceptable à la résolution de ce problème. Les tests effectués permettent également d'investiguer l'influence des différents paramètres et approches sur la résolution par les Métaheuristiques.

Même si la taille des échantillons considérés est insuffisante pour établir une analyse statistique formelle, nous pensons que nos choix sont assez représentatifs et que d'autres études similaires ne bouleverseraient pas sensiblement les résultats trouvés.

CONCLUSION GENERALE

Conclusion générale

Les problèmes de l'ordonnancement sont présents dans tous les secteurs de l'économie et constituent une fonction importante en gestion de production. Un problème d'ordonnancement consiste à allouer dans le temps des tâches à des ressources qui existent en quantité limitée, tout en satisfaisant un ensemble de contraintes.

Le problème d'ordonnancement des ateliers de type Job Shop est l'un des problèmes d'ordonnancement intensivement étudiés. C'est un problème extrêmement complexe. Il est classé parmi les problèmes combinatoires difficiles au sens fort. Cette complexité est due à l'explosion combinatoire du nombre de solutions qui croît exponentiellement avec la taille du problème. L'utilisation de méthodes exactes en vue de l'obtention de solutions optimales semble non réaliste. Le recours à des méthodes approchées comme les heuristiques est devenu incontournable. Parmi ces méthodes, s'impose le paradigme des Métaheuristiques comme une approche très prometteuse.

Les Métaheuristiques, en plus de leur adaptabilité aux différents problèmes combinatoires, ont l'avantage de ne parcourir qu'une faible fraction de l'espace de solutions pour parvenir à une solution acceptable.

Dans ce travail, nous avons traité la problématique d'application des Métaheuristiques au problème d'ordonnancement de Job Shop. Les méthodes retenues sont : les Algorithmes Génétiques, la Recherche Tabou et le Recuit Simulé. Un cadrage théorique de notre thème a consisté à présenter ses trois volets : les problèmes de l'ordonnancement en général, le problème d'ordonnancement des ateliers de type Job Shop et les Métaheuristiques. La deuxième partie du travail s'est attachée à implanter les trois méthodes en se basant sur des principes proposés par plusieurs chercheurs. Des essais numériques sont réalisés ensuite, afin d'évaluer et de comparer la performance des différentes approches implémentées, et de valider notre application.

Ce travail nous a permis de constater que ces méthodes dans leur version standard (basale), sont capables de fournir de précieux outils adaptés aux problèmes combinatoires industriels. On aurait pu croire que des versions de base de ces heuristiques auraient de la difficulté de fournir une bonne qualité de solutions, mais au contraire, les résultats obtenus montrent que ces

méthodes "simples" ont parvenu à des solutions pratiquement très satisfaisantes. Ce qui prouve que le recours à des techniques trop sophistiquées n'est pas toujours fiable.

Nous avons pu constater également, que l'application des trois Métaheuristiques à la résolution du problème complexe de l'ordonnancement de Job Shop donne des résultats qui ne peuvent conduire à une conclusion de supériorité d'une de ces méthodes par rapport aux autres.

L'approche expérimentale que nous avons conduit, nous a permis d'arriver aussi à plusieurs résultats concernant l'application de chacune des Métaheuristiques.

Pour les Algorithmes Génétiques, méthode évolutive simulant les principes de l'évolution naturelle en faisant évoluer une population de solutions via générations successives par les opérateurs de croisement, mutation et sélection. Son application nous a permis de déduire les constatations suivantes :

- L'adaptation de la méthode au problème étudié est conditionnée par la définition du codage et des opérateurs génétiques adéquats. Ces éléments déterminent grandement le comportement de la méthode.
- Le type de codage employé est très intéressant. Par exemple, parmi les codages que nous avons implémentés, le codage à base de tâches donne des résultats plus mauvais qu'avec les autres : à base d'opérations, à base des règles de priorité et à base des listes de préférences.
- Les résultats obtenus par les différents types de croisement utilisés révèlent que les croisements basés sur l'héritage de l'ordre tels que : JOX, GOX et SXX sont mieux adaptés que ceux basés sur l'héritage d'adjacence comme LOX et PPX. Pour la mutation et la sélection, les différents types n'induisent pas, d'après notre échantillon de test, de différences significatives.
- Les paramètres de dimensionnement sont un déterminant important de la méthode. Un taux de croisement faible ralentit la recherche. Alors qu'un taux élevé induit son accélération, mais il peut causer une perte de l'information génétique originale. Le taux de mutation contribue à garder une certaine diversité dans la population, mais son élévation peut causer une détérioration des individus. La taille de la population, un autre paramètre de la méthode se caractérise par un impact similaire à celui du nombre de générations. L'augmentation de la taille de la population contribue à l'obtention de bonnes solutions jusqu'à une certaine limite, après laquelle cette augmentation n'a plus d'effet, au contraire, elle peut alourdir la recherche.

