N° : 03 / 2012 – D / INF


**République Algérienne Démocratique et Populaire**
**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**
**Université des Sciences et de la Technologie Houari Boumediene**
**Faculté d'Electronique et Informatique**
**Département Informatique**



# THESE

**Présentée pour l'obtention du grade de DOCTEUR**
**en : Informatique**
**Spécialité : Informatique**
**Par**


**AitZai Abdelhakim**


**Sujet :**

# Ordonnancement du job shop avec contrainte de blocage

**Soutenue publiquement, le 05/03/2012 devant le Jury composé de :**

| | | | |
|---|---|---|---|
| **Mme H. DRIAS** | **Professeur** | **USTHB** | **Présidente** |
| **M. M. BOUDHAR** | **Professeur** | **USTHB** | **Directeur de Thèse** |
| **M. Dj. REBAINE** | **Professeur** | **UQAC, Canada** | **Examinateur** |
| **Mme F. SI TAYEB** | **M.C.A.** | **ESI, Alger** | **Examinatrice** |
| **M. A. BOUKRA** | **M.C.A.** | **USTHB** | **Examinateur** |
| **M. M. BOULIF** | **M.C.A.** | **U.M.B. Boumerdes** | **Examinateur** |

# Remerciements

Avant tout, je remercie Allah Tous Puissant…

Ensuite je remercie et très vivement mes très chers parents qui ont mis à ma disposition tout ce qu'il faut (amour, soutien, encouragement, moyens, etc.) depuis mon premier jour de l'école et sans cesse, à ce jour. Je tiens à remercier aussi ma très chère femme dont la présence, l'aide et le soutien ne peuvent être occultés.

Je tiens à exprimer mes sincères remerciements à mon directeur de thèse M. Mourad Boudhar pour la confiance qu'il m'a accordée, pour son accueil, sa disponibilité, ses conseils et son soutien tout au long de la préparation de cette thèse.

Je remercie aussi et très vivement les deux experts Mme Fatima Si Tayeb et M. Abdelmadjid Boukra qui ont accepté d'expertiser mon travail. Mes remerciements vont également aux membres du jury : Mme Habiba Drias du Département Informatique de l'USTHB, Mme Fatima SiTayeb de l'Ecole Supérieure d'Informatique, M. Menouar Boulif de l'université M'hamed Bougara de Boumerdes, M. Djamal Rebaine de l'Université du Québec à Chicoutimi et M. Abdelmadjid Boukra du Département Informatique de l'USTHB qui m'ont fait l'honneur de participer à la soutenance de cette thèse.

Je désire témoigner ma gratitude à Mme Malika Boukala-Ioualalen, M. Ahmed Guessoum, M. Abdelhakim Artiba, M. Djamal Rebaine, M. Rachid Benmansour pour leurs conseils, leurs accueils et leurs disponibilités à toutes mes sollicitations.

Je remercie très vivement, aussi, MM. Noureddine Bali, Larbi Benaissa, Ahmed chawki Benchikh El Hocine, Mohamed Kadouche, Mourad Heddi, Djamel Meguellati pour leurs conseils, leurs disponibilités et leur précieuse aide.

Je tiens aussi à remercier particulièrement Mme Habiba Drias, M. Yazid Mati et M. Mohamed Ahmed Nacer dont l'aide m'a été très bénéfique.

Que toute personne qui m'a apporté sont aide de prés ou de loin trouve ici mes sincères remerciements.

# Table des matières

**Introduction Générale** 1
**Chapitre 1 : Les fonctions Production & Ordonnancement** 4
1. Introduction 4
2. Fonction Production 5
2.1 Objectifs Associés 5
2.2 Niveaux Hiérarchiques de la Gestion de Production 6
3. Rôle de l'Ordonnancement en Gestion de Production 6
4. Définitions de Base 7
4.1 Gamme de Fabrication 7
4.2 Routage 7
4.3 Job (Tâche) 7
4.4 Ressource 8
4.5 Opération 8
4.6 Contraintes 8
4.7 Ordonnancement 9
5. Problèmes d'Ordonnancement d'Ateliers 9
6. Identification des problèmes d'ordonnancement 10
7. Conclusion 15
**Chapitre 2 : Théorie de Graphes et Complexité** 16
1. Introduction 16
2. Caractéristiques d'un graphe 17
3. Complexité 19
3.1 Complexité des algorithmes 19
3.2 Complexité des problèmes d'optimisation combinatoire 20
4. Complexité de quelques problèmes d'ordonnancement 21
4.1 Les problèmes à une-machine 21
4.2 Les problèmes à machines parallèles 23
5. Méthodes de résolutions exactes et approchées 24
5.1 Méthodes exactes 24
5.1.1 Programmation dynamique 24
5.1.2 Méthode par séparation et évaluation 24
5.1.3 Méthode par programmation linéaire 25
5.2 Méthodes Approchées 25
2.2.1 Les métaheuristiques 26
7. Conclusion 27
**Chapitre 3 : Le Job Shop Simple et la Contrainte de Blocage** 28
1. Introduction 28
2. Définition du Job shop simple 28
3. Formulation Mathématique 29

3.1 La programmation linéaire 30
3.2 Les graphes disjonctifs 30
4. Complexité du problème job shop 32
4.1 Problème du Flow shop 32
4.1.1 Algorithme de Johnson 32
4.1.2 Algorithme de Gilmor et Gomory 1964 33
4.2 Problème du Job Shop 35
4.2.1 Job shop à deux machines 35
4.2.2 Job shop à deux jobs 36
5. La Contrainte de Blocage 36
5.1. Définition 36
5.2. Quelques exemples d'application 37
5.2.1 Organisation de la circulation des trains dans un terminus 37
5.2.2 Organisation des blocs opératoires dans un hôpital 38
5.2.3 Déchargement de conteneurs dans un port 39
5.3 Etat de l'art 39
5.4. Modélisation de la contrainte de blocage 42
5.4.1 Modèle linéaire 42
5.4.2. Modélisation algébrique 43
5.4.3 Modélisation par les graphes alternatifs 43
5.4.4 La notion de circuit 47
6. Conclusion 47
**Chapitre 4 : Résolution Exacte** 49
1. Introduction 49
2. SEP pour le Job Shop Simple JSS 49
2.1 La séparation 50
2.2 *Heads* (*Têtes*) et *tails* (*Queues*) 54
2.3 Sélection immédiate 55
2.4 Les bornes inférieures 56
3. Méthode exacte pour le Job Shop avec Blocage 57
3.1. Principe de la méthode 58
3.2 Quelques définitions 59
3.2.1 L'algorithme de Bellman (plus long chemin) 60
3.2.2 L'algorithme de mise-à-niveau 60
3.2.3. L'algorithme de recherche des composantes fortement connexes 61
3.2.4. Graphe réduit 62
3.3 Description de l'algorithme séparation et évaluation 62
3.3.1 Séparation 63
3.3.2 Évaluation 64
3.3.3 Stérilisation 65
3.3.4 Bornes inférieures 65
4. Parallélisation de la méthode par Séparation et Evaluation 65
4.1 Principe de l'Algorithme SEP Parallèle 66
4.2 La topologie physique et la topologie logique 66

4.3 Les acteurs du système 70
4.3.1. Le contrôleur 70
4.3.2. Le calculateur 70
4.4. L'utilité du jeton dans l'algorithme 70
5. Conclusion 71
**Chapitre 5 : Résolution approchée** 72
1. Introduction 72
2. Le codage des solutions 73
2.1 Le codage basé sur les arcs alternatifs 73
2.2 Le codage basé sur les règles de priorité 75
2.3 Codage par rang 77
3. Les Algorithmes Génétiques 78
3.1 Le Croisement 78
3.2 La Mutation 79
3.3 La sélection 79
3.4 La population initiale 79
3.4.1 Algorithme heuristique 80
3.4.2. Algorithme par Séparation et Evaluation 81
3.5 Evaluation des individus 81
3.5.1 Algorithme de Ford 81
3.5.2. La Fonction *Eval* 82
4. Algorithme Génétique Parallèle 83
5. Optimisation par Essaims Particulaires PSO 86
5.1 Le codage des solutions 87
5.2 Opérateurs vitesse 87
5.2.1. 1ère Variante de PSO 88
5.2.2. 2ème Variante de PSO 89
6. Algorithme PSO parallèle proposé 90
7. Conclusion 92
**Chapitre 6 : Expérimentations numérique** 93
1. Résultats des méthodes par Séparation et Evaluation 93
1.1 Résultats issus des méthodes séquentielles 93
1.2 Résultats issus de la méthode parallèle 95
2. Résultats des métaheuristiques 96
2.1 Résultats issus des méthodes séquentielles 96
2.2 Résultats issus de la méthode parallèle 99
3. Analyse et discussion des résultats 100
4. Conclusion 103
**Conclusion Générale** 104
**Bibliographie** 106

## Introduction Générale

Un des problèmes importants à résoudre, afin d'améliorer le fonctionnement d'un système de production, est le problème d'ordonnancement des produits sur les machines. Celui-ci est connu dans la littérature comme étant un problème difficile à résoudre car la majorité de ses variantes sont NP-difficiles. Le problème du job shop est une classe de problèmes d'ordonnancement d'atelier à cheminement multiple. Pour le résoudre, nous devons trouver l'ordre de passage des produits sur les machines. Cet ordre doit minimiser un critère donné (la durée ou le délai de fabrication par exemple), tout en respectant un certain nombre de contraintes.

Les premières méthodes exactes par séparation et évaluation (SEP), permettant de résoudre le problème du job shop, datent de 1963. Ces méthodes donnaient des résultats, seulement, pour des petits exemples dont la taille est inférieure à 10 jobs $\times$ 10 machines. Il a fallu attendre l'année 1989 pour voir la première méthode exacte permettant de résoudre des exemples de taille égale à 10 jobs $\times$ 10 machines. Cette méthode a été développée par Carlier & Pinson [19]. Toutes les méthodes exactes développées plus tard jusqu'à ce jour utilisent le même principe de base de Carlier.

Avec l'évolution des systèmes de production, de nouveaux types de produits sont apparus donnant naissance à de nouvelles contraintes compliquant encore plus le problème. L'une de ces contraintes est la contrainte de blocage. Celle-ci apparaît dans les systèmes de fabrication ayant des ressources (machines) sans espace de stockage. En effet, un produit dont le traitement est terminé sur une machine donnée ne peut commencer son prochain traitement sur la machine suivante que si celle-ci est libre. Dans ce cas, puisque les deux machines n'admettent pas d'espace de stockage, alors le produit traité, se trouvant sur la première machine va y rester, malgré l'achèvement de son traitement, jusqu'à ce que la deuxième machine se libère. On dit alors que la première machine est bloquée, car elle ne peut être exploitée par un autre produit bien qu'elle ait terminé son travail.

