République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Boudiaf M'Sila
Faculté Mathématiques et Informatique
Département d'Informatique

Mémoire

Présenté par

RAHNEBI Asma

Pour l'obtention du diplôme de Master

Filière: Informatique

Option: Informatique Décisionnelle Optimisé

Thème

Etude comparative sur l'application des métaheuristiques à un problème d'ordonnancement d'atelier de type job shop

Soutenu le :		Devant le Jury composé de :
Nom et Prénom		Grade
Dr. Bouderah Brahim	Université de M'sila	**Président**
Dr. Mouhoub Nacer Eddine	Université de M'sila	**Rapporteur**
Dr. Hemmak allaoua	Université de M'sila	**Examinateur**

Année Universitaire : **2017-2018**

Remerciements

Je remercie Dieu le tout puissant, qui m'a donné la force et la patience pour l'accomplissement de ce travail.

Nous remercions en particulier Mr. ***Mouhoub Nasser Eddine****, pour l'honneur qu'il m'a fait de bien vouloir m'encadrer, et pour les conseils donnés lors de la réalisation de ce travail.*

J'adresse mes remerciements au membres de jury pour avoir accepté de me prêter de leur attention et évaluer ce travail.

Je réserve les derniers remerciements à toute ma famille et mes amis, en particulier mes parents pour leurs sacrifices et leurs encouragements.

TABLE DES MATIÈRES

CHAPITER 2 LE PROBLEME D'ORDONNANCEMENT JOB SHOP

CHAPITRE 3 LES METAHEURISTIQUES

Liste de Figures

Liste de Tables

INTRODUCTION GENERALE

La recherche opérationnelle est une discipline dont le but est de fournir des méthodes pour répondre à un type précis de problème. Sa vocation scientifique est donc de construire des modèles formels d'aide à la décision, en particulier les modèles lies à des problèmes d'optimisation, et de proposer des méthodes de résolution efficace de ces modèles.

Ainsi, la théorie de l'ordonnancement est une branche de la recherche opérationnelle qui s'intéresse au calcul de dates d'exécution optimales des taches. Pour cela, il est très souvent nécessaire d'affecter en même temps les ressources nécessaires à l'exécution de ces tâches.

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.

Le problème d'ordonnancement est l'un des problèmes les plus étudiés dans le domaine de la recherche opérationnelle. Ce problème consiste à trouver une séquence optimale pour l'exécution de *n* jobs sur *m* machines afin d'optimiser une fonction objectif et de déterminer également les dates de début et de fin d'exécution de chaque job.

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 deux méthodes ont démontré leur efficience dans nombreuses applications : les Algorithmes Génétiques appartiennent aux méthodes évolutives , le Recuit Simulé qui est basés sur le principe de la recherche locale.

Dans notre mémoire, nous développons une application qui minimise le Cmax (makespan, c'est le temps nécessaire pour terminer toues le tâches) pour résoudre un problème job shop.

Dans le premier chapitre, nous allons étudier l'ordonnancement des activités de production en général, les définitions de l'ordonnancement et de ses éléments, les notations des problèmes d'ordonnancement et classification des ateliers (Job Shop, Flow Shop, Open Shop).

Dans le deuxième chapitre, nous allons détailler sur le problème de job shop : la modélisation graphique, la complexité et les méthodes pour résoudre ce problème.

Le troisième chapitre, nous allons discuter sur les métaheuristiques l'algorithme génétique et le recuit simulé, ainsi que leur application dans le problème de job shop.

Le dernier chapitre, fait appel à une démarche expérimentale afin de montrer l'efficience des métaheuristiques utilisées dans la résolution du problème étudié. Les résultats obtenus sont discutés et analysés afin d'appartenir certaine conclusion.

CHAPITRE 1

LA PRODUCTION ET GESTION DE LA PRODUCTION

1 Introduction

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

Dans la première section on va présenter des notions de base concernant la production, la gestion de production et le rôle de l'ordonnancement. Dans la deuxième section vise à présenter le problème de l'ordonnancement en précisant sa définition et les différents éléments qui le déterminent. La troisième section présente notation des problèmes d'ordonnancement, La dernière section est consacrée classification des ateliers (Job Shop, Flow Shop, Open Shop).

2 La production et gestion de la production

La production est le processus conduisant à la création de produits par l'utilisation et la transformation de ressources [1]. 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[2].

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.

Les systèmes de production industrielle se sont considérablement diversifiés et compliqués. 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 [3].

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 [4] [2].

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).

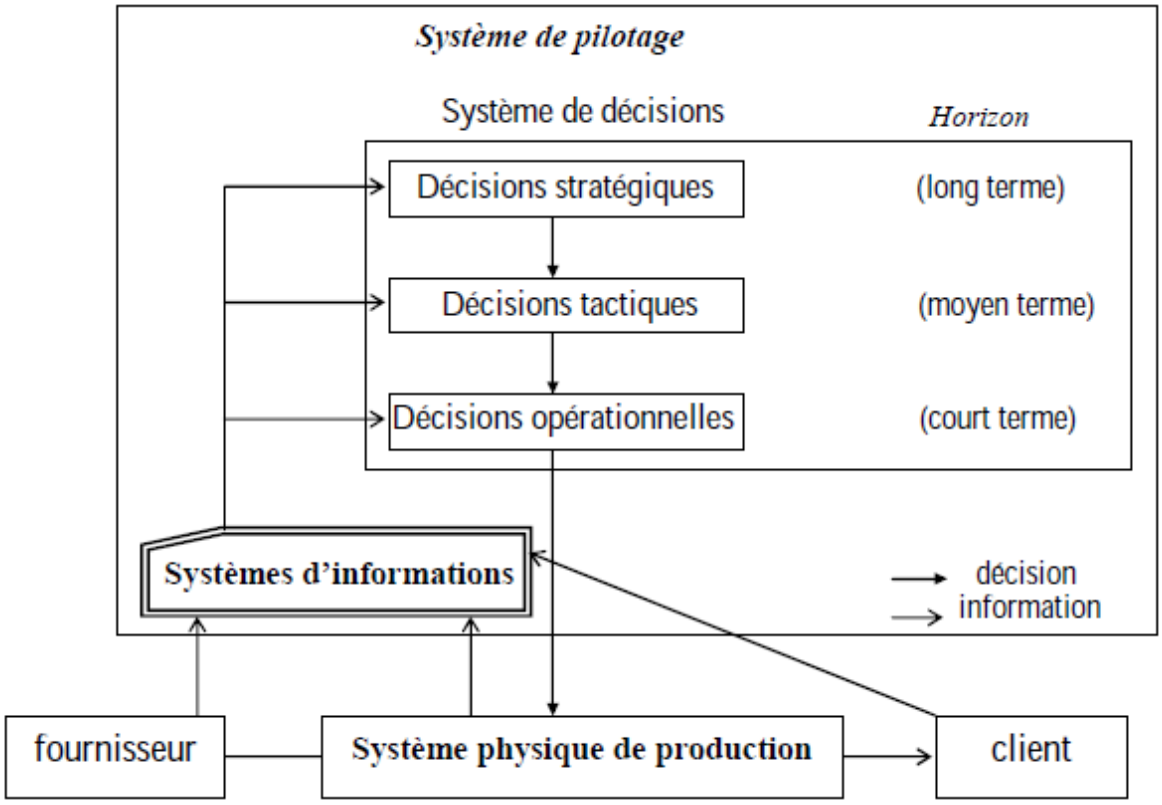


Figure 1.1 Les sous-systèmes constituant le système de production d'après [4].

- Le système physique de production : Transforme les matières premières ou composantes 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 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 [3].

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 » [5].

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 » [3].

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).

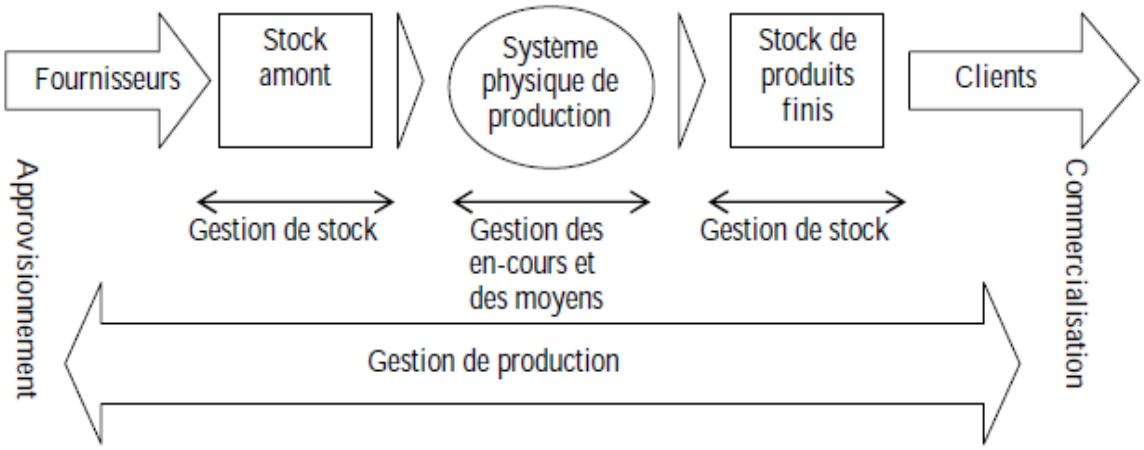


Figure 1.2 Objectifs de la gestion de production [2].

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 Décomposition hiérarchique de la gestion de production

On distingue trois niveaux hiérarchiques de la gestion de production (Figure 1.1) : stratégique, tactique et opérationnel [4] [5]:

a. Le niveau stratégique

Ce niveau trace la politique à long terme de l'entreprise (à un horizon de plus de deux ans). Cette politique 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

Ce niveau relie les deux niveaux stratégique et opérationnel, il porte sur les décisions à moyen terme. Le but est d'assurer une production satisfaisante à la demande en minimisant les coûts, tout en respectant le plan tracé par le niveau stratégique de l'entreprise.

c. le niveau opérationnel

Ce niveau porte sur les décisions à court terme. Il s'agit d'une gestion quotidienne pour satisfaire les demandes en respectant les décisions tactiques.

Parmi les décisions opérationnelles : la gestion de la main d'œuvre, la gestion des stocks, la gestion des équipements.

2.3 Rôle de l'ordonnancement dans la gestion de production

Le modèle général en gestion de production, décompose les décisions en trois niveaux [3] :

Stratégique, tactique et opérationnel (pilotage et suivi quotidien des flux de matières et du travail). Cette hiérarchisation se traduit par une échelle de responsabilité (direction, cadre, agent ...).

Dans la mesure où des événements aléatoires telles que des pannes ou des commandes imprévues, qui peuvent survenir à tout moment, il est nécessaire de recalculer fréquemment l'ordonnancement, il est difficile de résoudre le problème d'ordonnancement au niveau supérieur (stratégique). Donc, il est résolu au niveau inférieur (opérationnel) [3].

La place de l'ordonnancement varie entre le niveau tactique et le niveau opérationnel. Il s'occupe de la réalisation des décisions venant de niveau supérieur. Son rôle consiste à transformer les décisions de fabrication définies par le programme directeur en instructions d'exécution destinées à contrôler et piloter à court terme l'activité des postes de travail [6].

En sortie de la fonction d'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 [2].

La fonction ordonnancement se décompose en trois sous fonctions [2] :

- L'élaboration des ordres de fabrication (OF) : consiste à transformer les informations du programme directeur de production (suggestion de fabrication) en OF.
- L'élaboration du planning d'atelier : 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.

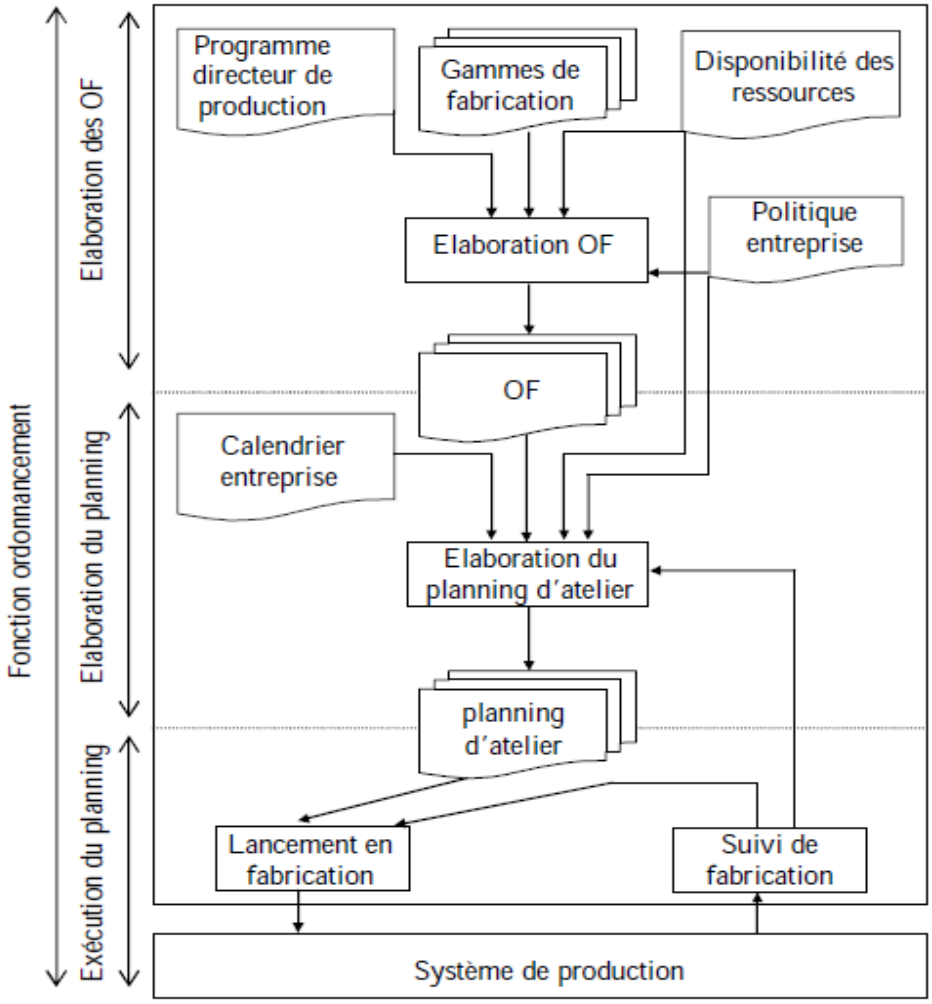


Figure 1.3 Les sous fonctions de l'ordonnancement dans l'atelier [2].