- Le processus de la recherche par la méthode déroule en deux phases essentielles : la première correspond à une recherche efficace, l'évolution des génération est fructueuse. Après, commence une phase de stagnation de la recherche.

Pour la Recherche Tabou, fondue sur la recherche locale, le principe se résume en passage à chaque itération, de la solution courante à la voisine avec mémorisation des solutions les plus récentes dans une mémoire de tabous pour éviter le retour arrière. Son application à notre problème a fait émerger plusieurs résultats importants :

- Vu le comportement "hasardeux" de la méthode, le processus de résolution se caractérise par des oscillations entre des solutions de makespans divers.
- Les différentes fonctions de voisinage implémentées : $N1$, $N2$, NA, RNA, NB et NS, ne génèrent pas généralement de différences entre les solution retournées.
- Les résultats donnés par l'emploi d'une mémoire dynamique sont mieux qu'avec une mémoire de taille fixe. Ceci s'explique par le fait que le premier choix élimine automatiquement le cyclage qui constitue un obstacle sérieux à la méthode.
- L'emploi d'une mémoire trop courte ou trop longue donne pour plusieurs instances, de mauvaises solutions. La taille trop courte risque de favoriser le cyclage. Alors que, la taille trop grande peut bloquer la recherche en mettant tous les mouvements tabous.

La méthode de Recuit Simulé, est basée à l'instar de la Recherche Tabou sur le principe de la recherche locale, mais au lieu d'utiliser une mémoire, la méthode Recuit utilise la température pour guider la recherche. Son application au problème de Job Shop a permis de dégager certains résultats :

- Le processus de la recherche avec le Recuit Simulé présente, comme celui de la Recherche Tabou, des oscillations entre solutions de diverses qualités. Mais, à l'inverse de la méthode Tabou, les dégradations tendent à être rares au fur et à mesure du refroidissement (avancement de la recherche).
- Les solutions générées, au moins pour les problèmes testés, ne sont pas affectées par l'emploi de différentes fonctions de voisinage.
- Le rôle de la qualité d'initialisation est primordial dans les méthodes de recherche locale, mais il semble qu'il est plus important pour le Recuit Simulé que pour la Recherche Tabou.

- Les valeurs de Température déterminent le comportement de la méthode, des valeurs initiales ou finales trop élevées peuvent influencer négativement la recherche. Le même effet est observé lorsque l'écart de Température est trop réduit.

A l'issue du travail présenté dans ce mémoire, différentes pistes formant les directions futures à cette recherche, sont envisagées :

- Il est certain que cette étude présente des limites qui se doivent être soulevées. Tout d'abord, l'échantillon des problèmes qui a servi à comparer les méthodes est faible. En ce sens, la résolution d'un plus grand nombre de problèmes, mais aussi des problèmes de taille et de nature différentes, serait souhaitable pour compléter cette analyse. Le recours à une telle analyse peut renforcer mieux les résultats obtenus.
- La taille des instances testées dans ce travail ne dépasse pas 500 opérations pour la plus grande. Etendre ce principe de résolution à des instances plus volumineuses est un objectif majeur, surtout du point de vue pratique.
- Il peut être intéressant, d'envisager un raffinement des méthodes pour améliorer davantage les résultats obtenus. Il est envisagé aussi, d'implanter des concepts d'apprentissage afin d'apporter une aide supplémentaire pour l'ajustement des nombreux paramètres rencontrés.
- Pour des raisons de simplification, nous avons appliqué le makespan comme critère d'optimisation. Toutefois, ce critère peut n'être plus performant dans certaines situations. L'emploi d'une stratégie multicritères semble être plus bénéfique.
- Une autre direction de recherche envisagée à travers cette étude consiste à l'application des Métaheuristiques à l'ordonnancement intégré avec d'autres systèmes : la maintenance, la chaîne logistique, etc.
- Il est envisagé aussi, d'y intégrer d'autres aspects tels que le caractère stochastique ou dynamique des phénomènes qui caractérise mieux les systèmes réels de production.