Peu de travaux ont été réalisés dans la littérature pour la résolution de cette variante de job shop. Nous dressons plus loin dans cette thèse une situation détaillée de ces travaux. C'est la raison principale pour laquelle nous nous sommes intéressés à ce problème.

Dans cette thèse, nous nous intéressons au problème d'ordonnancement de type job shop avec blocage (JSB). Pour aborder la résolution de ce problème, nous avons divisé notre thèse en six chapitres. Le premier chapitre est consacré aux deux fonctions de production et

ordonnancement. Nous y avons rappelé les notions relatives au monde de la fabrication et de la production tout en présentant les différents types d'ateliers existants ainsi que les différents types de problèmes d'ordonnancement d'atelier. Un classement de ces problèmes est ensuite donné en se basant sur la description des contraintes et les différents critères d'optimisation possibles.

Le deuxième chapitre est dédié à la présentation de quelques notions de base de la théorie des graphes utilisées tout au long de cette thèse, ainsi qu'à la complexité des problèmes.

Le troisième chapitre présente le problème du job shop avec et sans contrainte de blocage. Nous commençons par présenter quelques exemples de l'application de cette contrainte dans des domaines autre que la production. Nous développons, ensuite, un état de l'art couvrant les plus importantes contributions concernant la résolution du problème d'ordonnancement avec la contrainte de blocage. Plusieurs modélisations de cette contrainte en utilisant la programmation linéaire, la modélisation algébrique et les graphes alternatifs, ont été décrites à la fin de ce chapitre.

Le quatrième chapitre contient la plupart de nos contributions dans cette thèse. En effet, dans un premier temps, nous proposons pour la résolution du problème de job shop avec la contrainte de blocage, une méthode exacte basée sur le principe de l'énumération implicite des solutions du problème. Cette méthode exploite différentes techniques de base de la théorie des graphes à savoir : les algorithmes de Bellman et Ford, l'algorithme de mise à niveau, l'algorithme de recherche des composantes fortement connexes et l'algorithme de réduction de graphes et ce, dans le but d'effectuer par la suite les processus de séparation et d'évaluation. Comme deuxième contribution, et étant donné la complexité du problème, nous proposons une parallélisation de la méthode par séparation et évaluation, proposée initialement, qui utilise deux nouvelles notions qui sont : le jeton et l'anneau logique plongé dans un réseau à topologie en étoile.

Vu le grand nombre de solutions qui croit avec la taille du problème, et toujours dans la perspective d'explorer un maximum de solutions en un temps raisonnable, nous avons consacré le cinquième chapitre, à l'étude du comportement de deux méta-heuristiques pour résoudre ce problème. Ces méthodes approchées sont : les algorithmes génétiques (AG) et l'Optimisation par Essaims de Particules (OEP). Dans ce chapitre, qui contient notre troisième contribution, nous proposons deux types de codages différents, ainsi que plusieurs techniques permettant d'améliorer la qualité des solutions retournées par ces méthodes. Une suite logique à ce travail est la parallélisation de ces deux méthodes. Cette approche de parallélisation est

basée sur le principe maître-esclave. Les résultats expérimentaux auxquels nous sommes parvenus sont satisfaisants, ouvrant ainsi la voie à de nouveaux horizons pour des travaux de recherches futurs.

Le sixième chapitre présente les résultats obtenus par les méthodes développées dans cette thèse appliquées sur des benchmarks connus dans la littérature. Une comparaison et une analyse des performances des différentes méthodes sont dressées dans ce chapitre.

# Chapitre 1
# Les fonctions Production & Ordonnancement

## 1. Introduction

La production est le résultat de la mise en œuvre de processus de fabrication reflétant la manière dont doivent être conçus les produits à fabriquer et les moyens de production. Ces derniers constituent une ressource limitée dans l'entreprise. Leur disponibilité et leurs spécificités font apparaître des concurrences entre les opérations nécessaires à la réalisation des multiples produits. Il convient donc de mettre en place des règles de conception et de gestion du système qui permettent de réaliser la fonction production par les meilleures manières (stratégies) qui soient.

Un système de production met en jeu de manière interactive des opérateurs, des moyens et de la matière, dans le but de faire évoluer cette dernière vers un état d'achèvement donné en suivant une organisation donnée des ressources (machines, poste de travail, poste de contrôle etc.) et du travail. Cette évolution se déroule durant une certaine période.

La valeur ajoutée représente la qualité du produit final fabriqué par un système de production par rapport à l'état initial de la matière première utilisée dans sa production. Afin d'améliorer la valeur des biens et des services et de réduire les coûts de production, l'entreprise doit recourir à une stratégie manufacturière qui assure plus de flexibilité, une meilleure sensibilité aux besoins de la clientèle et plus d'indépendance à l'égard des économies d'échelle.

L'ordonnancement est une activité incontournable de la fonction production. Son but est de permettre l'organisation dans le temps d'un ensemble de jobs qui doivent s'exécuter sur un ensemble de machines.

En général, un problème d'ordonnancement est soumis à un ensemble de contraintes telles que les contraintes portant sur : les temps de fabrication, la disponibilité des ressources requises, etc. La théorie de l'ordonnancement décrit un nombre pratiquement illimité de types de problèmes proposés et modélisés dans la littérature (voir par exemple, Blazewicz et al. [11], French [29], Lenstra [53], Pinedo [72], Rinnooy Kan [75], Tanaev et al. [84], Tanaev et al. [85]).

## 2. Fonction Production

Le système de production décrit l'ensemble du processus grâce auquel l'entreprise produit un bien ou un service apte à satisfaire une demande à l'aide des facteurs de production acquis sur le marché. Dans le cadre d'une entreprise, le système de production, outre sa finalité première qui est de produire un bien économique, cherche à satisfaire d'autres objectifs secondaires. Ces objectifs sont décrits ci-dessous.

### 2.1 Objectifs Associés

- *Objectif en terme de quantités produites :* la fonction production doit permettre à l'entreprise de satisfaire la demande qui lui est adressée. Ce qui suppose que l'entreprise adapte sa capacité de production au volume des ventes. Ceci passe par des actions visant à maintenir en l'état les capacités productives ou par la mise au point de plans d'investissement en terme de capacité.

- *Objectif en terme de qualité :* les biens économiques produits doivent être de bonne qualité, c'est-à-dire doivent permettre de satisfaire les besoins de la clientèle. Mais la production doit aussi être de qualité en termes d'utilisation de ressources afin de respecter le critère d'efficience attaché au système productif. Ce dernier doit donc être économe en ressources et constant en terme de qualité.

- *Objectif de coût :* le système productif adopté par l'entreprise doit proposer les plus faibles coûts de production possibles de manière à garantir la compétitivité de l'entreprise. De plus, les coûts de production calculés doivent aussi être mis en relation avec les coûts de production prévus par le centre opérationnel. Sur la longue période, cet objectif de coût se traduit par la recherche permanente de gains de productivité afin de détenir ou de conserver un avantage coût compétitif pour l'entreprise.

- *Objectif de délai :* le système de production doit certes produire, mais dans des délais raisonnables, c'est-à-dire en conformité avec le niveau de la demande à laquelle doit faire face l'entreprise. Ceci suppose la mise en place d'un mode de production réactif qui permettrait soit d'éviter des stocks de biens finaux, soit de ne pas connaître de goulots d'étranglement lors de la production. En terme de productivité, l'objectif de délai signifie aussi réduire les délais de fabrication.

- *Objectif de flexibilité :* le système productif doit être flexible soit pour pouvoir s'adapter aux variations de la demande, soit pour tenir compte des évolutions de l'environnement productif de l'entreprise (innovations technologiques par exemple),

soit pour permettre une production simultanée de plusieurs types de produits différents en même temps.

Bien que ces objectifs soient poursuivis conjointement par l'entreprise, il arrive que celle-ci soit amenée à donner un ordre de priorité dans la poursuite de ces objectifs en fonction du mode de production retenu. Ainsi, une organisation du travail basée sur le modèle de Taylor a comme objectif principal la recherche de gains de productivité. Mais ce type d'organisation se fait forcément au détriment de la recherche de flexibilité. Le choix d'un mode de production répond donc aux caractéristiques du marché dans lequel évolue l'entreprise. En effet, un marché de consommation de masse se caractérise par une standardisation des produits et donc par une compétitivité centrée sur les coûts de production ce qui entraîne la mise en place d'un mode de production ayant comme objectif prioritaire la recherche de gains de productivité.

**2.2 Niveaux Hiérarchiques de la Gestion de Production**

La gestion de production se compose de trois niveaux hiérarchiques connus dans la littérature: niveau stratégique, niveau tactique et enfin niveau opérationnel.

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

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

*c. Niveau opérationnel*, Il s'agit des décisions à court et à très court terme (au plus un an). C'est une gestion quotidienne pour faire face à la demande au jour le jour, dans le respect des décisions tactiques. Parmi les décisions opérationnelles, citons : la gestion des stocks, la gestion de la main d'œuvre et la gestion des équipements.

**3. Rôle de l'Ordonnancement en Gestion de Production**

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

pratique et classique, il est impossible de résoudre le problème d'ordonnancement globalement (au niveau supérieur), d'autant plus que les évènements aléatoires endogènes (pannes de machines, grèves, ...) et exogènes (commandes imprévues et prioritaires, ...) qui interviennent constamment nécessitent de reconstruire fréquemment la solution en cours. Il s'en suit que les problèmes d'ordonnancement sont traités aux niveaux inférieurs. La place de l'ordonnancement varie entre le niveau tactique et le niveau opérationnel.

## 4. Définitions de Base

Dans ce qui suit, nous présenterons quelques définitions de bases que nous utiliserons tout au long de cette thèse. Ces définitions permettront au lecteur de se familiariser avec le jargon du domaine de la production et de l'ordonnancement.

### 4.1 Gamme de Fabrication

Est une suite ordonnée et temporisée de traitements que subit un produit donné durant sa fabrication.

### 4.2 Routage

Un routage est une suite donnée de ressources que doit visiter le produit durant sa fabrication pour réaliser sa gamme de fabrication.

### 4.3 Job (Tâche)

Est une entité élémentaire de travail, notée *j*, qui consomme des moyens donnés avec une intensité donnée, localisée dans le temps par une date de début $t_j$ et une date de fin $c_j$ dont la réalisation nécessite une durée $p_j$.

Chaque job *j* peut être caractérisé par :

- Un vecteur temps de traitement $\overline{p_j} = \{p_{j1},..., p_{jm}\}$: $p_{jk}$ est le temps de traitement nécessaire à la machine $k$ pour terminer le traitement du job *j ; m* étant le nombre total de machines.
- Une date d'arrivée ou de disponibilité $r_j$ : c'est la date à laquelle le job *j* est prêt à être traité.
- Une date échue $d_j$ : si le traitement du job *j* doit être achevé avant la date $d_j$. Dans le cas contraire, le retard est pénalisé.