3 Présentation du problème d'ordonnancement

3.1 Définition du problème d'ordonnancement

Ordonnancer c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution.

Présentons une autre définition qui est plus explicite : « Un ordonnancement constitue une solution au problème d'ordonnancement. Il décrit l'exécution des tâches et l'allocation des ressources au cours du temps, et vise à satisfaire un ou plusieurs objectifs. Plus précisément, on parle de problème d'ordonnancement lorsqu'on doit déterminer les dates de début et les dates de fin des tâches, alors qu'on réserve le terme de problème de séquencement au cas où l'on cherche seulement à fixer un ordre relatif entre les tâches qui peuvent être en conflit pour l'utilisation des ressources. Un ordonnancement induit nécessairement un ensemble unique de relations de séquencement [7].

L'ordonnancement est la programmation dans le temps de l'exécution d'une série de tâches (ou activités, opérations) sur un ensemble de ressources physiques (humaines et techniques), en cherchant à optimiser certains critères, financiers ou technologiques, et en respectant les contraintes de fabrication et d'organisation [4].

3.2Eléments 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" [8].

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 [1].

3.2.2 Les ressources

Pour l'exécution des tâches, ces dernières requièrent certaines ressources telles que des machines, la main d'œuvre, les moyens financiers, etc.

Une ressource k est donc, un moyen humain ou technique qui est utilisé dans la réalisation d'une tâche. Elle est disponible en quantité limitée.

La disponibilité est généralement exprimée par une capacité propre à chaque ressource k notée Q_k ($Q_k \geq 1$). On distingue deux types de ressources: les ressources renouvelables et les ressources consommables.

Une ressource est consommable si, après avoir été allouée à une tâche, elle n'est plus disponible pour les tâches suivantes. Le cas pour l'argent, la matière première, etc.

Une ressource est renouvelable si, après avoir été allouée à une tâche, elle redevient disponible après la fin de cette tâche pour les tâches suivantes. C'est le cas pour les machines, les processeurs, les fichiers, le personnel, etc.

On distingue par ailleurs, principalement dans le cas de ressources renouvelables, les ressources disjonctives (ou non partageables) qui ne peuvent exécuter qu'une tâche à la fois(machine, robot, etc.) et les ressources cumulatives (ou partageables) qui peuvent être utilisées par plusieurs tâches en même temps (équipe d'ouvriers, poste de travail, etc.).

Les problèmes d'ordonnancement à ressources disjonctives couvrent une classe importante d'applications qu'on appelle les problèmes d'atelier ou de machines.

Les problèmes d'ordonnancement d'atelier et sous les contraintes cumulatives feront l'objet d'une étude à vol d'oiseau ultérieurement dans ce chapitre.

La nature des ressources prises en considération permet de dresser une typologie des problèmes d'ordonnancement (figure1.4) [7]

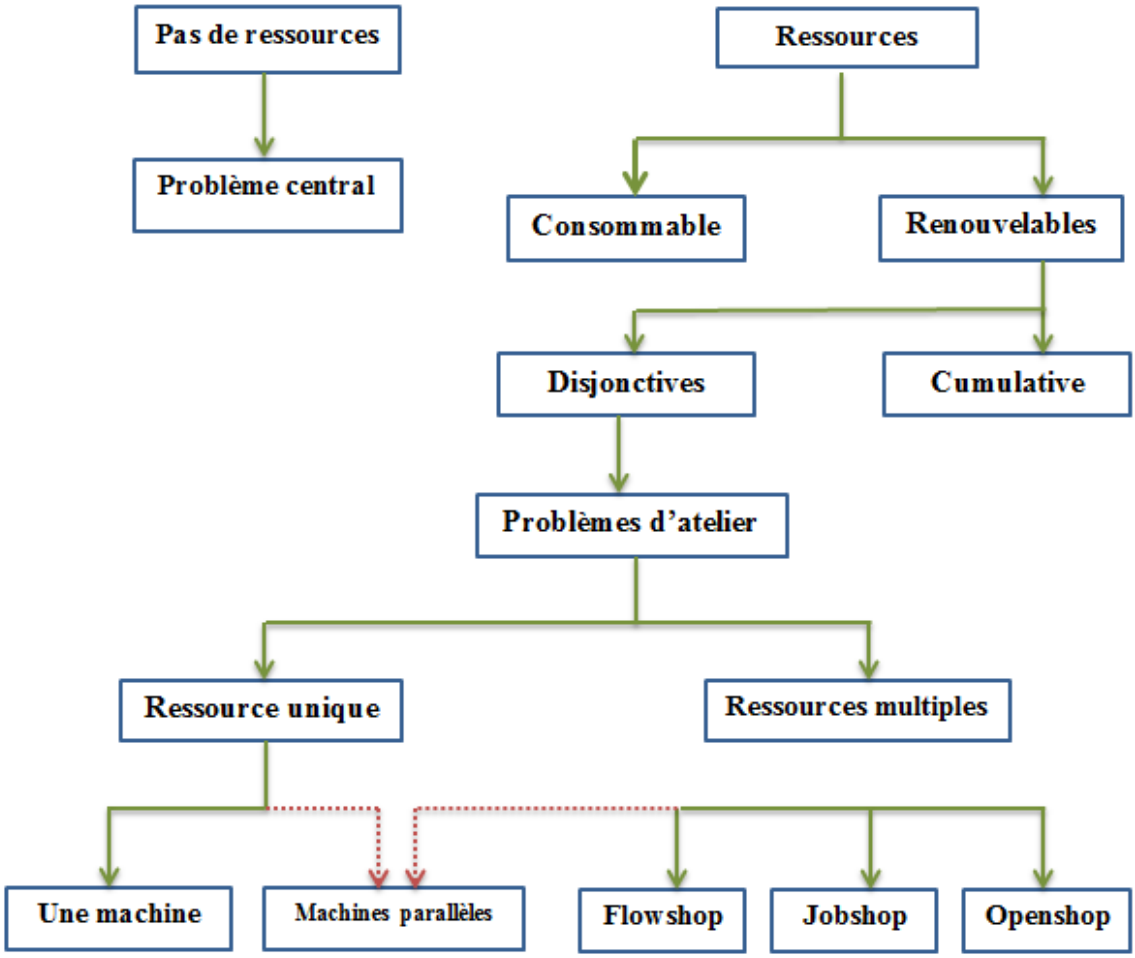


Figure 1.4 Typologie des problèmes d'ordonnancement par les ressources tirée de [7].

3.2.3 Les contraintes

Suivant la disponibilité des ressources et suivant l'évolution temporelle, deux types de contraintes peuvent être distingués [9] : contraintes de ressources et contraintes temporelles.

- les contraintes de ressources : plusieurs types de contraintes peuvent être induites par la nature des ressources. A titre d'exemple, la capacité limitée d'une ressource implique un certain nombre, à ne pas dépasser, de tâches à exécuter sur cette ressource.

Les contraintes relatives aux ressources peuvent être disjonctives, induisant une contrainte de réalisation des tâches sur des intervalles temporels disjoints pour une même ressource, ou cumulatives impliquant la limitation du nombre de tâches à réaliser en parallèle.

- les contraintes temporelles : elles représentent des restrictions sur les valeurs que peuvent prendre certaines variables temporelles d'ordonnancement. Ces contraintes peuvent être :
  - des contraintes de dates butoirs, certaines tâches doivent être achevées avant une date préalablement fixée,
  - des contraintes de précédence, une tâche i doit précéder la tâche j,
  - des contraintes de dates au plus tôt, liées à l'indisponibilité de certains facteurs nécessaires pour commencer l'exécution des tâches [9].

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 [3] :

- 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 Notation des problèmes d'ordonnancement

Dans la littérature, un système de notation a été proposé, pour représenter un problème d'ordonnancement d'une manière simple. Il consiste à représenter le problème par trois champs α, β et γ, ($\alpha/\beta/\gamma$) [10] :

- le champ α, permettant de décrire le type d'atelier, le nombre de jobs à réaliser et le nombre de machines disponibles,
- le champ β, caractérisant les conditions d'exécution des jobs ainsi que les états des ressources présentes dans l'atelier. Il indique en particulier la présence ou non des contraintes de précédence entre les tâches et la possibilité de tolérer la préemption,
- le champ γ, dédié au(x) critère(s) à optimiser.

Pour l'exemple suivant :

$$J, 10, 6|Prec, ri\ |Cmax$$

il s'agit d'un problème d'ordonnancement d'un atelier de type job-shop, ayant dix jobs et six machines disponibles. Le deuxième champ indique que les jobs présentent une contrainte de précédence, *Prec*, et une contrainte *ri* de dates de début au plus tôt. En plus, la préemption est interdite (puisqu'elle n'est pas mentionnée dans le champ *prem*). Le dernier champ montre que l'objectif est de minimiser le makespan, C_{max}[10].

5 Classification des ordonnancements

Cette notion de compacité qui est un objectif majeur de tout ordonnancement est la base de distinction de plusieurs classes d'ordonnancement : admissible, semi-actif, actif, sans délai [11].

5.1 Ordonnancement admissible

Si toutes les contraintes du problème sont bien respectées, l'ordonnancement est dit : « 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 [11] :

• Décalage à gauche local : l'avancement du début d'une opération ne remet pas en cause l'ordre des autres opérations,

• Décalage à gauche global : l'avancement du début d'une opération engendre une modification au niveau de l'ordre relatif aux deux opérations au minimum.

5.2 Ordonnancement semi-actif

Si aucun décalage local n'est possible, l'ordonnancement est dit Semi-actif. Donc, aucune opération ne peut être exécutée en plus tôt sans modifier l'ordre relatif au moins de deux opérations [12].

5.3 Ordonnancement actif

Si aucun décalage à gauche que ce soit local ou global n'est possible, l'ordonnancement est dit Actif. En conséquence, il est impossible d'avancer une opération sans reporter le début d'une autre opération [11].

5.4 Ordonnancement sans délais

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. À noter que la transformation en un ordonnancement sans délai peut mener à une solution plus mauvaise du point de vue makespan [12].

Il existe une relation d'inclusion entre les différentes classes d'ordonnancement précédentes (Figure 2.2) :

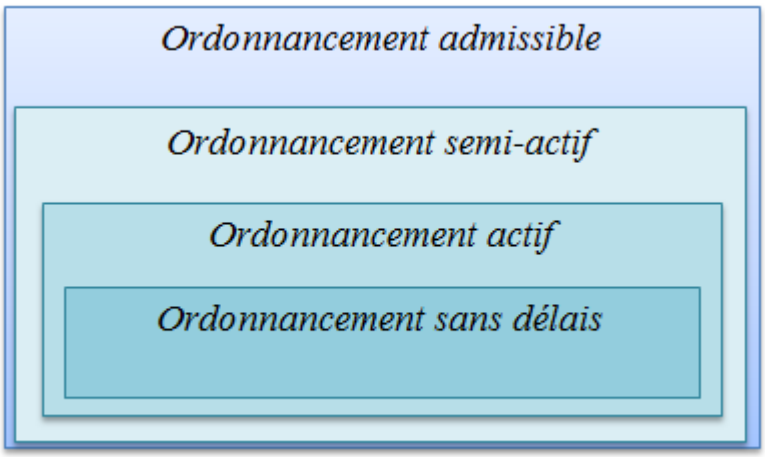


Figure 1.5 Relation d'inclusion entre les différentes classes d'ordonnancement [11].

D'après la Figure 1.5, on constate que les ordonnancements semi-actif, actif et sans délais sont inclus dans la classe d'ordonnancement admissible, puisqu'ils vérifient toutes les contraintes d'un ordonnancement admissible.

6 Classification des ateliers

Classifier les problèmes d'ordonnancement revient à étudier les différents types de machines utilisées ainsi que la manière avec laquelle l'atelier est organisé.