REFERENCES BIBLIOGRAPHIQUES

Références bibliographiques

[Bap, 98] Baptiste P., *une étude théorique et expérimentale de la propagation des contraintes de ressources*, Thèse de Doctorat, Université de Technologie de Compiègne, 1998.

[Bie, 95] Bierwirth, C., *A Generalized Permutation Approach to Job-Shop Scheduling with Genetic Algorithms*, OR Spektrum, vol. 17(2-3), pp. 87-92.

[Bie & al., 96] Bierwirth, C., Mattfeld, D. C. & Kopfer, H., *On Permutation Representations for Scheduling Problems*, in Voigt, H. M. *et al*. (eds) *PPSN'IV Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, pp. 310-318.

[Blo, 04] Blondel F., *Gestion de la production*, 3ème édition, DUNOD, Paris 2004.

[Blu & Rol, 03] Blum C. & Andrea R., *Metaheuristics In Combinatorial Optimization: Overview And Conceptual Comparison*, ACM Computing Surveys, vol. 35, n°. 3, September 2003, pp. 268-308.

[Bri & Bru, 01] Brinkk•tter W. & Brucker P., *solving open benchmark for the job shop, Journal of Scheduling*, vol. 4 (2001), pp. 53-64.

[DCr & al., 95] Della Croce, F., Tadei, R. Volta, *A Genetic Algorithm for the Job Shop Problem*, Computers and Operations Research, 1995, vol. 22(1), pp. 15-24.

[Del & Tru, 93] M. Dell'Amico & M. Trubian, *Applying tabu search to the job-shop scheduling problem*, Annals of Operations Research, 1993, vol. 41, pp. 231-252.

[Dor & Pes, 95] Dorndorf, U. and Pesch, E., *Evolution Based Learning in a Job-Shop Scheduling Environment*, Computers and Operations Research, vol. 22(1), pp. 25-40.

[Duv & al., 98] D. Duvivier, Ph. Preux, C. Fonlupt, D. Robilliard, E-G. Talbi, *The fitness function and its impact on Local Search Methods*, IEEE Systems, Man. and Cybernetics (IEEE SMC'98), pp. 2478-2483, San Diego, USA, 1998 .

[Duv, 00] Duvivier D. *Etude de l'hybridation des méta-heuristiques, Application à un problème d'ordonnancement de type jobshop*, Thèse de Doctorat, Université du Littoral Côte d'Opale, LIL, Calais 2000.

[Fal & Bou, 91] Falkenauer E. & Bouffouix S. *A Genetic Algorithm for the Job-Shop*, Proceedings of the IEEE International Conference on Robotics and Automation, Sacremento, California, 1991, pp. 824-829.

[Fan & al., 93] Fang, H. L., Ross, P. & Corne, D. *A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling and Open-Shop Scheduling Problems*, ICGA'5 Proceedings

of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, California, 1993, pp. 375-382.

[Fon, 99] Fontanili F., *Intégration d'outils de simulation et d'optimisation pour le pilotage d'une ligne d'assemblage multiproduit à transfert asynchrone*, thèse de doctorat, Université Paris XIII, 1999.

[Fre, 82] French S., *Sequencing and scheduling: An introduction to the mathematics of the Job Shop*, Wiley, New York 1982.

[Gia, 88] Giard V., *Gestion de la Production,* 2ème édition, Economica, Paris, 1988.

[Glo & al., 96] Glover F., *tabu search and adaptive memory programming advances, applications and challenges*, in: Interfaces in Computer Science and Operations Research. Barr Helgason and Kennington, eds., Kluwer Academic Publishers, 1996.

[Hao & al., 99] Hao J-K., Galinier Ph., Michel H., *Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes*, Revue d'Intelligence Artificielle, vol. 13(2), pp. 283-324, 1999.

[Hau & Hau, 04] Randy L. Haupt & Sue Ellen Haupt, *practical genetic algorithms*, 2nd ed. John Wiley & Sons, Inc., New Jersey 2004.

[Hen, 99] Hentous H., *contribution au pilotage des systèmes de production de type Job Shop*, Thèse de Doctorat, INSA Lyon, 1999.

[Her, 05] Hertz A., *Les métaheuristiques : quelques conseils pour en faire bon usage*, dans : *Gestion de production et ressources humaines : méthodes de planification dans les systèmes productifs,* Presse internationale Polytechnique, Montréal. p. 205-222.