- Une date limite $\tilde{d}_j$ : Dans ce cas le retard n'est pas permis et le traitement du job *j* doit obligatoirement être achevé avant $\tilde{d}_j$.

- Un poids $w_j$ : qui exprime, en général, la priorité relative au job *j*.

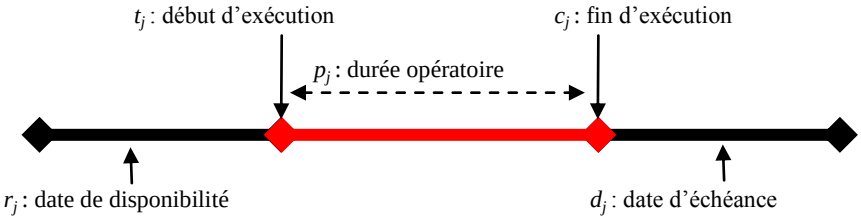


Figure 1. Caractéristique d'un *Job*

### 4.4 Ressource

Une ressource est un moyen (technique, humain, etc.) destiné à être utilisé pour la réalisation d'un travail et disponible en quantité limitée. Plusieurs types de ressources sont à distinguer :

- Une ressource est renouvelable si, après avoir été utilisée par un ou plusieurs jobs, elle est à nouveau disponible en même quantité (les hommes, les machines, etc.).

- Une ressource est consommable si, après avoir été utilisée par un ou plusieurs jobs, elle n'est plus disponible en même quantité (matière première, budget, etc.).

- Une ressource peut être *disjonctive*, i.e. : non partageable. Elle ne peut donc exécuter qu'un seul job à la fois.

- Une ressource peut être *cumulative*, i.e. : partageable. Elle peut donc être utilisée par plusieurs jobs simultanément (équipe d'ouvriers, poste de travail, etc.).

### 4.5 Opération

Nous appelons opération $o_{ji}$, le $i^{ème}$ traitement que subit un job *j* sur une machine, suivant sa gamme de fabrication.

### 4.6 Contraintes

Une contrainte est une restriction limitant le champ d'action d'une variable donnée du problème d'ordonnancement. Il existe plusieurs types de contraintes :

- Contraintes temporelles liées généralement au temps de production et de livraison ;

- Contraintes d'antériorités qui décrivent un ordre d'exécution impératif entre les jobs ou entre les opérations composant un job donné ;

- Contraintes d'utilisation de ressources qui expriment la nature et la quantité des moyens utilisés par un job ;

- Contraintes de disponibilités des ressources qui précisent la nature et la quantité des moyens disponibles au cours du temps.

- Contraintes de préemption qui précisent si l'exécution d'un job ou d'une opération peut être interrompue et repris par la suite. Dans ce cas, nous parlons d'un ordonnancement préemptif. Dans le cas contraire, c'est-à-dire les jobs ou les opérations sont exécutés sans interruption alors nous parlons, d'un ordonnancement non préemptif.

- Etc.

### 4.7 Ordonnancement

Un ordonnancement est un ordre de passage, à définir dans le temps, des différentes opérations sur différentes machines en vue de leur exécution, en respectant les contraintes du problème. En effet, c'est l'établissement du planning de production qui est appelé *Ordonnancement*. Ce dernier permettra d'avoir le temps de début $t_{jk}$ de chaque job *j* sur chaque machine *k*. L'un des buts de ce planning est de réduire le temps total de production appelé *makespan*.

Un problème d'ordonnancement peut être définit avec une seule ou plusieurs machines. Les machines peuvent être de différentes natures : elles peuvent être interchangeables, on parle dans ce cas la de machines parallèles ***identiques*** ; elles peuvent être ***uniformes*** entre elles, un seul paramètre de vitesse les faisant différer ; elles peuvent être ***indépendantes*** les unes des autres. Dans le cas où nous avons des machines dédiées alors nous parlons de problèmes d'ordonnancement d'ateliers.

## 5. Problèmes d'Ordonnancement d'Ateliers

Les problèmes d'ordonnancement d'ateliers sont généralement définit suivant le type d'orientation des machines sur le sol de l'atelier. Cette orientation dépend en général des produits réalisés par celui-ci. Nous distinguons en général, trois classes d'organisation d'ateliers : Atelier à cheminement unique (Flow Shop), Atelier à cheminement multiple (Job Shop) et Atelier à cheminement libre (Open Shop).

Dans un atelier à cheminement unique, l'organisation de la production est linéaire. Elle est caractérisée par une séquence d'opérations identiques pour tous les produits. Un job est ainsi constitué d'opérations passant par différentes machines de manière linéaire ; l'ordre de passage par les machines est le même pour tous les jobs. Autrement dit, on dispose de *m* machines en série et de *n* jobs à exécuter. Ces derniers doivent passer par chacune des machines dans le même ordre.

Dans un atelier à cheminement multiple chaque job emprunte un "chemin" qui lui est propre et qui respecte sa gamme de fabrication. Ce type d'organisation correspond généralement à une production par lot, notamment dans une unité de production disposant de moyens polyvalents utilisés suivant des séquences différentes afin de réaliser des produits divers. Si, pour chaque machine, nous disposons d'un seul exemplaire, on dit que l'organisation est un Job Shop simple.

Dans un atelier à cheminement libre, les contraintes de précédence sont relâchées. Donc, les opérations nécessaires à la réalisation de chaque job peuvent être effectuées dans n'importe quel ordre (les gammes sont libres). Ce cas se présente lorsque chaque produit à fabriquer doit subir une séquence d'opérations, mais dans un ordre totalement libre.

Tous les problèmes d'ordonnancement d'atelier partagent un noyau d'informations de base qui les caractérisent. Ces informations sont :

- Les données d'un problème d'ordonnancement sont liées aux machines et aux jobs (nombre de jobs et de machines, gammes de fabrication de chaque job, etc.).
- Les indicateurs de performance varient essentiellement en fonction des temps d'attente que subissent les traitements entre deux opérations. Ce qui se répercute sur les dates de fin globales des jobs. Très souvent, les critères d'optimisation utilisés minimisent le maximum ou la moyenne de ces indicateurs sur l'ensemble des jobs. Le critère du makespan qui représente la durée totale de l'ordonnancement est généralement le plus utilisé.

**6. Identification des problèmes d'ordonnancement**

Un ordonnancement peut être représenté par le diagramme de Gantt comme le montre la Figure 2. Ce diagramme représente les machines en fonction des jobs (Figure 2.*a*) ou bien les jobs en fonction des machines (Figure 2.*b*).

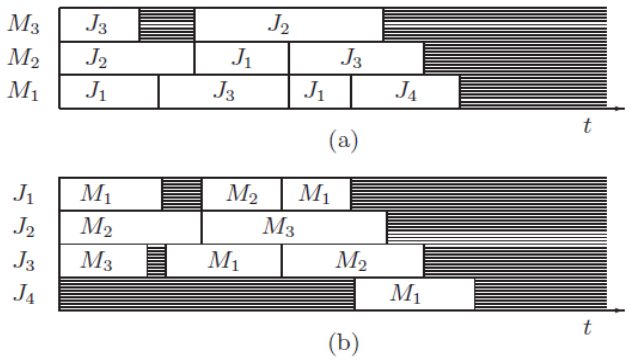


Figure 2. Diagrammes de Gantt

En général dans un problème d'ordonnancement, chaque job *j* est composé d'un ensemble d'opérations $O_j = \{O_{j1}, O_{j2}, ....O_{jv_j}\}$ où $v_j$ représente le nombre maximum d'opérations composant le job *j*. De ce fait, $O_j$ décrit l'ordre de passage du job *j* sur les différentes machines, ce qui engendre des temps de traitement sur chaque machine *k* notés $p_{jk}$.

Pour l'identification des problèmes d'ordonnancement, nous adoptons la notation constituée de trois champs $\alpha / \beta / \gamma$, assez utilisée dans la littérature [15].

Le premier champ est constitué de deux éléments : $\alpha = \alpha_1 \alpha_2$ et décrit l'environnement des machines utilisées :

- Le paramètre $\alpha_1$ décrit le type des machines utilisées $\alpha_1 = \{1, P, Q, R, F, J, O\}$ tel que :

  $\alpha_1 = 1$, une seule machine.

  $\alpha_1 = P$, machines parallèles identiques.

  $\alpha_1 = Q$, machines parallèles uniformes.

  $\alpha_1 = R$, machines parallèles générales.

  $\alpha_1 = F$, atelier de Type Flow shop.

  $\alpha_1 = J$, atelier de Type Job shop.

  $\alpha_1 = O$, atelier de Type Open shop.

- Le paramètre $\alpha_2$ $(pour\ \alpha_1 \neq 1)$ indique le nombre de machines utilisées :

  1. Si $\alpha_2$ est égal à un entier positif quelconque {2,3, ...}, dans ce cas $\alpha_2$ désigne le nombre exacte de machines.
  2. Si $\alpha_2 = k$, alors *k* est un nombre quelconque fixé de machines.
  3. Si $\alpha 2 = \varnothing$ le nombre de machines est arbitraire.

Le second champ $\beta$ décrit l'ensemble des caractéristiques des jobs, $\beta = \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$ tels que :

- $\beta_1 = pmtn$, indique que la préemption est autorisée, i.e. : qu'un job ou une opération peut être interrompue, ensuite reprise à une date ultérieure, même sur une autre machine à plusieurs reprises. Si la préemption n'est pas autorisée $\beta_1$ n'apparaît pas dans $\beta$.

- $\beta_2 = prec$, indique l'existence de relations de précédence entre les jobs. Ces relations de précédence peuvent être représentées par un graphe acyclique, orienté $G = (V, U)$, où : $(J_i, J_k) \in U$, ssi $J_i$ doit être terminée avant $J_k$.

  Si $\beta_2$ = *Intree*, respectivement *outtree*, alors *G* est une arborescence avec une racine et tous les arcs sont orientés vers la racine, respectivement dans le sens opposé de la racine.

  Si $\beta_2$ = *arbre*, alors *G* est soit un *Intree* ou bien un *outtree*.

  Si $\beta_2$=*chaîne* alors *G* est une arborescence dans laquelle les degrés entrant et sortant de chaque sommet sont au plus égaux à 1, c'est-à-dire que *G* forme un ensemble de chemins indépendants.

- $\beta_3 = r_j$, indique que chaque job *j* admet une date de disponibilité $r_j$. Dans le cas où $r_j = 0$ pour tous les jobs, alors $\beta_3$ n'apparaît pas dans $\beta$.