Les ateliers dans les systèmes de production sont nombreux, ils se diffèrent par le nombre de machines, la nature des jobs (par exemple la composition des jobs d'une ou de plusieurs tâches), l'unicité ou la diversité du routage des jobs au niveau des machines, ou par la flexibilité des ressources (c'est-à-dire la possibilité d'avoir un sous-ensemble de machines

candidates dans lesquelles une telle tâche ou opération peut être traitée), dans le cas d'une flexibilité partielle, sinon n'importe quelle tâche peut être exécutée sur n'importe quelle ressource. Selon ces paramètres, nous distinguons les types d'ateliers suivants [3]:

6.1 Ateliers Flow-Shop

Dans ce type d'atelier, la ligne de fabrication est constituée de plusieurs machines en série, de telle sorte que toutes les opérations de tous les jobs passent par toutes les machines en respectant le même ordre. C'est pour cela que ces ateliers sont appelés les ateliers à cheminement unique. Si on trouve plusieurs exemplaires identiques et parallèles de la même machine, l'atelier devient Flow-Shop Hybride [12].

6.2 Ateliers Job Shop et job shop flexible

Contrairement au type d'atelier précédent, l'atelier Job Shop se caractérise par un cheminement multiple, puisque les opérations de chaque Job peuvent emprunter divers chemins (routage des opérations).

Généralement, on distingue deux organisations d'atelier Job Shop :

1. Organisation simple : s'il existe pour chaque machine un seul exemplaire.
2. Organisation hybride : s'il existe au moins une machine disposant de plusieurs exemplaires.

L'atelier Job Shop Flexible est une extension du problème classique décrit précédemment. La flexibilité est due aux ressources, c'est-à-dire, l'attribution d'un sous-ensemble de ressources (machines candidates) pour le traitement de chaque opération de telle sorte que le temps opératoire de chaque opération dépend de la machine candidate sélectionnée.

En effet, il existe plusieurs degrés de flexibilité : la flexibilité faible dans laquelle quelques opérations qui sont traitables dans quelques machines. Ensuite, dans la flexibilité moyenne et forte le nombre d'opérations traitables dans plusieurs machines devient de plus en plus important. En Arrivant à la flexibilité extrême (totale), n'importe quelle opération est traitable sur n'importe quelle machine [12].

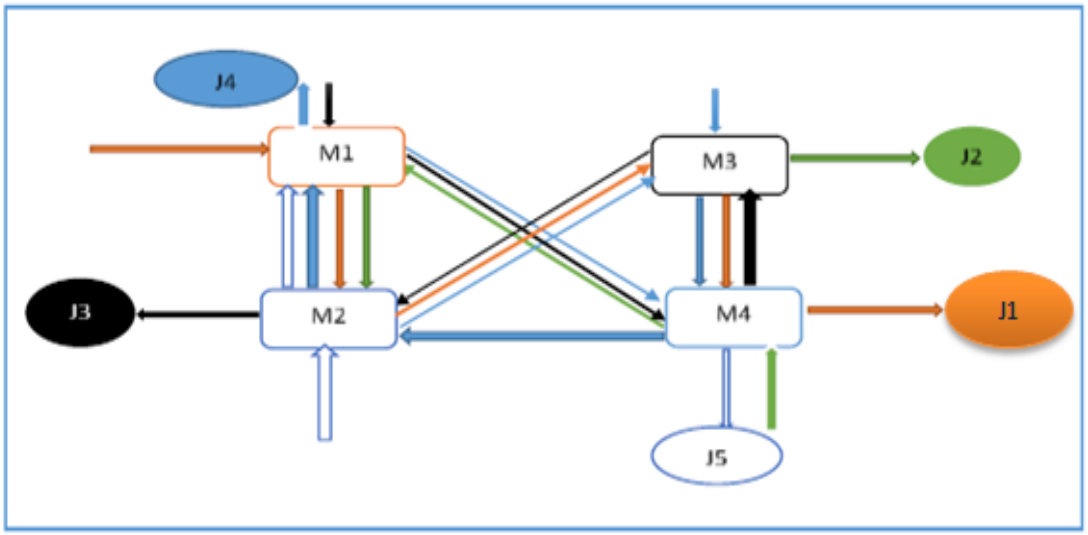


Figure 1.6 Représentation d'un système de type Job-shop classique [13].

6.3 Ateliers Open Shop

Dans ce type d'ateliers, l'acheminement d'opérations est multiple et libre, autrement dit, il n'existe aucun ordre d'exécution des opérations (les gammes sont libres). On trouve ce type d'atelier dans le cas où la fabrication de chaque produit se traduit par le traitement de plusieurs opérations, dont l'ordre est totalement libre [3].

7 Conclusion

Dans ce chapitre Nous avons abordé l'aspect commun des problèmes d'ordonnancement en production. D'abord, en présentant brièvement 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 st les objectives, ainsi que les classifications des ateliers (Job Shop, Flow Shop, Open Shop). Après, en présentant les méthodes de résolution de problèmes d'ordonnancement.

La description de notre problème d'ordonnancement job shop va présenter dans le chapitre suivant.

CHAPITRE 2
LE PROBLEME D'ORDONNANCEMENT DE JOB SHOP

1 Introduction

Dans ce chapitre, nous présentons les bases théoriques au problème, en se concentrant sur sa spécificité en termes 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 objectives. La troisième section présente les différentes classes d'ordonnancement. La modélisation du problème de Job Shop 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.

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×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 [5] :

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 :

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

Où • 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 i 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

Selon [3], 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 [3]:

▪ 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 [3].

Le problème de Job Shop sous cette forme est dit simple : chaque tâche est composée 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 [3].

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

Nous ne rappelons ici que quelques critères les plus importants [3] :

▪ C_{max} : le makespan, est la durée totale de l'ordonnancement.

▪ $\overline{C}$: est la moyenne des temps d'achèvement des tâches.

▪ $\Sigma w_i C_i$: 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 Modélisation du problème de Job Shop

Dans le domaine d'ordonnancement, plusieurs outils sont disponibles non seulement pour la représentation graphique des solutions mais également pour la spécification sans ambiguïté des données et des contraintes du problème. Dans cette section, nous présentons quelques outils de base largement employés dans la littérature des problèmes d'ordonnancement.

3.1 Diagramme de Gantt

Malgré que les diagrammes de Gantt datent de plus d'un siècle, ils sont considérés comme outil de modélisation très populaire en gestion pendant la phase de planification, puis comme un procédé de contrôle pour valider les résultats. Le diagramme de Gantt est un modèle graphique simple et flexible pour représenter l'exécution des tâches sur une période de temps. Un segment horizontal de longueur proportionnelle à la durée opératoire est associé à chaque tâche avec l'hypothèse que ces durées sont connues avec certitude. Il existe deux visions pour l'élaboration de ce genre de diagrammes [14] :

- Le diagramme à base de ressource où la ligne horizontale correspond à la ressource, ce qui rend possible l'illustration de ses périodes d'opération et d'oisiveté ainsi que l'ordre de passage des opérations ;
- Le diagramme à base de produit où une ligne est affectée à chaque tâche pour permettre de déterminer le chaînage de ses opérations et le temps d'attente entre deux opérations consécutives.

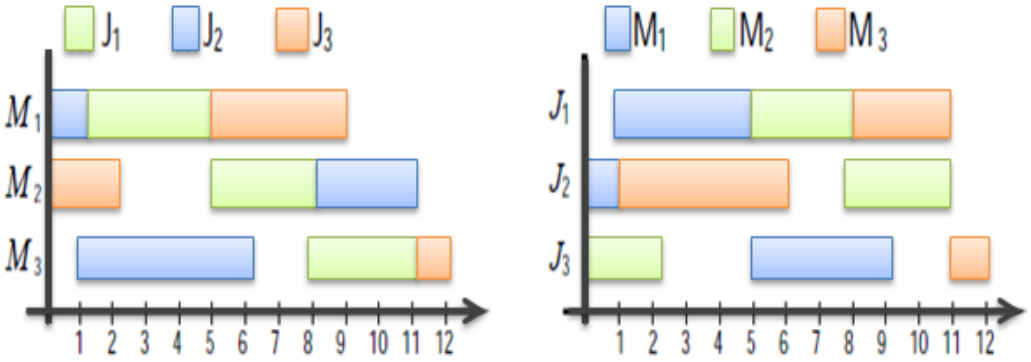


Figure 2.1 Diagrammes de Gantt (problème de Job Shop) [3].

3.2 Graphe disjonctif

L'utilisation des graphes disjonctifs pour la modélisation des problèmes d'ordonnancements d'ateliers est sans doute parmi les techniques les plus exploitées. Introduite initialement par Roy et Sussman en 1964, elle fut reprise par la suite par plusieurs chercheurs où elle est à l'origine des premières contributions pour le traitement des problèmes d'ordonnancement. Cette tendance peut être expliquée par la simplicité de construction de ces graphes et le support de différentes contraintes entre les dates de début et de fin des opérations. Un graphe disjonctif est composé des éléments suivants [14] :

- Les nœuds : Ces nœuds représentent soit des débuts des opérations, ou bien des opérations fictives correspondantes à la fin de chaque travail, ou bien les dates du début et de la fin de l'ordonnancement;
- Les arcs : Nous distinguons deux types d'arcs. Les arcs conjonctifs sont associés aux contraintes de précédence entre deux opérations consécutives et valués par la durée opératoire de la première opération. Les arcs disjonctifs exprimant les conflits d'utilisation de ressources où deux opérations ne peuvent pas s'exécuter en même temps si elles sont reliées par un tel arc.

Un ordonnancement admissible sur le graphe est obtenu en arbitrant chacune des disjonctions. Cela revient donc à préserver uniquement un arc de chaque paire d'arcs dans le but de fixer l'ordre de passage sur la machine. Donc, le graphe disjonctif contient toutes les informations nécessaires pour décrire une solution partielle ou complète du problème d'ordonnancement comme indiqué par la figure 2.4 associée à un atelier du type job shop avec trois machines [14].

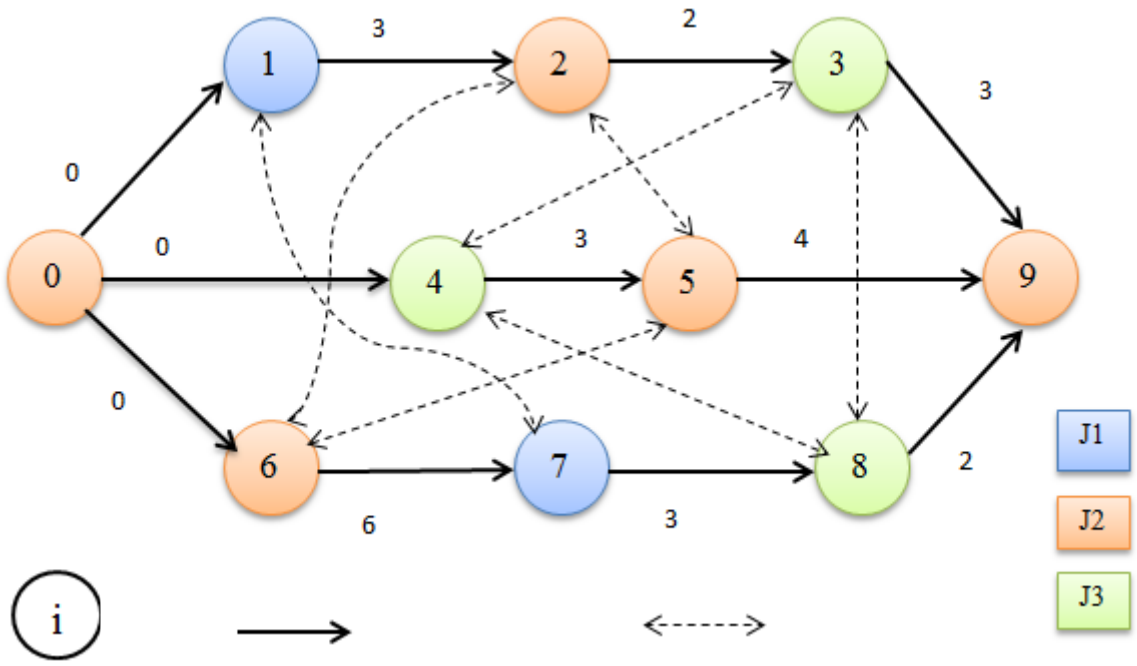


Figure 2.2 Graphe disjonctif d'un problème d'ordonnancement de type job-shop[14].

4 Complexité

Dans les problèmes d'ordonnancement, la complexité est liée à la complexité des méthodes de résolution et des algorithmes utilisés. La complexité de quelques types de problèmes d'ordonnancement dotés d'une taille relativement grande devient importante [10].

La complexité d'ordonnancement peut être divisée en deux grandes catégories : algorithmique et problématique.

4.1 La complexité algorithmique

L'objectif de la théorie de complexité est d'analyser les coûts de résolutions surtout en termes de temps de calcul. Elle vise aussi à classifier les problèmes en plusieurs niveaux de difficulté. Une étude a prouvé que les problèmes d'ordonnancement sont des problèmes difficiles.

4.2 La complexité problématique

La complexité problématique est relative au problème à résoudre ainsi que la méthode de résolution adoptée pour élaborer la solution optimale par rapport aux critères retenus.

- Un problème de décision comprend deux parties : une partie donnée du problème et un processus binaire ayant « oui » ou « non » comme réponse possible.

- Un problème de recherche est un problème constitué d'un ensemble de données dont chacun

représente un ensemble de solutions. Donc, la résolution d'un problème de recherche consiste à trouver pour chaque ensemble de données D des solutions S associées. Un problème d'optimisation est un problème de recherche en associant à chaque solution une valeur qualitative. À chaque problème d'optimisation, on peut associer un problème de décision (par exemple l'exclusion ou l'inclusion d'une solution dans les futures générations pour les AG), donc l'étude de la complexité du problème de décision peut donner des indications au problème d'optimisation associé [10].