[Hur & Knu, 05] Hurink, J.L. & Knust, S. *Tabu search algorithms for Job-Shop problems with a single transport robot*. European journal of operational research, vol. 162 (1), 2005.

[Jai, 98] Jain, A. S., *A Multi-Level Hybrid Framework for the Deterministic Job-Shop Scheduling Problem*, PhD. Thesis, Dept. of APEME, University Of Dundee, Scotland, UK, October 1998.

[Jai & Mee, 99(1)] Jain A. S. & S. Meeran, *deterministic job shop scheduling past present and future*, European Journal of Operational Research, vol. 113, n° 2, 1, march 1999, pp. 390-434.

[Jai & Mee, 99(2)] Jain, A.S. & Meeran, S., *A State-of-the-Art Review of Job-Shop Scheduling Techniques*, European Journal of Operations Research, vol. 113, pp. 390-434, Elsevier 1999.

[Jai & al., 00] Jain, A. S., Rangaswamy, B. & Meeran, S., *New and Stronger Job-Shop Neighborhoods : A Focus on the Method of Nowicki and Smutnicki*, Journal of Heuristics, vol. 6(4), pp. 457-480, Kluwer Academic Publishers, 2000.

[Jav, 04] Javel G., *Organisation et Gestion de la Production, 3ème édition*, DUNOD, Paris 2004.

[Jef & al., 06] Jeffrey W. Herrman, & al. *Handbook Of Production Scheduling*, Springer NY, 2006.

[Jen, 01] Jensen Mikkel T., *Robust and Flexible Scheduling with Evolutionary Computation*, Ph.D. Thesis, University of Aarhus, Denmark, 2001.

[K•s & al., 99] .; Teich, T., *Algorithms for the Job Shop Problem : a comparison of different methods*. In : Proceedings of the European Symposium on Intelligent Techniques, 1999.

[Kob & al., 95] Kobayashi, S., Ono, I. & Yamamura, M., *An Efficient Genetic Algorithm for Job Shop Scheduling Problems*, Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Francisco, pp. 506-511.

[Kol, 99] M. Kolonko, *some new results on simulated annealing applied to the job shop*, European journal of operational research, 1999, vol. 113, n° 01, pp. 123-136, Elsevier.

[Kris & al., 95] K. Krishna K.Ganeshan, D. Janaki Ram, *Distributed Simulated Annealing Algorithms for Job Shop Scheduling*, IEEE Transactions on Systems Man. And Cybernetics Vol 25 no 7, 1102-1109, July 1995.

[Laa & al., 92] V. Laarhoven, P.J.M., Aarts, E.H.L., Lenstra, J.K, *Job shop scheduling by Simulated Annealing*, Operations Research, Vol. 40, Iss. 1, pp. 113-125.

[Let, 01] Letouzey A., *Ordonnancement interactif basé sur des indicateurs : Applications à la gestion de commandes incertaines et à l'affectation des opérateurs*, Thèse de Doctorat, Institut National Polytechnique de Toulouse, 2001.

[Lop & Esq, 99] Lopez P. & Esquirol P. *L'ordonnancement,* Economica, Paris 1999.

[Mat & Bie, 98] Mattfeld, D. C., Bierwirth, C., *Minimizing job tardiness : Priority rules vs. adaptive scheduling*. In I. C. Parmee editor, Proceedings of ACDM, 1998, pp. 59-67, Springer.

[Pen, 94] Penz B., *Constructions agrégatives d'ordonnancements pour des Job-Shops statiques, dynamiques et réactifs*, Thèse de Doctorat, Université Joseph Fourier - Grenoble I, 1994.

[Pez & Mer, 00] F. Pezzella & E. Merelli, *A tabu search method guided by shifting bottleneck for the Job Shop scheduling problem*, European Journal of Operational Research, vol. 120, pp.297-310, 2000.

[Pin, 95] Pinedo, M.: *Scheduling Theory, Algorithms and Systems*. Prentice-Hall, Inc. New Jersey, 1995. .

[Sad & Nak, 96] N. Sadeh Koniecpol & Y. Nakakuki, *Focused Simulated Annealing Search : An Application to Job-Shop Scheduling, tech.* report CMU-RI-TR-94-29, Robotics Institute, Carnegie Mellon University, Sept. 1994. Also in : Annals of Operations Research, vol. 60, pp. 77-103, 1996.

[Sch, 01] K. Schmidt, *Using Tabu Search to Solve the Job Shop Scheduling Problem with Sequence Dependent Setup Times*, communication, 2001.