- $\beta_4$, indique l'existence de certaines restrictions sur les temps de traitements des opérations sur les machines. Dans le cas où $\beta_4$ prend la valeur "$p_i = 1$" ou bien "$p_i = p$" alors cela veut dire que toutes les opérations ont des temps de traitement égaux à 1 ou *p*, respectivement.

- $\beta_5 = d_j$, indique que chaque job *Jj* admet une date échue $d_j$ (si le critère d'optimalité n'est pas en fonction des dates échues).

Le dernier champ $\gamma$ désigne le (ou les) critère(s) d'optimisation. Pour chaque job $j, j = 1,...,n$, traité dans un ordonnancement admissible donné, les paramètres suivants peuvent être calculés:

- Une date d'achèvement $c_i$ qui représente la date de fin de traitement du job $j_i$.

- Une durée de séjour $f_i$ qui est la somme du temps d'attente et du temps de traitement. Elle peut être calculée de la manière suivante : $f_i = c_i - r_i$.
- Une tardivité $l_i$ qui est la différence entre la date d'achèvement et la date échue. $l_i = c_i - d_i$.
- Un retard $t_i$ avec $t_i = \max\{l_i, 0\}$.
- Un indicateur de retard $u_i$ où : $u_i = \begin{cases} 1 \; si \; c_i > d_i \\ 0 \; si \; non \end{cases}$

Le critère à optimiser (à minimiser ou à maximiser) est exprimé sous forme de fonction appelée fonction objectif, qui est le but que l'on cherche à atteindre en résolvant un problème d'ordonnancement.

L'objectif principal d'un algorithme d'ordonnancement est de modifier l'ordre des jobs traités afin d'améliorer la satisfaction des utilisateurs qui est représentée par la fonction objectif. L'algorithme d'ordonnancement dépend donc de l'objectif souhaité.

Parmi les critères fréquemment rencontrés dans la littérature, on trouve les critères qui minimisent :

- Le temps total d'exécution de l'ordonnancement : $C_{\max} = \max_{1 \le i \le n}\{c_i\}$ (makespan).
- Le temps de séjour moyen : $\bar{F} = \frac{1}{n}\sum_{i=1}^{n} f_i$
- La grande tardiveté : $L_{\max} = \max_{1 \le i \le n}\{l_i\}$
- Le nombre de jobs en retard : $N_T = \sum_{i=1}^{n} U_i$

Dans certaines applications, d'autres critères sont aussi utilisés :

- Le temps de fin de traitement moyen : $\bar{C} = \frac{1}{n}\sum_{i=1}^{n} c_i$
- Le temps de flot moyen pondéré : $\bar{F}_w = \frac{\sum_{i=1}^{n} w_i f_i}{\sum_{i=1}^{n} w_i}$
- Le temps de fin de traitement moyen pondéré : $\overline{C_w} = \frac{\sum_{i=1}^{n} w_i c_i}{\sum_{i=1}^{n} w_i}$
- Le grand temps de séjour : $F_{\max} = \max_{1 \le i \le n}\{f_i\}$
- Le grand retard : $T_{\max} = \max_{1 \le i \le n}\{t_i\}$
- La tardiveté moyenne : $\bar{L} = \frac{1}{n}\sum_{i=1}^{n} l_i$
- La tardiveté moyenne pondérée : $\overline{L_w} = \frac{\sum_{i=1}^{n} w_i l_i}{\sum_{i=1}^{n} w_i}$

- Le retard moyen : $\bar{T} = \frac{1}{n}\sum_{i=1}^{n} t_i$
- Le retard moyen pondéré : $\overline{T_w} = \frac{\sum_{i=1}^{n} w_i t_i}{\sum_{i=1}^{n} w_i}$

Pour mieux illustrer la notation des problèmes d'ordonnancement à trois champs α | β | γ, nous présentons dans ce qui suit quelques exemples tirés des travaux de Brucker [15]. Dans chaque cas, nous allons décrire le problème, ensuite nous donnerons l'ordonnancement correspondant en utilisant le diagramme de Gantt.

**Exemple1**: $P/prec, p_j = 1/C\max$, est un problème d'ordonnancement de $n$ jobs, avec temps de traitement unitaire et contraintes de précédences, sur des machines identiques, ayant pour critère d'optimisation la minimisation du makespan, noté *Cmax*. Une instance de ce problème est représentée par un graphe orienté avec $n = 7$ sommets et $m = 2$ machines. La Figure 3 montre un exemple de ce problème avec l'ordonnancement correspondant.

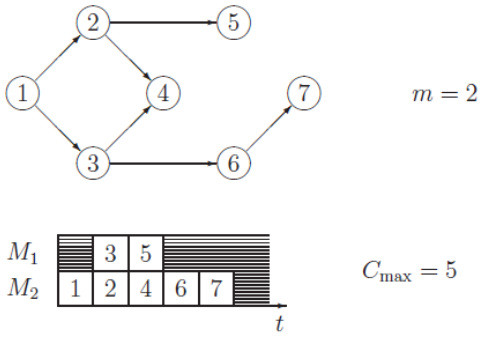


Figure 3. Exemple d'une instance de $P/prec, p_j = 1/C\max$.

**Exemple2 :** $1/r_j, pmtn/L\max$, est un problème d'ordonnancement de $n$ jobs préemptifs sur une machine avec des temps de disponibilité et des dates échues non tous nuls, avec l'objectif de minimiser le maximum des retards. Une instance est présentée dans la Figure 4.

| $j$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $p_j$ | 2 | 1 | 2 | 2 |
| $r_j$ | 1 | 2 | 2 | 7 |
| $d_j$ | 2 | 3 | 4 | 8 |

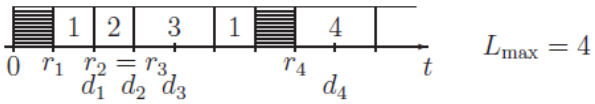


Figure 4. Exemple d'une instance de $1/r_j, pmtn/L\max$

**Exemple3 :** $J3/p_j = 1/C\max$ **,** est un problème d'ordonnancement de type job shop à 3 machines avec temps de traitement unitaire en minimisant le makespan. Une instance à 5 jobs est présentée dans la Figure 5.

| *j/i* | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | $M_1$ | $M_3$ | $M_2$ | $M_1$ |
| 2 | $M_2$ | $M_3$ | - | - |
| 3 | $M_3$ | $M_1$ | - | - |
| 4 | $M_1$ | $M_3$ | $M_1$ | - |
| 5 | $M_3$ | $M_1$ | $M_2$ | $M_3$ |

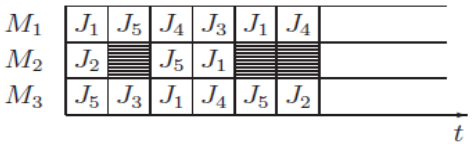


Figure 5. Exemple d'une instance de $J3/\ p_{jk} = 1/C\max$

## 7. Conclusion

Nous avons consacré ce premier chapitre aux fonctions production et ordonnancement. Nous avons d'abord introduit quelques définitions de base qui nous serons utiles dans la suite de ce travail, tout en décrivant les différents systèmes de production existant dans la littérature. Ensuite, nous avons introduit les problèmes d'ordonnancement en général afin de familiariser le lecteur avec ce type de problèmes. Dans le chapitre suivant, nous introduisons quelques notions de la théorie des graphes, de complexité des problèmes et de complexité des algorithmes.

# Chapitre 2
# Théorie des Graphes & Complexité

## 1. Introduction

La théorie des graphes [35] est largement utilisée dans la modélisation des problèmes d'ordonnancement. C'est une filière des mathématiques qui modélise des problèmes réels sous forme de graphes en utilisant les notions de sommets et arcs. Ce chapitre est consacré à la présentation des définitions de base de la théorie des graphes, de la complexité et à la présentation de quelques méthodes de résolution exacte et approchées, qui sont utilisées dans cette thèse. Le but étant de familiariser le lecteur avec ces notions afin de faciliter la compréhension de ce travail.

Un graphe est une représentation géométrique d'un ensemble de sujets, symbolisés par des points appelés sommets et liés entre eux soit par des traits non orientés appelés arêtes ou bien par des traits orientés appelés arcs. Ces traits expriment une relation donnée.

Mathématiquement, un graphe est toujours définit par un ensemble de sommets noté $V=\{v_1,v_2,\ldots, v_n\}$ et une relation $R$ définit sur les éléments de $V$. Pour chaque couple de sommets $(v_1,v_2)\in V^2$, si le sommet $v_1$ vérifie la relation $R$ avec le sommet $v_2$, alors les deux sommets sont reliés par un arc ou une arête. Si la relation $R$ est vérifiée dans les deux sens alors les deux sommets seront reliés par une arête. Sinon, c'est un arc qui joint les deux points dont l'orientation suit le sens de la relation $R$ qui est vérifiée. Ainsi, en regroupant tous les couples de sommets qui vérifient la relation $R$, nous construisons l'ensemble de toutes les arêtes du graphe noté $E$ ou bien un ensemble de tous les arcs noté $U$.

De là, nous distinguons deux classes principales de graphes, à savoir : les graphes orientés dont les sommets sont reliés par des arcs et les graphes non orientés dont les sommets sont reliés par des arêtes. Un graphe non orienté est toujours noté $G=(V,E)$, et un graphe orienté est noté $G=(V,U)$.

Dans ce chapitre, nous nous focaliserons sur les graphes orientés car la modélisation que nous avons adoptée se base sur ce type de graphes.

**Exemple :**

Soient $V=\{v_1,v_2,v_3, v_4, v_5\}$ un ensemble de cinq sommets et $U=\{u_1,u_2,u_3,u_4\}$ un ensemble de quatre arcs. Chaque arc est définie comme suit : $u_1=(v_1,v_2)$, $u_2=(v_3,v_5)$, $u_3=(v_5,v_2)$, $u_4=(v_1,v_4)$. Le graphe orienté $G=(V,U)$ engendré est donné à la Figure 6.

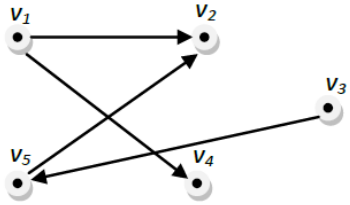


Figure 6. Exemple de graphe Orienté

**2. Caractéristiques d'un graphe**

Soit, $G=(V,U)$ un graphe orienté avec $n$ sommets et $m$ arcs, où $V=\{v_1,v_2,\ldots, v_n\}$ et $U=\{u_1,u_2,\ldots,u_m\}$.