La théorie de complexité permet de classer les problèmes en deux classes P et NP. La classe P regroupe les problèmes qui peuvent être résolus 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. Les algorithmes dont la complexité ne peut pas être bornée polynomialement sont qualifiés d'exponentiels et correspondent à la classe NP [10].

Un problème de décision est dit NP-Complet s'il appartient à la classe NP et il est résolu, au mieux, en un temps exponentiel.

Un problème d'optimisation est dit NP-Difficile, si le problème de décision associé est NP-complet.

4.3 Complexité du problème d'ordonnancement job shop

Selon [3], Les problèmes de job shop sont en général NP-complets, même si l'atelier est simple. En effet, Lenstra et al ont montré que les ateliers possédant plus de trois machines, ou un nombre de taches supérieur ou égal à trois, sont NP-difficiles même si la préemption est permise.

De même, pour les problèmes a deux machines, dès qu'il y a recirculation, ils deviennent fortement NP-difficiles. L'expérience a montré que les problèmes de job shop ayant un nombre m >2 machines, et optimisant les critères Cmax et F sont NP-difficile au sens fort, même en utilisant des heuristiques.

Comme 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 39594×1065 solutions différentes (par comparaison, l'âge de la Terre ne dépasse pas 1018 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.

Pour résoudre un problème d'ordonnancement de manière efficace, il faut prendre en compte la particularité de chaque problème. C'est pourquoi il faut s'assurer de la classe de complexité associe au problème a ordonnancer. Il existe plusieurs méthodes de résolution du problème d'ordonnancement, la connaissance de la classe de complexité peut nous aider énormément quant au choix judicieux de la méthode de résolution.

5 Les méthodes de résolution du problème d'ordonnancement

Au fil des années, de nombreuses méthodes de résolution de problèmes ont été proposées. Ainsi, une grande variété de concept et principe, de la stratégie et des performances ont été discernées. Cette variété et ces différences ont permis de regrouper les différentes méthodes de résolution de problèmes NP-difficiles en deux classes principales : la classe de méthodes exactes et la classe des méthodes approchées [13].

5.1 Méthodes exactes

Ces méthodes sont généralement utilisées pour résoudre des problèmes de petite taille. Dans ce cas, le nombre de combinaisons possibles est suffisamment faible pour pouvoir explorer l'espace de solutions en un temps raisonnable. On distingue trois sous-classes de méthodes exactes [10] : la procédure de séparation et d'évaluation (Branch and Bound), la programmation dynamique et la programmation linéaire.

5.2 Méthodes approchées

La résolution d'un problème d'optimisation combinatoire, de taille comparable à ceux rencontrés dans la pratique, se heurte à des tailles mémoire et des temps de calcul trop importants. L'objectif n'est plus alors d'obtenir systématiquement l'optimum mais plutôt

d'obtenir une solution proche de l'optimum ou de « bonne qualité » en un temps minimal. Ainsi, au lieu d'effectuer une recherche exhaustive, les méthodes approchées échantillonnent l'espace de recherche, n'en considèrent qu'une partie, et fournissent ainsi, en un temps raisonnable, la meilleure configuration rencontrée.

On distingue deux types de méthodes : les heuristiques et métaheuristiques [10].

5.2.1 Les heuristiques

Les heuristiques sont des méthodes empiriques basées sur des règles simplifiées pour optimiser un ou plusieurs critères. Le principe général de ces méthodes est d'intégrer des stratégies de décision pour construire une solution proche de l'optimum, tout en essayant de l'obtenir en un temps de calcul raisonnable [13].

On distingue:

- **FIFO** (First In First Out) : la première tâche qui vient est la première tâche ordonnancée,

- **SPT** (Shortest Processing Time) : la tâche ayant le temps opératoire le plus court est traitée en premier lieu,

- **LPT** (Longest Processing Time) : la tâche ayant le temps opératoire le plus important est ordonnancée en premier lieu,

- **EDD** (Earliest Due Date) : la tâche ayant le date due la plus petite est la plus prioritaire,

- **SRPT** (Shortest Remaining Processing Time) : cette règle, servant à lancer la tâche ayant la plus courte durée de travail restant à exécuter, est très utilisée pour minimiser les encours et dans le cas des problèmes d'ordonnancement préemptifs,

- **ST** (Slack Time) : à chaque point de décision, l'opération ayant la plus petite marge temporelle est prioritaire. Faute de disponibilité des ressources de production, cette marge peut devenir négative.

5.2.2 Les métaheuristiques

Une métaheuristique est un processus itératif qui subordonne et guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l'espace de recherche. Des stratégies d'apprentissage sont utilisées pour structurer l'information afin de trouver efficacement des solutions optimales, ou presque-optimales. L'ensemble des

métaheuristiques proposées dans la littérature sont partagées en deux catégories : des métaheuristiques à base de solution unique et des métaheuristiques à base de population de solutions.

- **Les Métaheuristiques à base de solution unique**

Les métaheuristiques à base de solution unique débutent la recherche avec une seule solution initiale. Elles se basent sur la notion du voisinage pour améliorer la qualité de la solution courante. En fait, la solution initiale subit une série de modifications en fonction de son voisinage. Le but de ces modifications locales est d'explorer le voisinage de la solution actuelle afin d'améliorer progressivement sa qualité au cours des différentes itérations.

De nombreuses méthodes à base de solution unique ont été proposées dans la littérature, le recuit simulé, la recherche tabou. [10]

- **Les métaheuristiques à base de population de solutions**

Les métaheuristiques à base de population de solutions débutent la recherche avec une panoplie de solutions. Elles s'appliquent sur un ensemble de solutions afin d'en extraire la meilleure (l'optimum global) qui représentera la solution du problème traité. L'idée d'utiliser un ensemble de solutions au lieu d'une seule solution renforce la diversité de la recherche et augmente la possibilité d'émergence de solutions de bonne qualité.

Une grande variété de métaheuristiques basées sur une population de solutions a été proposée dans la littérature, algorithmes génétiques, et les algorithmes à base d'intelligence par essaims : l'algorithme d'optimisation par essaim de particules, l'algorithme de colonies de fourmis [10].

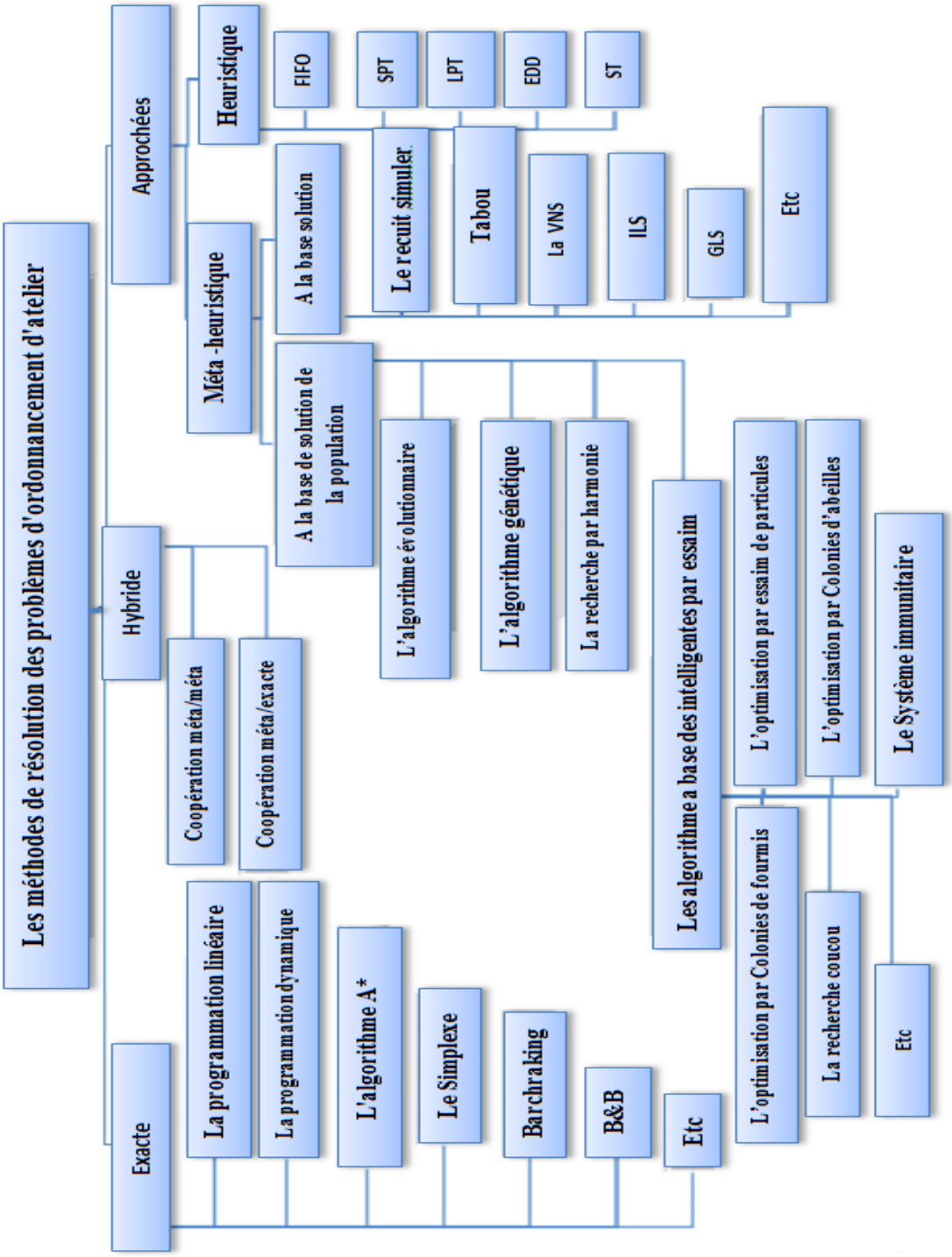


Figure 2.3 Classification des méthodes de résolution des problèmes d'ordonnancement d'atelier tirée de [13].

6 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 possédant pour le test d'une multitude de méthodes généralement les nouvelles approches de résolution, comme les métaheuristiques, qui font le sujet de notre application, et qui sont présentées dans le prochain chapitre.

CHAPITRE 3

LES METAHEURISTIQUES

1 Introduction

Nous présentons dans ce chapitre le principe fondamental de deux métaheuristiques qui constituent l'objet de ce mémoire : les Algorithmes Génétiques 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 deux Méthodes retenues en précisant notamment les principes généraux et le fonctionnement de chacune. La dernière section on a présenté l'implémentation des deux méthodes sur le problème de job shop.

I.1 Les métaheuristiques

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

« 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 » [3].

2 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 [15]. Cette section présente le modèle de base des Algorithmes Génétiques, puisque il existe plusieurs variantes de la méthode qui engagent des mécanismes plus complexes.

2.1 Principes généraux des Algorithmes Génétiques

Pour utiliser un Algorithme Génétique pour un problème particulier, on doit donc disposer des cinq éléments suivants [15] :

▪ **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.

▪ **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 ce dessus [3] :

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 : Principe de l'Algorithme Génétique.

2.2 Codage

Les algorithmes génétiques sont appliqués sur une population d'individus, chacun de ces derniers est codé par un chromosome ou génotype. Donc, une population est présentée par un ensemble de chromosomes. Le processus de codage d'individus consiste à trouver une structure de données pour les individus.

On distingue plusieurs types de codage, tels que : le codage binaire, le codage par permutation de valeurs entières et le codage par valeur, etc. [15] [3].

2.2.1 Le codage binaire

Les premiers algorithmes génétiques ont utilisé le codage binaire. Un chromosome est représenté par une chaine binaire (chaine de bits) précisant l'information nécessaire à la description d'un individu [12].

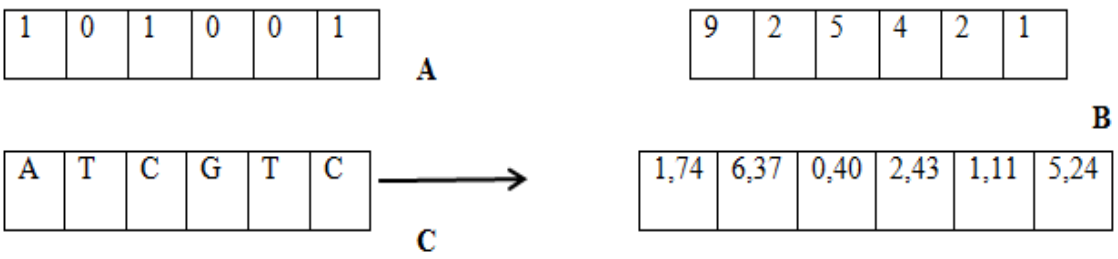
2.2.2 Le codage par valeurs entières

Chaque gène est codé par une valeur entière. Donc, un chromosome est représenté sous forme d'une chaine d'entiers.

Ce codage est bien adapté à l'optimisation des problèmes industriels réels [12].

2.2.3 Le codage par valeur

Dans ce type de codage, chaque gène est codé par une valeur qui appartient à un ensemble fini ou infini. Ces valeurs sont liées au problème à résoudre [12].



(A) : codage binaire,

(B) : codage par permutation de valeurs entières,

(C) : codage par valeur.

Figure 3.1 : Exemples de codage par valeurs tiré de [12].