[Sha & Yao, 04] Sharma B., Yao X., *Characterising Genetic Algorithm Approaches to Job Shop Scheduling*. UK Workshop on Computational Intelligence, Loughborough University, 2004.

[T'ki & Bil, 06] T'kindt V. & Billaut J.-C., *Multicriteria Scheduling Theory, Models and Algorithms*, Translated from French by H. Scott, Second Edition, Springer-Verlag Berlin 2006.

[Tai, 94] E. Taillard, *Parallel Taboo Search Techniques for the Job Shop Scheduling Problem*, ORSA Journal on Computing, vol. 6, pp. 108-117, 1994.

[Tai, 02] D. Taillard, *Principes d'implémentation des métaheuristiques*, Chapter 2 of J. Teghem, M. Pirlot (dir.), *Optimisation approchée en recherche opérationnelle*, Hermès, 2002, pp. 57-79.

[Tan & al., 94] Tanaev V.S, Sotskov Y.N. & Strusevich V. A., *Scheduling Theory, Multi-Stage Systems*, Kluwer Academic Publishers, Dordrecht, the Netherlands, 1994.

[Vac, 00] VACHER J. Ph., *Un système adaptatif par agents avec utilisation des algorithmes génétiques multi-objectifs : Application à l'ordonnancement d'atelier de type job-shop N×M*, Thèse de Doctorat, Université du Havre, 2000.

[Vae & al., 94] R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra, *Job Shop Scheduling by Local Search*, INFORMS Journal on Computing, vol. 3, pp. 302-317, 1996.

[V•z & Whi, 00] M. V•zquez, L. Darrell Whitley, *A Comparison of Genetic Algorithms for the Static Job Shop Scheduling Problem*, Source Lecture Notes In Computer Science, vol. 1917, Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, pp. 303-312, Publisher Springer-Verlag London, 2000.

[Wat & al., 03] Watson, J.-P., Beck, J.C., Howe, A.E. & Whitley, L.D., *Problem Difficulty for Tabu Search in Job-Shop Scheduling*, Artificial Intelligence, vol. 143(2), pp. 189-217, 2003.

[Wat & al., 06] J.P. Watson, A.E. Howe, and L.D. Whitley. *Deconstructing Nowicki and Smutnicki's i-TSAB Tabu Search Algorithm for the Job-Shop Scheduling Problem*, Journal of Computers and Operations Research, vol. 33, n°. 9, pp 2623-2644, September 2006.

[Yam & Nak, 92] T. Yamada & R. Nakano, *A genetic algorithm applicable to large scale Job Shop problems*, in Parallel problem solving from nature pp. 281-290 Elsevier Science Publishers 1992.

[Yam & Nak, 96] T. Yamada & R. Nakano, *Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search*, In Meta-heuristics: theory & applications, Kluwer academic publishers, MA, USA, 1996, pp. 237-248.

[Yam & al., 96] Ono, I., Yamamura, M., Kobayashi, S. : *A Genetic Algorithm for Job-shop Scheduling Problems Using Job-based Order Crossover, Proc*. of ICEC'96, pp.547-552 (1996), Proceedings of 1996 IEEE International Conference on Evolutionary Computation, pp. 547-552.

ANNEXE

Annexe

Les solutions optimales de différents benchmarks, et les premiers auteurs parvenant à leurs meilleures solutions :