1- Soit $u=(v_1,v_2)$ un arc de $V$, le sommet $v_1$ est appelé extrémité initiale de l'arc $u$ et $v_2$ son extrémité terminale.
2- Un arc dont les deux extrémités coïncident est appelé une *boucle*, $z=(v_k,v_k)$.
3- Le graphe $G$ est dit *simple*, s'il n'admet pas de boucles et $\forall(v_k, v_l) \in V^2$ $v_k$ et $v_l$ ne sont reliés, qu'au plus, par un seul arc dans le même sens.
4- On appel graphe partiel de $G=(V,U)$, un graphe ayant même ensemble de sommets $V$ que $G$ et ayant pour ensemble d'arcs une partie de $U$.
5- Soit $G_1=(V_1,U_1)$ un graphe. On dit que $G_1$ est un sous graphe de $G$ engendré par $V_1$. Si et seulement si, $V_1 \subset V$ , $U_1 \subset U$ et tous les arcs de $U_1$ ont leurs deux extrémités dans $V_1$.
6- Deux sommets sont dit adjacents si et seulement si, ils sont reliés par un arc.
7- Deux arcs sont dit adjacents si et seulement si, ils ont au moins une extrémité commune.
8- Un graphe G=(V,E) non orienté est dit complet si et seulement si tous les sommets de V sont adjacentes deux à deux.
9- Une clique, est un sous graphe complet.
10- Un arc u est incident intérieurement à un sommet v si et seulement si v est l'extrémité terminale de u.
11- Un arc u est incident extérieurement à un sommet v si et seulement si v est l'extrémité initiale de u.

12- Un demi-degré intérieur (respectivement extérieur) d'un sommet v, noté $d_g^-(v)$ (respectivement $d_g^+(v)$), est le nombre d'arcs incidents intérieurement (respectivement extérieurement) à v.

13- Dans les graphes orientés, le degré d'un sommet v est noté $d_g(v) = d_g^-(v) + d_g^+(v)$.

14- Une source est un sommet v, dont $d_g^-(v) = 0$.

15- Un puits est un sommet v, dont $d_g^+(v) = 0$.

16- Une chaîne est une suite ordonnée d'arcs, tels que chaque deux arcs qui se suivent ont une extrémité commune. Si tous les arcs d'une chaîne ont le même sens d'orientation cette chaîne est appelée un chemin.

17- Une chaîne (respectivement un chemin) est dit simple, si et seulement si, cette chaîne (respectivement ce chemin) ne repasse pas deux fois par la même arête (respectivement le même arc).

18- Un cycle est une chaîne simple dont les deux extrémités coïncident.

19- Un circuit est un chemin simple dont les deux extrémités coïncident.

20- Un sommet v est dit isolé, si et seulement si, $d_g(v) = 0$.

21- Un graphe G est connexe, si et seulement si, pour toute paire de sommets quelconque de V il existe une chaîne les joignant.

22- Un graphe orienté G est dit fortement connexe, si et seulement si, pour toute paire de sommets (vk,vl) quelconque, s'il existe un chemin reliant vk à vl alors il existe un autre chemin reliant vl à vk.

23- Un graphe connexe, sans cycle est appelé arbre (Tree en anglais).

24- Un sommet v est dit pendant, si et seulement si, $d_g(v) = 1$.

25- Un graphe pondéré est un graphe ayant des pondérations affectées aux arcs.

26- Un réseau est un graphe orienté, pondéré ayant au plus un sommet source et un sommet puits.

27- Un graphe peut être représenté sur machine (ordinateur) en utilisant différentes structures de données. A titre d'exemple, nous pouvons citer la matrice d'adjacence, matrice d'incidence, dictionnaire des successeurs etc.

28- Le poids d'un chemin, le poids d'un cycle ou le poids d'un circuit est la somme des pondérations des arcs qui le composent.

29- Le plus court chemin entre deux sommets v1 et v2, respectivement le plus long chemin entre deux sommets v1 et v2, est un chemin de poids minimum, respectivement un chemin de poids maximum.

30- Un circuit absorbant est un circuit dont la somme des pondérations des arcs qui le compose est inférieure à zéro pour une recherche d'un plus court chemin, ou bien supérieure à zéro pour une recherche d'un plus long chemin.

31- Un graphe *G* admet un plus long ou un plus court chemin si et seulement si *G* n'admet pas de circuit absorbant.

## 3. Complexité

Un problème $P_r$ est dit problème d'optimisation combinatoire s'il admet :

1- Un ensemble de solutions *D*, discret, fini et dénombrable.

2- Une fonction objectif $f$ permettant de définir un ordre total sur *D*.

Tout problème d'optimisation combinatoire comporte dans sa description des paramètres formels. En donnant des valeurs à ces paramètres (en les rendant effectifs) on obtient une instance *I* de ce problème. En d'autres termes, une instance est une spécification des données définissant un cas à traiter d'un problème donné.

Résoudre un problème d'optimisation combinatoire, c'est trouver $\hat{d} \in D$ tel que $f(\hat{d}) = \min_{d \in D}\{f(d)\}$ où D est l'ensemble de toutes les solutions possibles associées à une instance *I* du problème. Dans la suite de cette section, nous supposons que nous disposons d'un problème d'optimisation combinatoire $P_r$.

### 3.1 Complexité des algorithmes

Un algorithme est un ensemble d'instructions élémentaires séquentielles tel que, pour toute instance *I* d'un problème $P_r$ à traiter, il permet de passer des données initiales aux résultats recherchés. Définir un algorithme, c'est définir la suite des instructions de base qui le compose.

L'efficacité d'un algorithme est mesurée par sa complexité. Celle-ci nous permet de définir en même temps ses performances. La complexité d'un algorithme *A* permet d'évaluer, au pire des cas, le nombre d'opérations élémentaires effectuées par cet algorithme pour résoudre une instance *I* d'un problème donné.

Sur ordinateur, toutes les données d'un algorithme *A* sont codées en bits. Ces données codées définissent la taille de l'instance *I* à résoudre. Posons *f*(*A*,*I*) le nombre d'opérations élémentaires effectuées pour passer d'une instance *I* aux résultats voulus à l'aide de l'algorithme *A*. La complexité $c_A$ de *A* est alors définie comme la fonction qui, sur les

instances de *I* de taille fixée, considère le maximum de $f(A,I)$ où $c_A(n)=max\{ f(A,I)$ pour toute instance *I* telle que $|I|=n\}$, avec $|I|$ représente la taille de *I* [23].

Il est souvent très difficile de calculer avec précision la valeur de $c_A$. C'est pour cette raison qu'on se limite au calcul d'un majorant asymptotique de $c_A(n)$ qu'on note $O(g(n))$. En reprenant les notations utilisées par [23], nous avons la formulation suivante : $\exists\, k\,, \exists\, N_k, telle\ que\ \forall\, n \geq N_k : |c_A(n)| \leq k.|g(n)|$

L'appellation *algorithme polynomial en* $O(n^k)$ est donnée à tout algorithme dont la complexité est majorée par un polynôme $n^k$ de la taille *n* des données. Les autres algorithmes sont dit *exponentiels*.

Les algorithmes polynomiaux donnent en général des solutions aux instances de grande taille des problèmes posés en un temps raisonnable. Ce qui n'est pas le cas pour les algorithmes non polynomiaux ou exponentiels.

Par contre, un algorithme pseudo-polynomial est un algorithme polynomial tel que les coefficients du polynôme majorant sont différents par rapport au modulo de la taille du problème. D'après Garey et Johnson (1979) [31] « Un algorithme pseudo-polynomial se comporte d'une manière polynomiale sauf pour certaines instances ayant des nombres exponentiellement grands. L'apparition de telles instances n'étant qu'anecdotique».

### 3.2 Complexité des problèmes d'optimisation combinatoire

Il est très facile de remarquer que certains problèmes $P_r$ sont plus faciles que d'autres à résoudre sur un ordinateur. Une théorie de la complexité a été développée et permet mathématiquement de classer les problèmes faciles et difficiles en deux classes : la classe P et la classe NP-complet.

Nous appelons problème de décision, tout problème qui se résout par une réponse «*Vrai*» ou «*Faux*». Pour chaque problème $P_r$ nous pouvons, toujours, lui associer un problème de décision.

Un problème $P_r$ est Polynomial s'il peut être résolu en un temps polynomial (un temps raisonnable) par une machine de Turing. C'est-à-dire, par un « ordinateur ». Typiquement, la recherche d'un élément dans une liste et le tri d'une liste de valeurs sont des problèmes polynomiaux.

Un problème $P_r$ est Non deterministic Polynomial NP, s'il peut être résolu en un temps polynomial par une machine de Turing non déterministe.

Les définitions des deux classes P et NP ne sont pas exclusives. En particulier, les problèmes dans P sont aussi dans NP, car la résolution d'un problème en temps polynomial sur une machine de Turing déterministe, restera aussi en temps polynomial sur une machine de Turing non déterministe ; on a donc P⊂NP.

Un problème $P_r$ est NP-complet si :

- Il est dans NP et ;
- Tout problème de NP peut être polynomialement réduit à ce problème.

Si un jour nous trouvions un algorithme polynomial, sur une machine de Turing déterministe, qui permet de résoudre un problème NP-complet, alors tous les problèmes de NP pourront être résolus, en temps polynomial.

Un problème d'optimisation combinatoire est dit NP-difficile, si le problème de décision qui lui est associé appartient à la classe NP-complet.

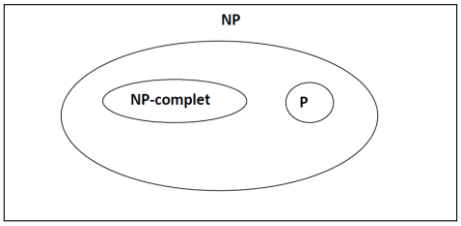


Figure 7. Classement des problèmes d'optimisation combinatoire.

Les différentes classes de complexité des problèmes nécessitent des méthodes de résolution différentes. Les problèmes appartenant à la classe P ont des algorithmes polynomiaux permettant de les résoudre. Pour les problèmes appartenant à la classe NP-complet ou NP-difficile, nous ne connaissons pas, jusqu'à présent d'algorithmes polynomiaux pour les résoudre. En plus, nous ne savons même pas s'ils existent ou non. C'est pour cette raison que les chercheurs s'orientent vers des méthodes exactes non polynomiales (pour les instances de petite taille) ou approchées polynomiales (pour les instances de grande taille).

## 4. Complexité de quelques problèmes d'ordonnancement

Généralement, les problèmes d'ordonnancement sont connus dans la littérature pour être NP-difficile. Nous présentons dans ce qui suit quelques cas particuliers de ces problèmes, cités par Brucker [15], qui sont soit polynomiaux ou bien pseudo-polynomiaux.

### 4.1 Les problèmes à une-machine

Les problèmes à une machine sont des problèmes d'ordonnancement connus dans la littérature. Les premiers articles qui les ont traités datent de 1955 et 1956. Généralement, ces

problèmes sont classés NP-difficiles. Cependant, nous présentons dans ce qui suit quelques cas particuliers qui sont soit polynomiaux, soit pseudo-polynomiaux.