2.3 Le croissement

Le croissement utilisé par les algorithmes génétiques est la transposition informatique du mécanisme qui permet, dans la nature, la production de chromosomes qui héritent partiellement des caractéristiques des parents.

Son rôle fondamental est de permettre la recombinaison des informations présentes dans le patrimoine génétique de la population.

En effet, plus le nombre de points de croisements sera grand et plus la probabilité de croisement sera élevée plus il y aura d'échange de segments, donc d'échange de paramètres, d'information, et plus le nombre de points de croisements sera petit et plus la probabilité de croisement sera faible, moins le croisement apportera de diversité [13].

3.3.1 Croisement avec à un point simple

Le croisement un point consiste à choisir un seul point de coupure, puis échanger les fragments situés après ce point de coupure [13].

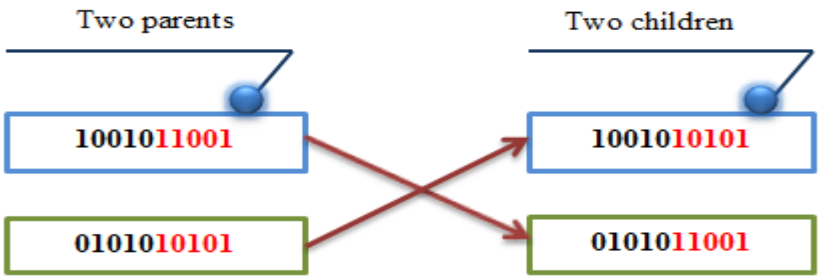


Figure 3.2 Croisement avec à un point simple [13].

3.3.2 Croisement uniforme

Ce type de croisement est fondé sur la probabilité. En fait, il permet la génération d'un enfant en échangeant chaque gène des deux parents avec une probabilité égale à 0.5

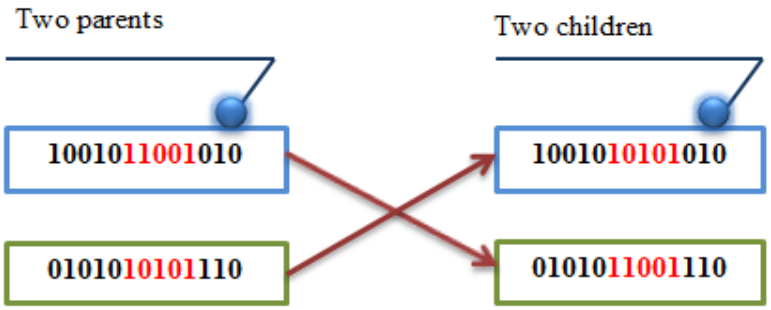


Figure 3.3 Croisement uniforme [13].

L'opérateur de croisement assure donc le brassage du matériel génétique et l'accumulation des mutations favorables. En termes plus concrets, cet opérateur permet de créer de nouvelles combinaisons des paramètres des composants. Malgré tout, il est possible que l'action conjointe de la sélection et du croisement ne permette pas de converger vers la solution optimale du problème [13].

2.4 La mutation

L'opérateur de mutation, assure le brassage et la recombinaison des gènes parentaux, permettant de diversifier la future population afin d'éviter une convergence rapide vers une solution optimale localement.

Étant aléatoire, cet opérateur agit selon une probabilité Pc fixée par l'utilisateur en fonction du problème à optimiser [12].

On trouve plusieurs stratégies de mutation [15] :

- **La mutation uni-point**

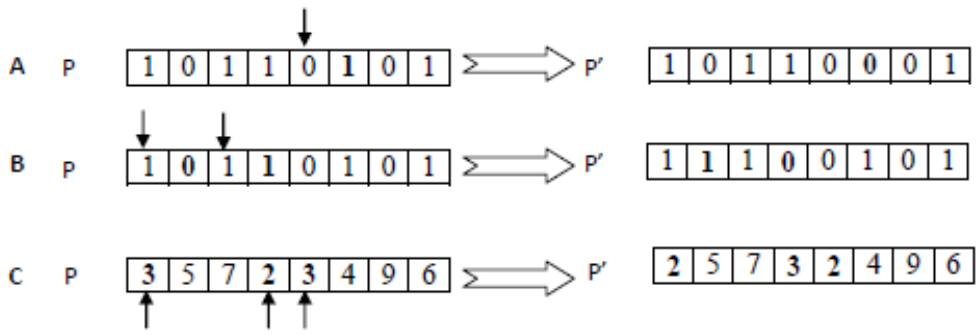
Ce type de mutation se fait par altération d'une seule valeur sur le chromosome.

- **La mutation bipoints et multipoints**

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

- **La mutation par valeurs**

La mutation par valeur se fait par transformation d'une valeur donnée en une autre valeur déterminée, sur tous les gènes du chromosome.



A : Mutation uni-point. B : Mutation bipoints. C : Mutation par valeurs 3 et 2.

Figure 3.4 Exemple d'opérateurs de mutation tiré de [11].

2.5 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 [3].

On trouve dans la littérature un grand nombre de stratégies de sélection. Les plus important parmi ces stratégies sont [15] :

3.5.1 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.

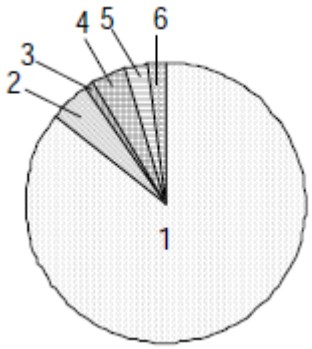


Figure 3.5 Sélection par la "roue de fortune" tirée de[3].

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 [15] :

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

3.5.2 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.

3.5.3 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.

3.5.4 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.

3.5.5 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.

I.3 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 [3].

4.1 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 [16].

4.2 Processus et ingrédients de la méthode

Selon [16], 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 [16].

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 [16] :

- 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 [3].

L'algorithme 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 $\check{Z} = s_0$ et $T_i = T_0$;
3. Calculer $f(s_0)$;
   a. Choisir $s' \in N(s_i)$;
   b. Calculer $\Delta f = f(s') - f(s_i)$;
   c. Si $\Delta f < 0$ Alors $\check{Z} = s_i$;
   d. **Sinon**
      Début
         tirer p dans [0, 1] suivant une distribution uniforme ;
      Si $p \leq e(-\Delta f / T)$: Alors $\check{Z} = s_i$;
         Sinon s_i est rejetée ;
      Fin ;
   e. Calculer $T = \gamma T$;
5. Retourner $\check{Z}$.

Fin.

Algorithme Pseudo code du Recuit Simulé [3].

II Application des métaheuristiques au problème d'ordonnancement de Job Shop

1 Application des Algorithmes Génétiques

1.1 Génération de la Population Initiale

Plusieurs méthodes existent pour définir le mécanisme de la génération de la population initiale qui représente le point de départ pour la constitution des générations futures : le tirage aléatoire, les heuristiques ou une combinaison de solution heuristique et aléatoire

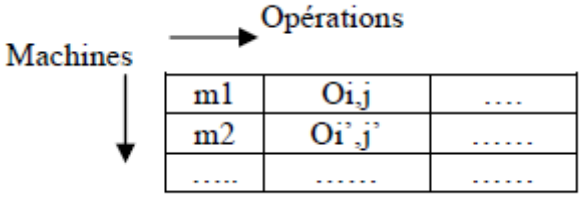


Tableau 3.1 Représentation du codage d'un individu i[17].

Une solution ou l'individu i qui compose cette population est codée par une matrice de séquence d'opérations qui définie l'ordre des opérations que doit effectuer chaque machine comme le montre la table 1. Les solutions constituantes la population initiale sont choisies aléatoirement dans cette étude [17].

Algorithme génération de la population initiale

Début
Fixer la taille n de la population initiale.
Générer une première solution.
k=1
Tant que « k ≠ n» **Faire**
Tester la réalisabilité :
Si « solution sans cycle » **Alors**
Si« solution non existante » **Alors**
- Solution k acceptée (solution réalisable).
k = k + 1.
Sinon
- Solution refusée (solution existe déjà).
Fin si ;
Sinon
- Solution refusée (solution non réalisable).
Fin si ;
Générer une autre solution.
Fin tant que ;
Fin.

1.2 Sélection

La phase de sélection consiste à choisir parmi les N individus i de la population courante les plus forts individus à partir desquels la génération suivante sera créée. Soit pour un ordonnancement i réalisable, on calcul la valeur de la fonction objectif fi « Makespan » pour notre problème c'est évaluer le plus long chemin sur le graphe disjonctif. On associe à chaque solution une fonction d'évaluation pour calculer sa force d'adaptation Fi, dans notre cas le problème à pour but de minimiser la fonction objectif min f (i), alors la force de l'individu i (fitness) peut être exprimer tout simplement par Fi = 1/fi, Les individu ainsi sélectionnés constituent une population intermédiaires [17].

Algorithme de sélection

Début
pour i = 1 jusqu'à N **Faire**
Générer un individu i (ordonnancement réalisable).
Calculer Makespan _ individu i.
Calculer fitness_individu i.
Rangement.
Calculer Probabilité_sélection Pi
Générer un nombre x aléatoirement x ∈ [0, 1].
Six < Pi **alors**
Sélectionner l'individu i.
Fin si,
Fin pour,
Fin.

Il existe différente stratégie de sélection, tel que la sélection par la Roue de Loterie (Roulette Wheel Sélection) ou La sélection par la méthode de Tournois.

Dans cette étude nous avons utilisé la méthode de sélection par Rang (Rankine) de classement : on affecte à chaque individu un rang *Ri* qui dépend de son fitness, les individus d'une population sont ensuite classés dans une liste selon l'ordre croissant de leur fitness, on associe pour chaque individu i une probabilité Pi d'être sélectionné, la sélection sera proportionnelle à leur rang dans la liste de la population [17] :

$$p_i = R_i / \sum_{i=1}^{N} R_i$$

1.3 Croisement

L'opérateur de croisement est le plus important dans les AGs, il permet d'explorer efficacement l'espace de recherche. L'opérateur de croisement appliquer sur deux individu parent1 et parent2 choisis parmi la population sélectionnée, permet de générer deux nouvelles solutions enfant1 et enfant2 par combinaison des propriétés des parents 1 et 2 [17].

Algorithme de croisement

Début
Sélectionner aléatoirement deux parents indiv1 et indiv2.
Sélectionner aléatoirement une machine k parmi m.
Croisement :
Pour « i = 1 jusqu'à m » **Faire**
Si « i ≠ k » **Alors**
enf1 reçoit les mêmes affectations que indiv1.
enf2 reçoit les mêmes affectations que indiv2.
Sinon
enf1 reçoit les affectations de indiv2.
enf2 reçoit les affectations de indiv1.
Fin si ;
Fin de pour ;
Tester la réalisabilité :
Si « enf1, enf2 sans cycle » **Alors**
- enf1 et 2 acceptés (solutions réalisable)
Sinon
- enf1 et 2 refusés (solutions non réalisables)
Fin si ;
Fin .

La méthode implémenté dans cette étude consiste à choisir aléatoirement une machine k et on applique un croisement entre la séquence des opérations associé à cette machine seulement, le séquencement des opérations sur les autres machines reste identique sur les fils, puis on teste la réalisabilité des solutions obtenu, ceux qui ne comporte pas de cycle seront acceptées [17].

M1	$O_{2,2}$	$O_{1,1}$	$O_{3,3}$
M2	$O_{3,1}$	$O_{2,3}$	$O_{1,2}$
M3	$O_{1,3}$	$O_{2,1}$	$O_{3,2}$

M1	$O_{1,1}$	$O_{2,2}$	$O_{3,3}$
M2	$O_{2,3}$	$O_{1,2}$	$O_{3,1}$
M3	$O_{2,1}$	$O_{1,3}$	$O_{3,2}$

Tableau 3.2 Représente les des parents indiv1, indiv2 sélectionner pour le croisement sur la machine 2 [17].

M1	$O_{2,2}$	$O_{1,1}$	$O_{3,3}$
M2	$O_{2,3}$	$O_{1,2}$	$O_{3,1}$
M3	$O_{1,3}$	$O_{2,1}$	$O_{3,2}$

M1	$O_{1,1}$	$O_{2,2}$	$O_{3,3}$
M2	$O_{3,1}$	$O_{2,3}$	$O_{1,2}$
M3	$O_{2,1}$	$O_{1,3}$	$O_{3,2}$

Tableau3.3 Les enfants *enf1 et enf2* obtenu après croisement [17].

1.4 Mutation

Le deuxième opérateur génétique important est la mutation, elle vient en deuxième place sur le plan d'importance par rapport au croisement. Une mutation est une perturbation introduite sur la composante de l'individu afin de garantir la diversité et élargir le champ d'exploration.

La démarche suivie dans ce travail consiste à choisir aléatoirement un individu et la machine qui doit subir cette perturbation, ensuite on choisit aléatoirement deux opérations, l'opération consiste à permuter l'ordre entre eux [17].

Algorithme de mutation

Début
Sélectionner aléatoirement un individu i.
Sélectionner aléatoirement une machine k parmi m.
Pour « h = 1 jusqu'à m » **Faire**
Si « h ≠ k » **Alors**
Indiv muté reçoit les mêmes affectations que indiv i.
Sinon
Sélectionner aléatoirement deux opérations $O_{i,j}$ et $O_{i',j}$
Permuter l'ordre des opérations $O_{i,j}$ et $O_{i',j'}$.
Fin si ;
Fin de pour ;
Tester la réalisabilité :
Si « solution sans cycle » **Alors**
Indiv muté acceptés (solution réalisable)
Sinon
Indiv muté refusés (solutions non réalisable)
Fin si ;
Fin.