Problème	**Taille**	**Makespan optimal**	**Premiers auteurs donnant la meilleure résolution**
Benchmarks de Fisher & Thompson (1963)			
ft06	6x6	55	Balas (69)
ft10	10x10	930	Lageweg (84)
ft20	20x5	1165	McMahon & Florian (75)
Benchmarks de Adams, Balas & Zawack (1988)			
abz5	10x10	1234	Applegate & Cook (91)
abz6	10x10	943	Adams et al. (88)
abz7	20x15	656	Martin (96)
abz8	20x15	665* (645)	Martin (96)
abz9	20x15	679* (661)	Balas & Vazacopoulos (88)
Benchmarks de Lawrence (1984)			
la1 (f1)	10x5	666	Adams et al. (88)
la2 (f2)	10x5	655	Matsuo et al. (88)
la3 (f3)	10x5	597	Matsuo et al. (88)
la4 (f4)	10x5	590	Matsuo et al. (88)
la5 (f5)	10x5	593	Adams et al. (88)
la6 (g1)	15x5	926	Adams et al. (88)
la7 (g2)	15x5	890	Adams et al. (88)
la8 (g3)	15x5	863	Adams et al. (88)
la9 (g4)	15x5	951	Adams et al. (88)
la10 (g5)	15x5	958	Va Laarhooven et al. (92)
la11 (h1)	20x5	1222	Adams et al. (88)
la12 (h2)	20x5	1039	Adams et al. (88)
la13 (h3)	20x5	1150	Adams et al. (88)
la14 (h4)	20x5	1292	Adams et al. (88)
la15 (h5)	20x5	1207	Adams et al. (88)
la16 (a1)	10x10	945	Carlier and Pinson (90)
la17 (a2)	10x10	784	Matsuo et al. (88)
la18 (a3)	10x10	848	Matsuo et al. (88)
la19 (a4)	10x10	842	Matsuo et al. (88)
la20 (a5)	10x10	902	Van Laarhooven et al. (92)
la21 (b1)	15x10	1046	Vaessens (96)
la22 (b2)	15x10	927	Matsuo et al. (88)
la23 (b3)	15x10	1032	Adams et al. (88)
la24 (b4)	15x10	935	Vaessens (96)
la25 (b5)	15x10	977	Vaessens (96)
la26 (c1)	20x10	1218	Matsuo et al. (88)
la27 (c2)	20x10	1235	Carlier and Pinson (94)
la28 (c3)	20x10	1216	Matsuo et al. (88)
la29 (c4)	20x10	1152	Martin (96)
la30 (c5)	20x10	1355	Adams et al. (88)

la31 (d1)	20x10	1784	Adams et al. (88)
la32 (d2)	20x10	1850	Adams et al. (88)
la33 (d3)	20x10	1719	Adams et al. (88)
la34 (d4)	20x10	1721	Adams et al. (88)
la35 (d5)	20x10	1888	Adams et al. (88)
la36 (i1)	15x15	1268	Carlier & Pinson (90)
la37 (i2)	15x15	1397	Vaessens (96)
la38 (i3)	15x15	1196	Nowicki & Smutnicki (96)
la39 (i4)	15x15	1233	Vaessens (96)
la40 (i5)	15x15	1222	Vaessens (96)
Benchmarks de Applegate & Cook (1991)			
orb 1	10x10	1059	Applegate & Cook (91)
orb 2	10x10	888	Applegate & Cook (91)
orb 3	10x10	1005	Applegate & Cook (91)
orb 4	10x10	1005	Applegate & Cook (91)
orb 5	10x10	887	Applegate & Cook (91)
orb 6	10x10	1010	Vaessens (96)
orb 7	10x10	397	Vaessens (96)
orb 8	10x10	899	Vaessens (96)
orb 9	10x10	934	Vaessens (96)
orb 10	10x10	944	Vaessens (96)
Benchmarks de Storer, Wu & Vaccari (1992)			
swv 1	20x10	1407	Martin (96)
swv 2	20x10	1475	Martin (96)
swv 3	20x10	1398*(1369)	Vaessens (96)
swv 4	20x10	1483*(1450)	Vaessens (96)
swv 5	20x10	1424	Vaessens (96)
swv 6	20x15	1678*(1591)	Vaessens (96)
swv 7	20x15	1620*(1446)	Vaessens (96)
swv 8	20x15	1763*(1640)	Vaessens (96)
swv 9	20x15	1663*(1604)	Vaessens (96)
swv 10	20x15	1767*(1631)	Vaessens (96)
swv 11	50x10	2991*(2983)	Vaessens (96)
swv 12	50x10	3003*(2972)	Vaessens (96)
swv 13	50x10	3104	Vaessens (96)
swv 14	50x10	2968	Vaessens (96)
swv 15	50x10	2904*(2885)	Vaessens (96)
swv 16	50x10	2924	Storer et al. (92)
swv 17	50x10	2794	Storer et al. (92)
swv 18	50x10	2852	Storer et al. (92)
swv 19	50x10	2843	Storer et al. (92)
swv 20	50x10	2823	Storer et al. (92)
Benchmarks de Yamada & Nakano (1992)			
yn 1	20x20	888*(826)	Caseau & Laburthe (95)
yn 2	20x20	909*(861)	Caseau & Laburthe (95)
yn 3	20x20	893*(827)	Vaessens (96)
yn 4	20x20	968*(918)	Vaessens (96)

(*) : Indique que la solution optimale n'est pas encore atteinte.