Les problèmes suivants sont des problèmes polynomiaux :

- $1/prec;r_i/C\max$ *démontré par Lawler* $O(n^2)$
- $1/prec;r_i;p_i=p/L\max$ *démontré par Simons* $O(n^3\log\log n)$
- $1/prec/F\max$ *démontré par Lawler* $O(n^2)$
- $1/prec;r_i;p_i=1/F\max$ *démontré par Bruc ker* $O(n^2)$
- $1/prec;r_i;pmtn/F\max$ *démontré par ba ker et al.* $O(n^2)$
- $1/r_i;pmtn/\sum C_i$ *démontré par ba ker* $O(n\log n)$
- $1/prec;r_i;p_i=p/\sum C_i$ *démontré par Bruc ker* $O(n^2)$
- $1/prec;r_i;p_i=p,pmtn/\sum C_i$ *démontré par Baptiste et al.* $O(n^2)$
- $1/r_i;p_i=p/\sum w_iC_i$ *démontré par Baptiste* $O(n^7)$
- $1/sp-graph/\sum w_iC_i$ *démontré par Lawler* $O(n\log n)$
- $1//\sum U_i$ *démontré par Moore* $O(n\log n)$
- $1/r_i;pmtn/\sum U_i$ *démontré par Lawler* $O(n^5)$ *et par Baptiste* $O(n^4)$
- $1/r_i;p_i=p/\sum w_iU_i$ *démontré par Baptiste* $O(n^7)$
- $1/r_i;p_i=p;pmtn/\sum w_iU_i$ *démontré par Baptiste* $O(n^{10})$
- $1/r_i;p_i=p/T\max$ *démontré par Baptiste* $O(n^7)$
- $1/r_i;p_i=1/F\max$ *démontré par Bruc ker* $O(n^3)$
- $1/r_i;p;pmtn/T\max$ *démontré par Tian et.al.* $O(n^2)$

Les problèmes suivants sont des problèmes pseudo-polynomiaux :

- $1//\sum w_iU_i$ *démontré par Lawler & Moore*
- $1/r_i;pmtn/\sum w_iU_i$ *démontré par Lawler*
- $1//T$ *démontré par Lawler*

Dans tous les problèmes précédents, nous supposons que tous les jobs sont disponibles au temps $t$=0 et ont des fonctions objectif monotones.

### 4.2 Les problèmes à machines parallèles

Idem pour ce cas, les problèmes d'ordonnancement à machines parallèles sont classés en général, NP-difficile, qu'ils soient avec préemption ou sans préemption. Cependant, quelques cas particuliers sont résolus avec un algorithme polynomial ou pseudo-polynomial [15].

Les problèmes suivants sont à machines parallèles avec préemption. Ils peuvent être résolus d'une manière polynomiale :

- $P/\ pmtn/C\max$ *démontré par McNaughton* $O(n)$
- $P/outtree;\ pmtn;r_i/C\max$ *démontré par Lawler* $O(n^2)$
- $P/tree;\ pmtn/C\max$ *démontré par Gonzalez & Johnson* $O(n\log m)$
- $Q/\ pmtn;r_i/C\max$ *démontré par Labetoulle et al.* $O(n\log n+mn)$
- $Q/chains;\ pmtn/C\max$ *démontré par Gonzalez & Sahni* $O(n+m\log n)$
- $P/in\,tree;\ pmtn/L\max$ *démontré par Lawler* $O(n^2)$
- $Q2/\ prec;\ pmtn;r_i/L\max$ *démontré par Lawler* $O(n^6)$
- $P/\ pmtn/L\max$ *démontré par Baptiste* $O(n\log n)$
- $Q/\ pmtn/L\max$ *démontré par Labetoulle et al.* $O(n\log n+mn)$
- $Q/\ pmtn;r_i;d_i/-$ *démontré par Federguen & Gronevelt* $O(mn^3)$
- $R/\ pmtn;r_i/L\max$ *démontré par Lawler & Labetoulle*
- $P/\ p_i=p;outtree;\ pmtn/\sum C_i$ *démontré par Brucker et al.* $O(n^2)$
- $P/\ p_i=1;outtree;\ pmtn;r_i/\sum C_i$ *démontré par Brucker et al.* $O(n^2)$
- $P2/\ p_i=p;\ prec;\ pmtn/\sum C_i$ *démontré par Coffman et al.*
- $P2/\ p_i=p;outtree;\ pmtn;r_i/\sum C_i$ *démontré par Lushchakova*
- $P2/\ p_i=p;\ pmtn;r_i/\sum C_i$ *démontré par Herrbach & Leung* $O(n\log n)$
- $P/\ p_i=p;\ pmtn;r_i/\sum C_i$ *démontré par Brucker & Kravchenko*
- $Q/\ pmtn/\sum C_i$ *démontré par Labetoulle et al.* $O(n\log n+mn)$
- $P/\ p_i=p;\ pmtn/\sum w_iC_i$ *démontré par McNaughton* $O(n\log n)$
- $Q/\ p_i=p;\ pmtn/\sum U_i$ *démontré par Baptiste et al.*
- $Qm/\ pmtn/\sum U_i$ *démontré par Lawler* [81], *Lawler & Martel* $O(n^{3(m-1)})$

- $Pm / p_i = p; pmtn / \sum w_i U_i$ *démontré par Baptiste* $O(n^{3m+4})$
- $P / p_i = 1; pmtn; r_i / \sum w_i U_i$ *démontré par Bruc*ker *et al.* $O(mn^3)$
- $P / p_i = p; pmtn / \sum T_i$ *démontré Par Baptiste et al.* $O(n^3)$
- $P / p_i = 1; pmtn; r_i / \sum w_i T_i$ *démontré par Baptiste et al.* $O(mn^3)$

Les problèmes suivants sont à machines parallèles avec préemption. Ils peuvent être résolus d'une manière pseudo-polynomiale :

- $Pm / pmtn / \sum w_i C_i$ *démontré par McNaughton , Lawler et al.*
- $Qm / pmtn / \sum w_i U_i$ *démontré par Lawler , Lawler et Martel*

D'autres problèmes à machines parallèles classés polynomiaux, pseudo-polynomiaux ou bien NP-Difficile dans le cas non-préemptif, peuvent être consultés dans la référence [15].

## 5. Méthodes de résolutions exactes et approchées

Les méthodes de résolution proposées pour le problème d'ordonnancement sont très nombreuses. Nous décrivons dans un premier temps les méthodes de résolution exactes, puis les méthodes heuristiques.

### 5.1 Méthodes exactes

#### 5.1.1 Programmation dynamique

La programmation dynamique décompose un problème de dimension *n* en *n* problèmes de dimension 1. Le système est alors constitué de n étapes que l'on résout séquentiellement, le passage d'une à une autre étape se faisant à partir des lois d'évolution du système et d'une décision. Le principe d'optimalité est basé sur l'existence d'une équation récursive permettant de décrire la valeur optimale du critère à une étape en fonction de sa valeur à l'étape précédente.

#### 5.1.2 Méthode par séparation et évaluation

Pour plusieurs problèmes, en particulier les problèmes d'optimisation, l'ensemble de leurs solutions est fini dénombrable. Il est donc possible d'énumérer toutes ces solutions et ensuite choisir la meilleure. L'inconvénient majeur de cette approche est le nombre très important de solutions : il n'est guère évident d'effectuer cette énumération.

Les méthodes par évaluation et séparation consistent à énumérer ces solutions d'une manière intelligente en ce sens qu'en utilisant certaines propriétés du problème en question, cette technique arrive à éliminer des solutions partielles qui ne mènent pas à la solution que l'on recherche. De ce fait, on arrive souvent à obtenir la solution recherchée en des temps raisonnables. Bien entendu, dans le pire des cas, on retombe toujours sur l'élimination explicite de toutes les solutions du problème. Pour ce faire, ces méthodes se dotent d'une fonction qui permet de mettre une borne sur certaines solutions pour soit les exclure soit les maintenir comme des solutions potentielles. Bien entendu, la performance d'une méthode par Séparation et Evaluation dépend, entre autres, de la qualité de cette fonction (de sa capacité d'exclure des solutions partielles tôt).

### 5.1.3 Méthode par programmation linéaire

Dans les méthodes de la programmation linéaire, le problème est formulé de telle sorte que le critère et les contraintes s'expriment comme des fonctions linéaires des variables de décision. En ordonnancement, les variables sont souvent entières, voir binaires, ce qui rend les problèmes beaucoup plus difficiles à résoudre et limite les résolutions exactes à des instances de petite taille.

### 5.2 Méthodes Approchées

La résolution des problèmes NP-difficiles est souvent lourde d'application. On entend par là que l'utilisation des méthodes exactes de résolution est parfois possible mais ces dernières peuvent être désavantageuses en terme de temps par exemple, c'est-à-dire l'on ne peut connaître à priori le nombre exacte d'itérations nous menant à l'optimum, et souvent le temps écoulé à chercher une solution pour prendre une décision est lui-même aussi important que cette décision. De tels problèmes sont très fréquemment rencontrés dans la vie réelle et dans certains cas, les décideurs ont besoin de solution immédiate. Les chercheurs se sont donc tournés vers d'autres techniques de résolution dites *heuristiques*. Ces heuristiques sont en fait des procédures intuitives (empiriques) propres à chaque problème, conçues étape par étape en utilisant le bon sens et la logique et s'orientant selon les préférences de décideur, de sorte à trouver une solution aussi satisfaisante que possible. En effet, les heuristiques, à la différence des méthodes exactes, sont souvent construites à partir de l'expérience ou d'analogies plutôt que d'une analyse scientifique qui prend en compte un maximum d'éléments et qui serait difficilement exploitable.

Une heuristique reste cependant spécifiquement élaborée pour un type précis de problèmes contrairement aux métaheuristiques, qui peuvent être appliquées à différentes catégories de problèmes à condition d'en trouver une bonne adaptation à ces derniers.

Ce qui est à retenir concernant les heuristiques est qu'elles sont des méthodes approximatives (non exactes) conçues en vue d'atteindre des solutions satisfaisantes en un temps réduit par rapport à l'application d'une méthode exacte, qui ne convergerait pas assez vite vers un optimum. Aussi, il est important de souligner que ces méthodes sont fondées sur une certaine logique intuitive propre à chaque chercheur et adoptée face à une problématique précise.