M1	$O_{2,2}$	$O_{1,1}$	$O_{3,3}$
M2	$O_{3,1}$	$O_{2,3}$	$O_{1,2}$
M3	$O_{1,3}$	$O_{2,1}$	$O_{3,2}$

Tableau 3.4 Individu i avant permutation sur la machine 2 [17].

M1	$O_{2,2}$	$O_{1,1}$	$O_{3,3}$
M2	$O_{2,3}$	$O_{3,1}$	$O_{1,2}$
M3	$O_{1,3}$	$O_{2,1}$	$O_{3,2}$

Tableau 3.5 Individu i après permutation [17].

1.5 Remplacement

Cette étape constitue la population de la génération suivante à partir des parents et des enfants de la génération courante. Une fraction de la population est remplacée par sa descendance à chaque génération. L'écart entre les générations indique la proportionnel de parents remplacés par des enfants [17].

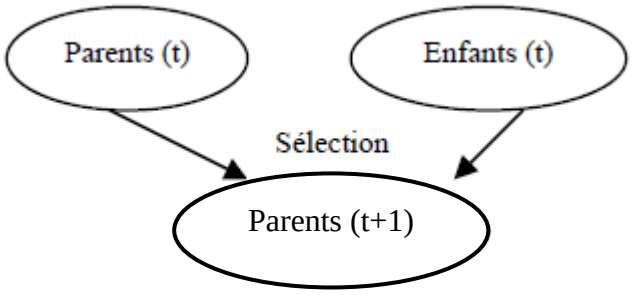


Figure3.6 Le Remplacement, cette phase permet de ne retenir que les meilleurs éléments entre enfants et parents.

Algorithme de remplacement

Début
Rangement de la population des parents.
Rangement de la population enfants.
Remplacement :
Choisir les N meilleurs individus parmi les enfants et
les Parents.
Fin.

1.6 Critère d'arrêt

Le processus est stoppé au bout d'un nombre fixé de génération, ou lorsque les individus ont convergé vers une ou plusieurs solutions satisfaisantes, dans notre étude nous avons opté pour la première démarche.

Les meilleurs individus de la population sont alors retenus comme solutions au problème [17].

2 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.

2.1 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 [3].

2.1.1 La Température Initiale

Certains auteurs font intervenir des procédures complexes pour déterminer la Température Initiale [18]. Dans notre application, l'initialisation de la Température peut se faire, soit par des valeurs arbitraires, soit empiriquement, après une série de tests.

2.1.2. 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 [18].

2.1.3 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 :

- $T_k = T_{0e^{-ck}}$ 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: estla Température Finale.

- T_k = γ T_{k-1}avec : γ ∈ [0.85, 0.99]. Dans le cas où la Température Finale est laissée indéterminée [18].

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 [3].

2.2La 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 [18]:

g (s') = 1/n où s' ∈ N(s), n : 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 :

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

2.3 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 uneprobabilité plus grande d'accepter les dégradations.

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

2 Conclusion

La présentation générale menée au cours de ce chapitre sur les Algorithmes Génétiques, le Recuit Simulé portant sur leur définition et l'explication de leur principe de fonctionnement a permis de faire ressortir l'idée de base de chacune et nous avons présenté aussi l'implémentation des métaheuristiques (Algorithmes Génétiques, Recuit Simulé) en les appliquant à la résolution du problème de Job Shop.

CHAPITRE 4

ETUDE EXPERIMENTALE ET COMPARATIVE

1 Introduction

Dans ce chapitre, nous avons commencé par présenter les outils et environnement de développement que nous avons utilisé. Ensuite, on a présenté l'application. Après Nous avons procédé quelque exemples pour tester, ces tests sont réaliser sur des données sont changer à chaque fois (nombre de job, nombre de machine). Les tests sont réalisés avec les deux méthodes déjà vu dans le chapitre 3, avec l'algorithme génétique et recuit simulé. Enfin on a parlé sur les résultats obtenu avec une petite comparaison entre les deux méthodes.

2 Environnement matériel:

Pour développer l'application, nous avons utilisé comme environnement matériel un ordinateurs acer qui possède comme caractéristiques :

Un processeur Intel Pentium® Core (TM) i3, 1.80 GHz.

Une mémoire vive de 4Go.

Un disque dur 450 Go.

3 Environnement logiciel

Plateforme utilisé est Windows 7 professionnel pack1 .le langage de développement choisit est C# Sharp.

3.1 Le langage de programmation "C#/Sharp":

C# est un langage de programmation orientée objet, fortement typé, dérivé de C et de C++, ressemblant au langage Java. Il est utilisé pour développer des applications web, ainsi que des applications de bureau, des services web, des commandes, des widgets ou des bibliothèques de classes. En C#, une application est un lot de classes où une des classes comporte une méthode Main, comme cela se fait en Java [19].

3.2 L'environnement Microsoft Visual Studio

Microsoft Visual Studio est un environnement de développement intégré (IDE) de Microsoft. Il peut être utilisé pour développer des applications d'interface utilisateur console et graphique avec les applications Windows Forms, les sites Web, les applications Web et les services Web dans le code natif avec le code géré pour toutes les plates-formes prises en charge par Microsoft Windows, Windows Phone, Windows CE, NET Framework, NET Compact Framework et Microsoft Silver light.

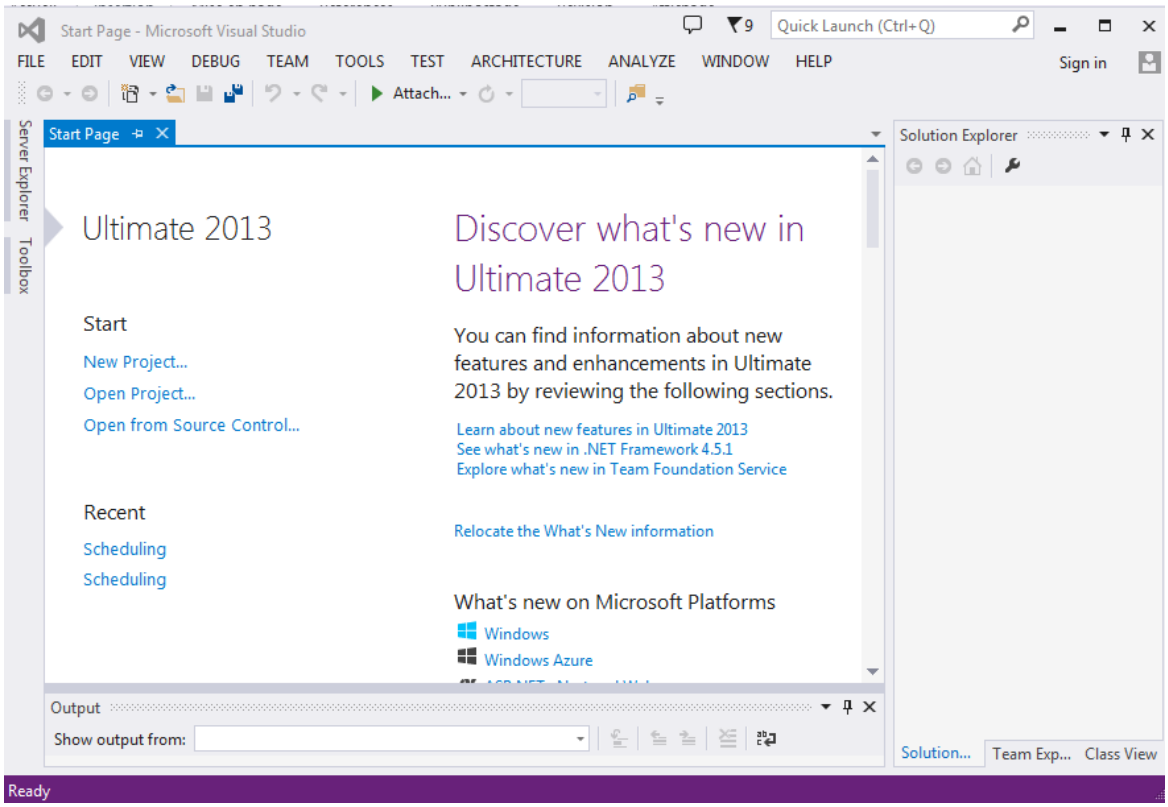


Figure 4.1 Visual Studio Professional 2013

4 Les résultats quelque exemples sur l'implémentation

4.1 Exemple 1

Le tableau 4.1 présente un problème de type de job shop composé de 3 jobs et 3 machines.

Au-dessous de est un exemple simple de problème de job shop, dans lequel chaque tâche est étiquetée par une paire de nombres (m, p) où m est le numéro de la machine sur laquelle la tâche doit être traitée et p le temps de traitement de la tâche

Job 1 = [(1, 3), (2, 3), (3, 3)].

Job 2 = [(1, 2), (3, 3), (2, 4)].

Job 3 = [(2, 3), (1, 2), (3,1)].

Dans l'exemple, le job 1 comporte trois tâches. Le premier, (1, 3), doit être traité sur la machine 1 dans 3 unités de temps. Le second (2, 3) doit être traité sur la machine 2 en 3 unités de temps, et ainsi de suite. Au total, il y a neuf tâches.

Numéro de job	Type de machine	Temps d'exécution
1	1	3
	2	3
	3	3
2	1	2
	3	3
	2	4
3	2	3
	1	2
	3	1

Tableau 4.1. Un exemple du problème job shop 3 machines et 3jobs

Avec notre implémentation on a trouvé le makespan avec le recuit simulé :

Cmax = 12

Le diagramme présenté dans la figure 4.2 représente le diagramme de Gantt d'une solution obtenu après l'application de notre algorithme de recuit simulé sur le problème 3x3.

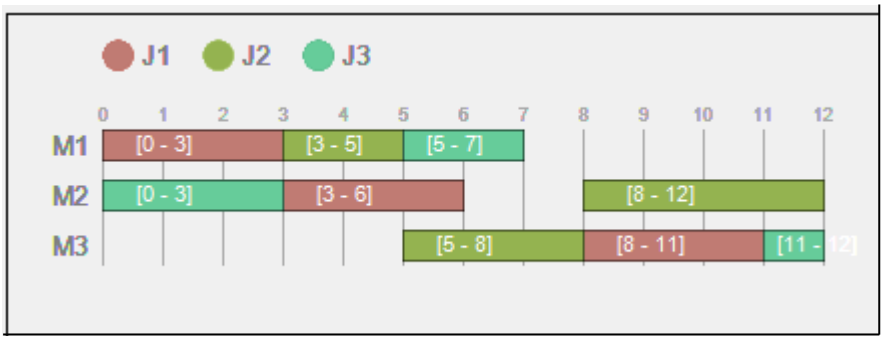


Figure 4.2 Exemple de Diagramme de Gant obtenu par recuit simulé de problème 3x3.

On a trouvé la solution du même exemple en algorithme génétique le makespan :

Cmax = 11

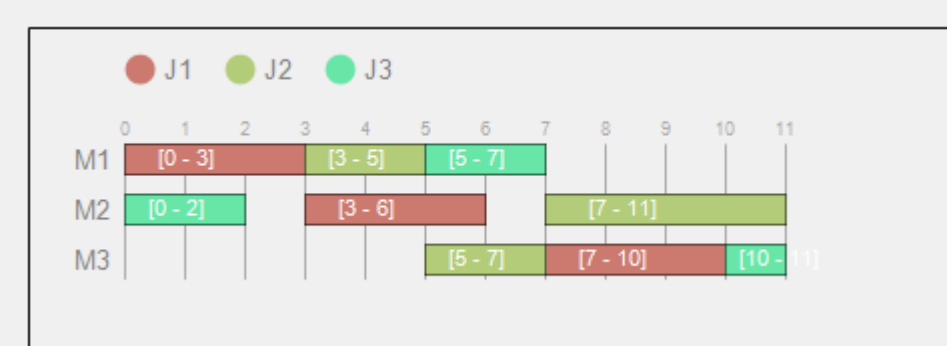


Figure 4.3 Exemple de Diagramme de Gant obtenu par Algorithme Génétique de problème 3x3.

4.2 Exemple 2

Le tableau 4.2 présente un problème de type de job shop composé de 3 jobs et 5 machines.

Numéro de job	Type de machine	Temps d'exécution
1	1	3
	2	3
	3	3
2	1	2
	3	3
	2	4
3	2	3
	1	2
	3	1
4	3	1
	2	2
	1	2
5	2	3
	1	4
	3	2

Tableau 4.2 Exemple 2 du problème job shop 3 machines et 5jobs

Job 1 = [(1, 3), (2, 3), (3, 3)]

Job2 = [(1, 2), (3, 3), (2, 4)]

Job 3 = [(2, 3), (1, 2), (3,1)]

Job 4 = [(3, 1), (2, 2), (1, 2)]

Job5 = [(2, 3), (1, 4), (3, 2)]

La figure 4.3 représente le diagramme de Gantt d'une solution obtenu après l'application de notre algorithme de recuit simulé sur le problème 5x3.