### 5.2.1 Les métaheuristiques

Depuis quelques décennies, les ingénieurs et chercheurs étaient déjà confrontés à des problèmes d'optimisation grandissants, auxquels ils ne trouvaient pas de techniques de résolution satisfaisantes en termes de temps de calcul et de convergence vers une solution optimale. Dés le début des années 80, s'inspirant de mécanismes naturels (physiques, biologiques ou éthologiques) les entourant, ils développent par analogie aux phénomènes observés dans leur milieu de vie direct, des procédés d'exploration des ensembles de solutions réalisables des problèmes difficiles de l'optimisation combinatoire. C'est l'apparition des métaheuristiques. Les métaheuristiques constituent une classe de méthodes très puissantes pour l'optimisation combinatoire et l'affectation sous contraintes de grande échelle. Ces méthodes ont permis de trouver des solutions de bonne qualité en temps raisonnable à des problèmes combinatoires réputés difficiles. On les trouve sous deux types:

- L'exploration par méthodes de descente (amélioration itérative par visite de voisinage) telles «la recherche tabous» ou «le recuit simulé».
- L'application de métaheuristiques dites «distribuées», qui itèrent sur un échantillon (de taille fixe) de solutions, comme «colonies de fourmis» ou «les algorithmes évolutionnaires».

Les méthodes de descentes (appliquées pour des problèmes de minimisation), par «recherche avec tabous» ou par «recuit simulé», sont les premières à avoir été utilisées; elles itèrent séquentiellement sur une solution réalisable dont elles tentent d'améliorer la valeur de l'objectif, consultant un voisinage (la notion de voisinage doit être précisément définie au préalable) de la solution courante, mais peuvent cependant souvent être piégées dans des minimums locaux. Même l'application répétée d'une telle procédure est particulièrement

inefficace si le nombre de minimums locaux croit exponentiellement avec la taille du problème.

Quant aux méthodes d'exploration distribuée telles que «les algorithmes évolutionnaires» dits aussi «génétiques» et les «algorithmes de colonie de fourmis», elles itèrent progressivement sur des ensembles finis de solutions en partant d'une population initiale non homogène aléatoirement choisie, et agissent, d'une itération à la suivante, sur les caractères représentés dans l'intégralité de celle-ci. Elles exigent cependant une bonne construction des différents opérateurs régissant le passage d'une population vers une nouvelle qui soit relativement meilleure.

## 6. Conclusion

Dans ce chapitre, nous avons présenté en premier lieu, quelques définitions de bases liées à la théorie des graphes ainsi que la complexité des algorithmes et des problèmes d'optimisation combinatoires. En second lieu, nous avons exposé la complexité des différentes classes des problèmes d'ordonnancement qui sont, en général, NP-difficile. Ensuite, nous avons présenté, quelques cas particuliers des problèmes d'ordonnancement qui peuvent êtres résolus par des algorithmes polynomiaux ou pseudo-polynomiaux. Dans le chapitre suivant, nous nous focaliserons sur le job shop simple afin d'introduire par la suite le problème du job shop avec blocage qui va faire l'objet de notre étude et contribution dans cette thèse.

# Chapitre 3
# Le Job Shop Simple et la Contrainte de Blocage

## 1. Introduction

Le problème d'ordonnancement de type job shop est un cas particulier du problème général d'ordonnancement d'atelier qui généralise, à son tour, le problème de flow shop. Le problème de job shop est défini dans les ateliers à cheminement multiple dans lesquels l'ordre de passage des opérations sur les différentes machines peut être différent; à l'inverse du flow shop qui exige que le passage de toutes les opérations sur les différentes machines se fasse dans le même ordre de passage.

## 2. Définition du Job shop simple

Soit un atelier composé de *m* machines qui doivent traitées *n* jobs. Cet atelier est soumis aux hypothèses suivantes :

- Les machines sont disponibles en un seul exemplaire, pendant toute la période de l'ordonnancement, i.e. les pannes ne sont pas prises en compte.
- Les machines sont indépendantes les unes des autres (pas d'utilisation d'outil commun).
- Une machine *k* ne peut traiter plus d'un job *j* à un instant donné.
- chaque job *j* est caractérisé par une séquence de $v_j\ (j=1,...,n)$ opérations qui doivent être traitées dans cet ordre ; ce qui exprime l'existence d'une contrainte de précédence entre les opérations d'un même job.
- Chaque job *j* nécessite un temps de traitement $p_{jk}$ sur une machine *k*.
- Les jobs sont indépendants les uns des autres. En particulier, il n'existe aucun ordre de priorité attaché aux jobs.
- Les jobs sont autorisés à attendre sur les ressources autant qu'il faut.
- Une opération en cours d'exécution ne peut être interrompue (pas de préemption).

- 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.
- Il n'y a pas de dates échues à respecter pour tous les jobs.
- Le routage de chaque job est respecté, donc, il n'existe pas d'évènements aléatoires pendant l'exécution.
- Le processus est sans répétition, c'est-à-dire, que chaque opération ne peut visiter une machine qu'une seule fois pendant le processus de fabrication.

Il s'agit donc de trouver un ordre de passage des opérations sur les machines qui minimise le temps total de traitement ou de production, en respectant les contraintes ci-dessus. Ce qui est équivalent à minimiser le temps de sortie de la dernière opération de la chaîne de fabrication, nommé le *makespan*, noté *Cmax*. Sous ces conditions ce problème est appelé job shop simple.

### 3. Formulation Mathématique

Afin de décrire le problème d'une façon univoque, nous le faisons dans un premier temps à l'aide du formalisme mathématique. Une première formulation, qui revient souvent dans la littérature est la suivante :

$t_i$ : La date de début de l'opération $O_i$

$p_i$ : Le temps de traitement de l'opération $O_i$ .

$A$ : C'est l'ensemble des contraintes cumulatives.

$E_k$ : C'est l'ensemble des opérations concurrentes sur la même machine $k$.

(1) $t_i + p_i \le t_j \qquad \forall (O_i, O_j) \in A$

(2) $(t_j - t_i) \ge p_i \;\; ou\, bien \;\; (t_i - t_j) \ge p_j \;\; \forall (O_i, O_j) \in E_k \;\; \forall k \in \{1,...,m\}$

(3) $t_i \ge 0 \quad \forall i \in O$

Dans cette formulation, une seule des deux inéquations de (2) doit être valide. L'objectif est de minimiser le *makespan*. Les contraintes (1), décrivent l'ordre de précédence des opérations d'un même job. Les contraintes (2), décrivent l'ordre de précédence des opérations concurrentes sur la même machine *k*. Cet ordre est à définir.

### 3.1 La programmation linéaire

La formulation présentée dans la section ci-dessus est une formulation disjonctive [59]. Elle est dû au fait qu'elle contient des contraintes disjonctives liées par un *ou* logique. Cette disjonction peut aisément être enlevée en introduisant la variable binaire $x_{ij}$ :

$$x_{ij} = \begin{cases} 1 \ \text{Si l'opération } \mathrm{O_j} \text{ est réalisée avant l'opération } \mathrm{O_i} \\ 0 \ \textit{Sinon} \end{cases}$$

$H$ est une constante suffisamment grande ( $H \geq \sum p_i$ ).

Les contraintes du problème sont exprimées par les inéquations suivantes:

(1) $t_i + p_i \leq t_j \qquad \forall (O_i, O_j) \in A$

(2) $x_{ij} \in \{0,1\} \qquad \forall (O_i, O_j) \in E_k \quad \forall k \in \{1,...,m\}$

(2') $(H + p_i)x_{ij} + (t_j - t_i) \geq p_i \qquad \forall (O_i, O_j) \in E_k \ \ \forall k \in \{1,...,m\}$

(2'') $(H + p_j)(1 - x_{ij}) + (t_i - t_j) \geq p_j \quad \forall (O_i, O_j) \in E_k \ \ \forall k \in \{1,...,m\}$

(3) $t_i \geq 0 \quad \forall O_i \in O$

(4) $Z \geq t_i + p_i \quad \forall O_i \in O$

min (Z)

La contrainte (1) définie la précédence entre les opérations d'un même job. Les contraintes (2), (2') et (2'') définissent l'ordre des opérations sur la même machine (ces contraintes sont équivalentes à la contrainte (2) de la formulation présentée dans la section 3). La contrainte (4) nous permet de calculer Z.

### 3.2 Les graphes disjonctifs

Pour une instance d'un problème de job shop général, le graphe disjonctif G = ($V$, $C \cup F$) est défini comme suit :

$V$ : est l'ensemble des sommets représentant les opérations de tous les jobs. Deux sommets additionnels: source $S \in V$ et puits $T \in V$ , représentant des opérations fictives.

Un poids est associé à chaque sommet, qui est nul pour $S$ et $T$ et égal au temps d'exécution de l'opération correspondante $p_v$ pour chaque sommet $v$.

$C$ : est un ensemble d'arcs orientés dit : arcs conjonctifs. Ces arcs représentent les relations de précédence entre les opérations d'un même job.

Il y a également des arcs conjonctifs entre :

- La source *S* et tous les sommets *v* ayant les $dg^{-}(v) = 0$.
- Les sommets *y* ayant $dg^{+}(y) = 0$ et le sommet *puits T*.

*F* : est l'ensemble d'arcs dit : arcs disjonctifs. Ces arcs sont associés aux conflits d'utilisation d'une ressource non partageable. De tels arcs non orientés relient chaque paire d'opérations devant s'exécuter sur la même machine.

Pour chaque arc conjonctif (*i,j*) qui est orienté, on a : $t_j - t_i \geq p_{ij} \quad \forall (i, j) \in C$. Pour chaque arc disjonctif (*y,k*) qui est non orienté, nous avons :

$$t_y - t_k \geq p_{yk} \text{ ou } t_k - t_y \geq p_{ky} \quad \forall (y,k) \in F .$$

L'ensemble des paires de disjonction associées à une même machine forme une clique de disjonction. Un ordonnancement admissible est caractérisé par le choix du sens d'un arc dans chaque paire de disjonction. Ce choix d'orientation ne doit pas engendrer de circuits de longueur positive dans le graphe *G*.

Une orientation arbitraire des arcs disjonctifs est appelée : *sélection*. Si toutes les disjonctions sont arbitrées de manière cohérente (*G n'admet pas de circuit de poids positif*), on dit que la sélection est *valide* et on obtient un ordre de passage des opérations sur chacune des machines. Une *sélection* peut être non valide si elle impose des ordres qui ne sont pas compatibles (*G admet un circuit de poids positif*).

Dans la modélisation par les graphes disjonctifs, Il faut utiliser les algorithmes de recherche de plus long chemin dans les graphes pour évaluer un ordonnancement. Il ne faut pas donc confondre les graphes contenants des circuits de poids nul et ceux qui contiennent des circuits absorbants.