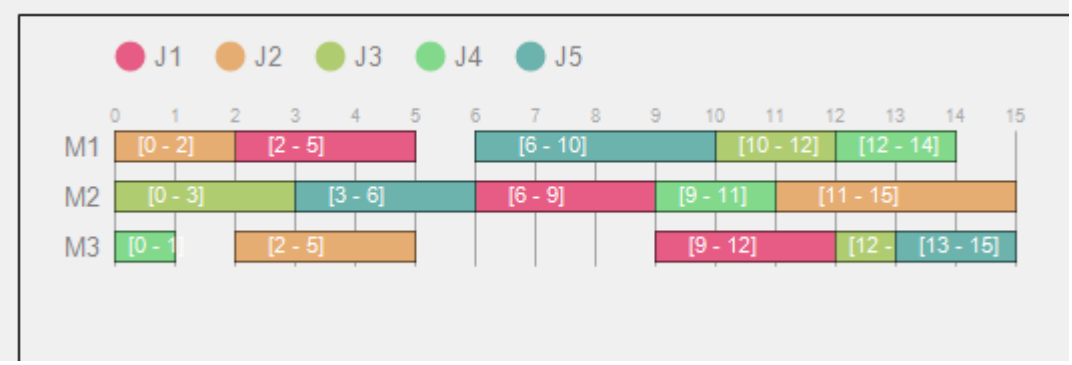


Figure 4.4 Exemple de Diagramme de Gant obtenu par recuit simulé de problème 5x3.

Aussi on a trouvé la même solution en l'algorithme génétique que le recuit simulé :

Cmax = 15

Et voici le diagramme de Gantt d'une solution obtenu par l'algorithme génétique :

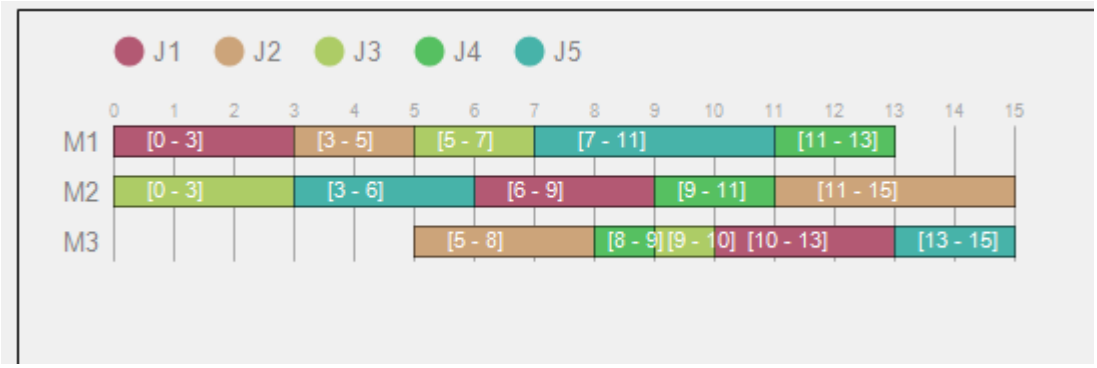


Figure 4.5 Exemple de Diagramme de Gant obtenu par Algorithme Génétique de probleme 5x3

4.3 Exemple 3

Le tableau suivant présente un problème de type de job shop composé de 10 jobs et 5 machines.

Numéro de job	Type de machine	Temps d'exécution
1	2	2
	3	3
	1	4
	5	4
	4	3
2	4	1
	1	5
	3	2
	2	3
	5	1
3	3	4
	4	6
	2	5
	1	2
	4	2
4	1	1
	5	7
	4	3
	2	4
	3	4
	3	2

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

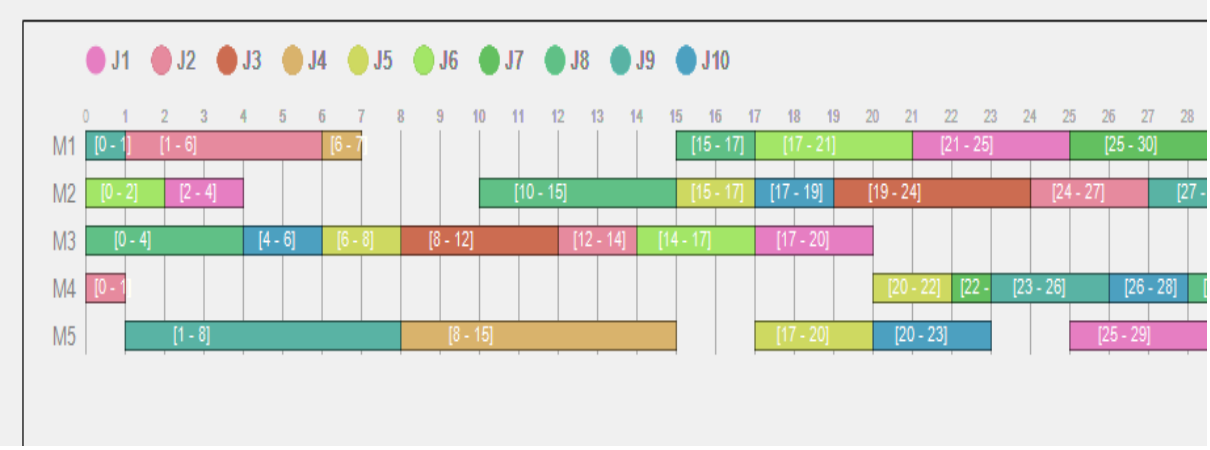
	4	3
	2	4
	3	4
10	3	2
	2	2
	5	3
	4	2
	1	2

Tableau 4.3 Exemple 3 du problème job shop 5 machines et 10 jobs

Avec notre implémentation on a trouvé le makespan avec le recuit simulé dons cette :

Cmax = 41

La figure 4.6 représente le diagramme de Gantt d'une solution obtenu après l'application de notre algorithme de recuit simulé sur le problème 5x10.



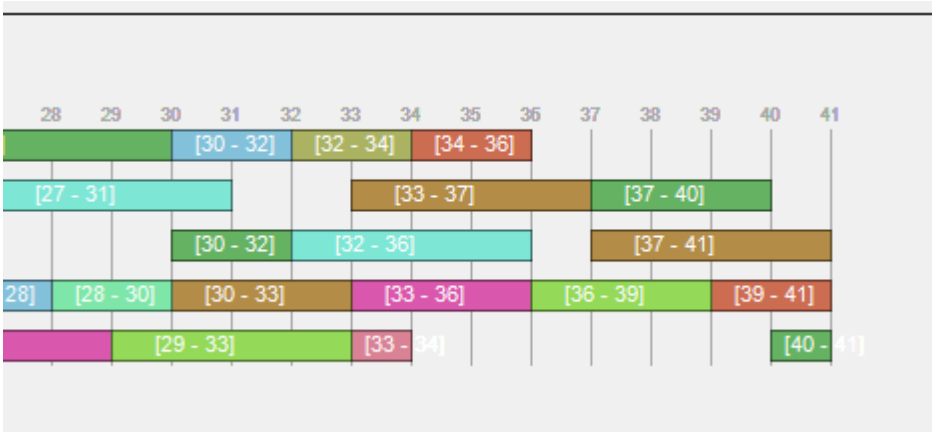
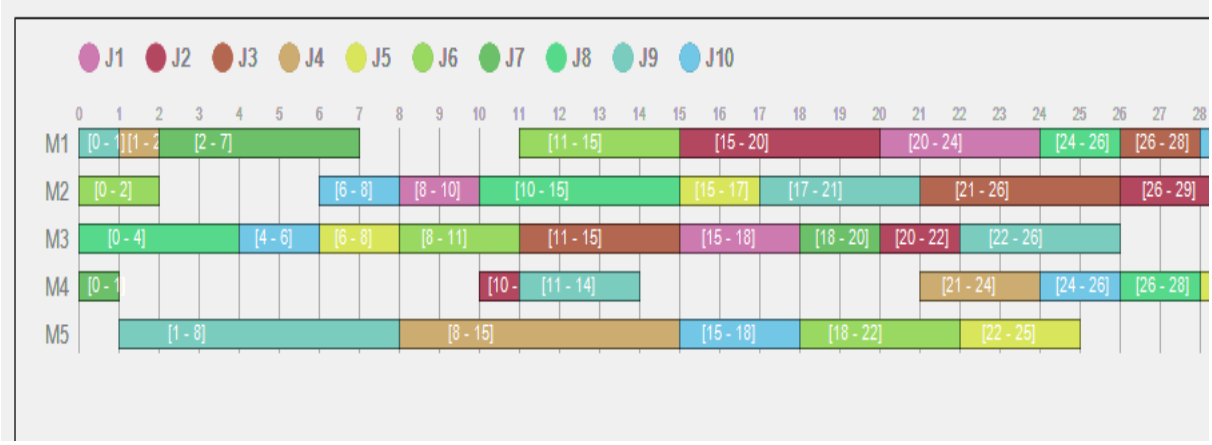


Figure 4.6 Exemple de Diagramme de Gant obtenu par recuit simulé de probleme 5x10

Dans ce problème de 5 machines et 10 jobs après l'implémentation on a trouvé le makespan avec l'algorithme génétique : **Cmax = 38**

La figure suivante présente le diagramme de Gantt d'une solution obtenu par l'algorithme génétique :



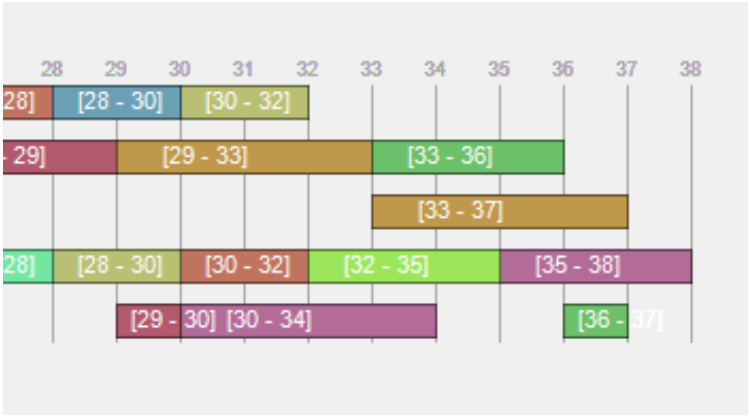


Figure 4.7 Exemple de Diagramme de Gant obtenu par l'algorithme génétique de problème 5x10

5 Comparaison

Après, les exemples qu'on a fait, nous avons trouvé défirent résultat de algorithme génétique et recuit simulé que nous avons implémenté sur le problème de job shop :

Dans le premier exemple on a trouvé le résultat d'algorithme génétique inferieur que le résultat de recuit simulé.

Le deuxième exemple on a trouvé les résultats de les algorithmes sont égaux.

Le troisième exemple on a trouvé le résultat d'algorithme génétique inferieur que le résultat de recuit simulé Comme le premier exemple.

Nous avons remarqué qu'Il n'y a pas de grande différence entre eux. Donc, l'algorithme génétique est meilleur que le recuit simulé, alors Nous concluons que les deux algorithmes sont capables pour résoudre ce probleme.

6 Quelques interfaces de l'application

Les figures suivantes ce sont Les déférentes interfaces existant dans l'application

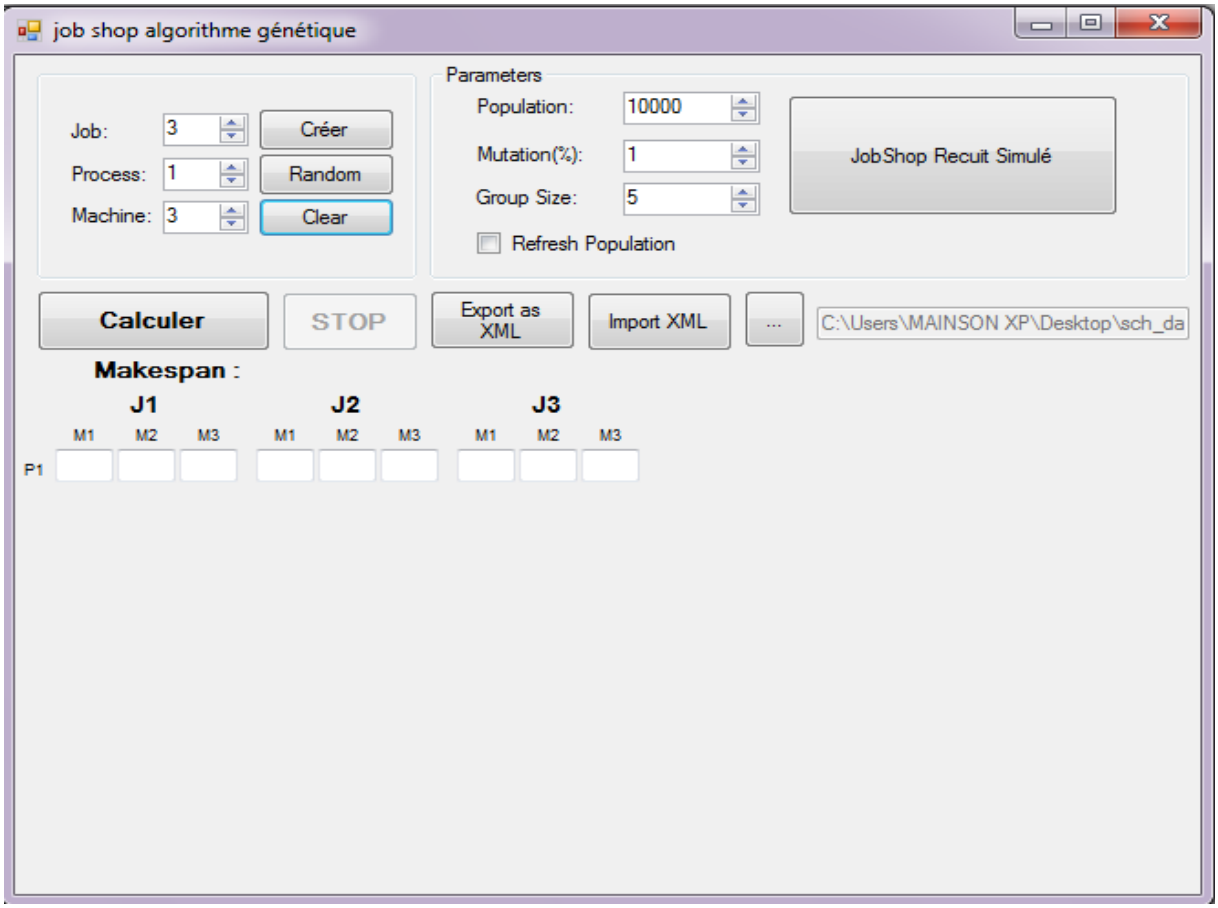


Figure 4.8 Interface principale de l'application

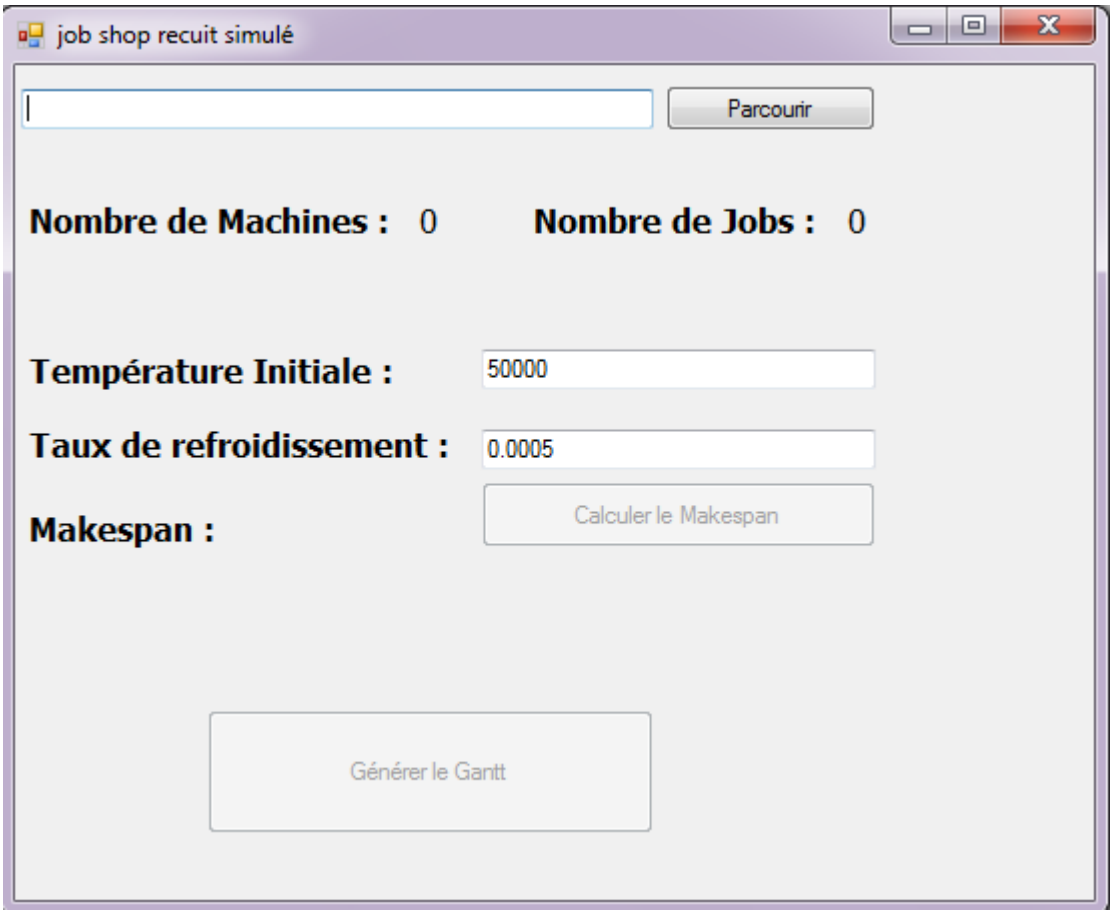


Figure 4.9 Interface de recuit simulé

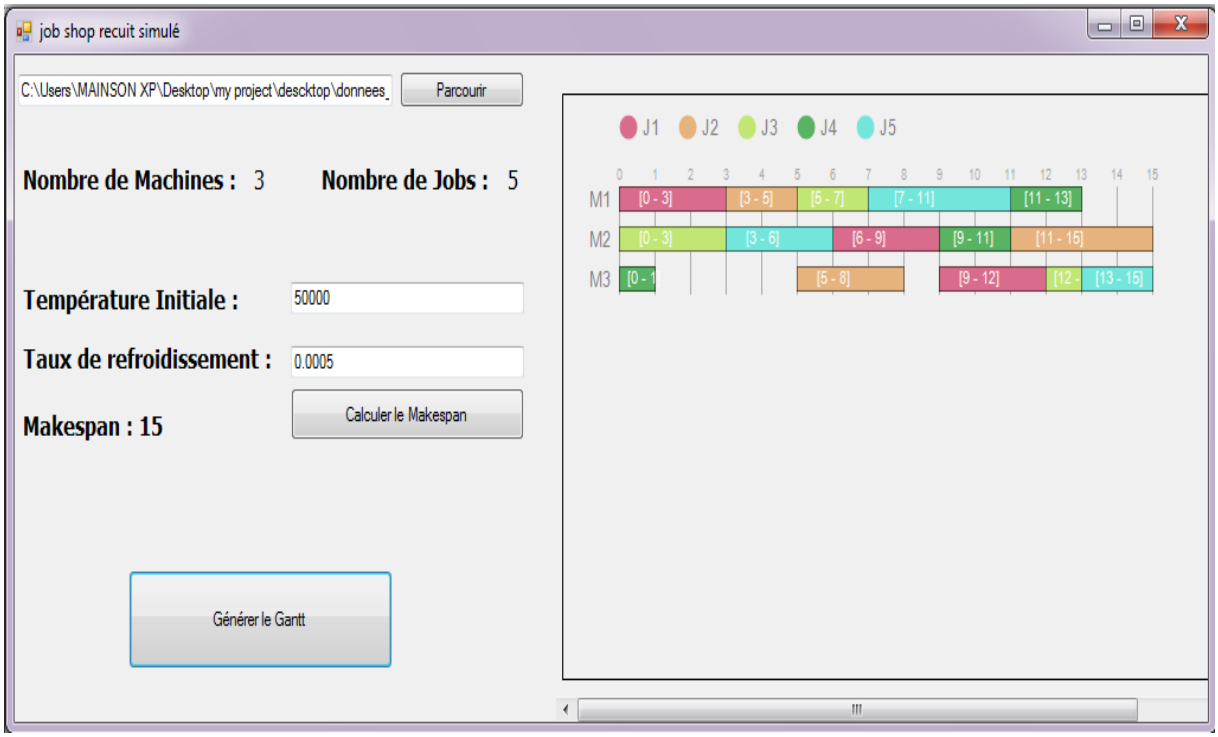


Figure 4.10 Exécution de exemple deux de recuit simulé

Conclusion

Dans le chapitre 4 nous avons présenté le logiciel développé, On a commencé par présenter l'environnement matériel et logiciel d travail réaliser, ensuit nous avons présenté notre étude comparative entre l'algorithme génétique et recuit simulé après nous avons discuté sur les résultats obtenus.

CONCLUSION GENERALE

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.

Dans notre travaille, nous avons développé une application qui minimise le Cmax (makespan, c'est le temps nécessaire pour terminer toues le taches) pour résoudre un problème job shop.

Le premier chapitre, on a étudié l'ordonnancement des activités de production en général, les définitions de l'ordonnancement et de ses éléments, les notations des problèmes d'ordonnancement et classification des ateliers (Job Shop, Flow Shop, Open Shop).

Dans le deuxième chapitre, nous avons détaillé sur le problème de job shop : la modélisation graphique, la complexité et les méthodes pour résoudre ce problème.

Le troisième chapitre, on a discuté sur les métaheuristiques l'algorithme génétique et le recuit simulé, ainsi que leur application dans le problème de job shop.

Le dernier chapitre, fait appel à une démarche expérimentale afin de montrer l'efficience des métaheuristiques utilisées dans la résolution du problème étudié. Les résultats obtenus sont discutés et analysés afin d'appartenir certaine conclusion.

REFERENCESBIBLIOGRAPHIQUES

[1] V.Giard, Gestion de la Production, 2ème édition, Economica, PARIS, 1988.

[2] G.Javel, Organisation et Gestion de la Production, 3ème édition, DUNOD, PARIS, 2004.

[3] K.Mebarek, Utilisation des stratégies Méta-heuristiques pour l'ordonnancement des ateliers de type Job Shop, Magister, BATNA, 2008.

[4] A.Letouzey, Ordonnancement interactif basé sur des indicateurs : Applications à la gestion de commandes incertaines et à l'affectation des opérateurs, Doctorat, TOULOUSE, 2001.

[5] J.VACHER 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, Doctorat, HAVRE, 2000.

[6] Jeffrey W. Herrman, & al. Handbook Of Production Scheduling, Springer NY, 2006.

[7] N. Mouhoub, Algorithmes de construction de graphes dans les problèmes d'ordonnancement de projet, Doctorat, SETIF, 2011.

[8] H.Hentous, contribution au pilotage des systèmes de production de type Job Shop, Doctorat, LYON, 1999.

[9] H.Boukef Ben Othman, l'ordonnancement d'ateliers job-shop flexibles et flow-shop en industries pharmaceutiques optimisation par algorithmes génétiqueset essaims particulaires, Doctorat, TUNIS, 2009.

10 Asma karay

[11] M. Meziane, Optimisation par phases pour le probleme d'ordonnancement des ateliers de type job shop totalement flexibles, Magister, ORAN, 2011.

[12] I. DRISS, analyse d'un système Job Shop aspect ordonnancement, Doctorat, BATNA ,2016.

[13] M.Pinedo, Scheduling Theory, Algorithms and Systems. Prentice-Hall, Inc. New Jersey, 1995.

[14] T. BENTRCIA Ordonnancement des systèmes de production sous contraintes technologiques ou contextuelles: Modélisation, étude de complexité et approches intégrées de résolution, Doctorat, BATNA, 2017.

[15] L. Randy Haupt & Sue Ellen Haupt, practical genetic algorithms, 2nd ed. John Wiley &

Sons, Inc, New Jersey, 2004.

[16] 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.

[17] Kh. Merhoum, M. Djeghaba , Algorithme génétique pour le problème d'ordonnancement de type job-shop , -LASA- , ANNABA.2007.

[18] 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.

[19] wikipedia,https://fr.wikipedia.org/wiki/C_sharp

ملخص

يتناولهذاالعملتطبيقفوقيةالاستدلاللمشكلةالجدولة.فيهذاالعملاقترحناستخداماالخوارزمياتالجينيةوخوارزميةمحاكاةالتلدينكطريقتينمقربتينلحلمسألةالجدولةجوبشوب.

المشكلةالتيتمتدراستهاهيجدولةجوبشوببسيطةبعدةالمهاموبعدةالات.الهدفهوتقليصمنالوقتالكليللتنفيذواستكشافوتبريرتطبيقفوقيةالاستدلالالقياسيةلهذهالمشكلة.

للقيامبذلكيتمتنفيذطرقمحددةلتطويرتطبيق (فوقيةالاستدلاللجدولةالجوبشوب).

يسمحهذاالتطبيقإجراءعدةسلسلةمنالتجاربللحصولعلىالنتائج.

Abstract

This work deals with the application of metaheuristics to the Job Shop scheduling problem. The methods selected are: Genetic Algorithms, and Simulated Annealing. The problem studied is the simple Job Shop n-tasks, m-machines. The objective is to minimize the makespan and to explore and justify the application of standard metaheuristics to this problem.

With this aim, an implementation of the methods selected according to several approaches is carried out, leading to the development of an application (Metaheuristcs for Job Shop Scheduling).

This application made it possible to carry out several series of experiments on implanted

Résumé

Ce travail traite l'application des Méta-heuristiques au problème d'ordonnancement des ateliers type Job Shop. Les méthodes retenues sont : les Algorithmes Génétiques, et le Recuit Simulé.

Le problème étudié est l'ordonnancement d'ateliers de type Job Shop simple, n-tâches, m-machines. L'objectif est de minimiser la date de fin de toutes les opérations et et d'explorer justifier l'application des Méta-heuristiques standards à ce problème.

Pour ce faire, une implémentation informatique des méthodes retenues selon plusieurs approches est réalisée, conduisant à développer une application (Metaheuristique pour ordonnancement de Job Shop).

Cette application a permis de réaliser plusieurs séries d'expérimentations portant sur choix implantés.

Mots clés : Ordonnancement, Job Shop, Méta-heuristiques, Algorithmes Génétiques, Recuit Simulé.