**Exemple**

Soit à ordonnancer 3 jobs $J_1, J_2, J_3$ sur 3 machines $M_1, M_2, M_3$.

| Les jobs | La séquence des machines | Les durées sur les machines |
|---|---|---|
| $J_1$ | $M_1$, $M_2$, $M_3$ | 5,3,8 |
| $J_2$ | $M_1$, $M_3$ | 4,6 |
| $J_3$ | $M_3$,$M_2$ | 2,7 |

Tableau1. Exemple d'un job shop

Pour simplifier l'exemple, nous considérons l'ensemble de toutes les opérations de $O_1$ à $O_7$ :

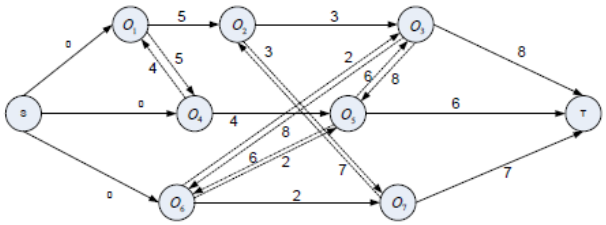

Figure 8. Graphe disjonctif

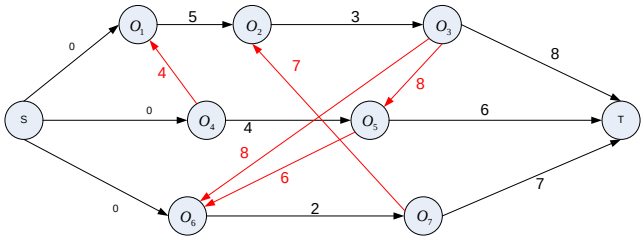

Figure 9. Exemple d'une *sélection* (les arcs en rouge forment une sélection)

### 4. Complexité du problème job shop

La complexité d'un problème d'ordonnancement de type job shop simple est connue dans la littérature pour être généralement NP-difficile [15]. En effet, Sotskov & Shakhlevich [80] ont montré que les problèmes $J3/n=3/C\max$ et $J3/n=3/\sum C_i$ sont NP-difficile. De même, Lenstra & Rinnooy Kan [54] ont montré que les problèmes *J2/ / Cmax* sont aussi NP-difficile. En revanche, il existe quelques cas particuliers qu'on peut résoudre en temps polynomial.

#### 4.1 Problème du Flow shop

Le problème du flow shop à deux machines F2/ /*Cmax*, avec espace de stockage illimité entre les machines, est polynomial. En effet, il existe un algorithme appelé algorithme de Johnson [46] permettant de le résoudre d'une manière optimale en $O(n\log n)$.

#### 4.1.1 Algorithme de Johnson

Soient $n$ jobs $j$=1,…,$n$ à traiter dans le même ordre sur deux machines A et B ayant respectivement des temps de traitement $p_{Aj}$ sur la machine A et $p_{Bj}$ sur la machine B.

La règle qui minimise la durée totale des opérations est appelée *règle de* Johnson et fonctionne comme suit :

- Classer les jobs en deux sous ensembles $J_L$ et $J_R$.

- $J_L$ contient les jobs vérifiant $p_{Aj} \leq p_{Bj}$.
- $J_R$ contient les jobs vérifiant $p_{Aj} > p_{Bj}$.
- Les jobs de $J_L$ sont ordonnés suivant la règle SPT (Shorted Processing Time).
- Les jobs de $J_R$ sont ordonnés suivant la règle LPT (Longest Processing Time).
- La séquence optimale est donnée par l'ordre $J_L$,$J_R$.

Gilmor & Gomory [33] ont étudié le problème de flow shop à deux machines avec la contrainte de sans-attente F2/*nwt* /*Cmax* ; c'est-à-dire que dès qu'une machine termine le traitement d'un produit, ce dernier doit être immédiatement transféré à la prochaine machine sans quoi des dommages peuvent être causés aux pièces. Ils ont démontré que ce problème est équivalent à un cas particulier du problème de voyageur de commerce (PVC) à *n*+l villes qu'ils résolvent par un algorithme polynomial en *O*(*n*log*n*). Par contre, la version à trois machines a été démontrée dans la littérature par Garey et al. [30] qu'elle est NP-difficile.

Cependant, le problème de flow shop à deux machines sans espace de stockage entre celles-ci, appelé flow shop à deux machine avec la contrainte de blocage et noté F2/*block*/*Cmax*, est équivalent au problème F2/*nwt* /*Cmax*. Par contre, ils sont différents à plus de deux machines.

### 4.1.2 Algorithme de Gilmor et Gomory 1964

Soit *n* jobs devant être traités sur deux machines *A* et *B* dans le même ordre. Soient $B_i$ le temps de traitement du job *i* sur la machine B, et $A_i$ le temps de traitement du job *i* sur la machine *A*.

(1) Classer les $B_i$*(i=1 à n)* en ordre croissant et numéroter les jobs en accord avec ce classement.

(2) Classer les $A_i$ *(i=1 à n)* en ordre croissant et déduire la permutation $\phi$ telle que $\phi(p)=q$ tel que $A_q$ est le $p^{\text{éme}}$ plus petit élément des $A_i$.

(3) Construire un graphe à *n* nœuds avec des arcs non orientés reliant le $i^{éme}$ nœud au $\phi^{\text{éme}}$ nœud ; si le graphe n'a qu'une seule composante, passer à l'étape (9).

(4) Calculer le coût des arcs $R_{i,i+1}$ *(i=1 à n-1)* reliant deux nœuds consécutifs selon $c_{i,i+1} = \max \{0, (\min \{B_{i+1}, A_{\phi(i+1)}\} - \max \{B_i, A_{\phi(i)}\}\}\}$.

(5) Ajouter les arcs $R_{i,i+1}$ à tour de rôle selon l'ordre croissant de leur coût, jusqu'à ce que le graphe n'ait qu'une seule composante connexe.

(6) Diviser les arcs ajoutés à l'étape (5) en deux groupes ; ceux dont $\sum A_{\phi(i)} \geq \sum B_i$ appartiennent au groupe 1, et ceux dont $\sum A_{\phi(i)} < \sum B_i$ appartiennent au groupe 2.

(7) Trouver le plus grand indice $i_1$, tel que $R_{i_1,i_1+1}$ appartient au groupe 1, le deuxième plus grand indice $i_2$…jusqu'à $i_l$ en considérant qu'il y a *l* éléments dans le groupe 1.

(8) Trouver le plus petit indice $j_1$ tel que $R_{j_1,j_1+1}$ appartient au groupe 2, le deuxième plus grand indice $j_2$…jusqu'à $j_m$ en considérant qu'il y a $m$ éléments dans le groupe 2.

(9) La tournée minimale est obtenue en faisant suivre le $i^{éme}$ job par l'ordre de fabrication $\psi^*(i)$,

où : $\psi^*(i) = \phi(\alpha_{i_1,i_1+1}(\alpha_{i_2,i_2+1}\dots(\alpha_{i_l,i_l+1}(\alpha_{j_1,j_1+1}(\alpha_{j_2,j_2+1}\dots(\alpha_{j_m,j_m+1}(i)))))))$

avec : $\alpha_{pq}(p) = q$, $\alpha_{pq}(q) = p$ et $\alpha_{pq}(i) = i$ (pour $i \neq p,q$).

Malheureusement, ces méthodes ne peuvent être généralisées aux cas à plus de deux machines. Ainsi, le problème du flow shop à $m$ machines F$m$/ /$Cmax$ est NP-difficile.

Pour cette raison, beaucoup d'auteurs se sont orientés vers les heuristiques. Celles-ci donnent des solutions approchées, généralement de bonne qualité.

L'une des toutes premières heuristiques est celle de Palmer [69] qui classe les jobs selon l'ordre décroissant d'un indice $S_j$ qu'il a définit et appelé *Slope index*.

L'heuristique de Dannenbring [24] ramène le tout à un problème fictif à deux machines auquel on applique la méthode de Johnson.

L'heuristique de Hundal et Rajgopal [44] choisit la meilleure des trois solutions obtenues en classant les jobs selon l'ordre décroissant de l'indice de Palmer $S_j$ et deux autres indices $S_{j1}$ et $S_{j2}$ qu'ils ont définit.

La méthode de Nawaz, Enscore et Ham [65] (NEH) consiste à résoudre le problème des deux premiers jobs et à trouver la meilleure position pour incorporer les jobs suivants ($j$=3,…,$n$) dans la séquence partielle.

La méthode de Campbell, Dudeck et Smith [16] (CDS) consiste quant à elle à résoudre $m$-1 problèmes fictifs à deux machines, avec des temps de traitement des jobs qu'ils définissent pour chaque sous problème $k$.

Une autre heuristique populaire pour le problème de minimisation de la durée totale des opérations du Flow shop à $m$ machines est celle de l'ajustement du profil (PF) [62] qui fonctionne comme suit :

On sélectionne d'abord un job pour débuter la séquence, par exemple celui dont la somme des temps de traitement est la moins élevée. Évidemment, il ne rencontre aucun blocage dans son passage d'une machine à l'autre, générant ainsi un profil. Pour déterminer quel sera le deuxième job à procéder, chaque job restant est testé. Pour chaque candidat, est calculée la somme des temps morts aux machines (ou la somme des temps de blocage des produits). Le

candidat au plus bas total est sélectionné, créant un nouveau profil. On répète ensuite la procédure pour trouver le troisième, quatrième, jusqu'au n$^{ème}$ job de la séquence.

### 4.2 Problème du Job Shop

Le problème de minimisation de la durée totale des opérations d'un job shop à *m* machines J*m*/ /*Cmax*, étant une généralisation du flow shop à *m* machines, est donc fortement NP-difficile. Les procédures de résolution sont alors basées sur l'énumération des solutions ou sur les heuristiques. Les techniques fréquemment utilisées sont les méthodes par séparation et évaluation progressives SEP, ainsi que l'heuristique du goulot changeant (Shifting bottlenek) [1] dont l'expérience a démontré qu'elle donne de très bons résultats.

Dans ce qui suit, nous présentons quelques job-shop particuliers, à deux machines ou à deux jobs, pour lesquels il existe des algorithmes polynomiaux ou pseudo-polynomiaux.

#### 4.2.1 Job shop à deux machines

Le premier cas particulier est le job shop à deux machines avec au plus deux opérations par job, noté: $J2/v_j \leq 2/C\max$. Ce problème peut être résolu d'une manière optimale grâce à l'algorithme de Jackson [45] qui permet de le ramener aisément au F2//*Cmax*.

**Algorithme de Jackson**

Soient :

- $J_{AB}$ l'ensemble des jobs qui doivent passer par la machine A en premier, ensuite la machine B. $J_{AB}$ est ordonné suivant la règle de Johnson.
- $J_A$ l'ensemble des jobs qui passent par la machine A uniquement, ordonnés d'une manière quelconque.
- $J_{BA}$ l'ensemble des jobs qui doivent passer par la machine B en premier, ensuite la machine A. $J_{BA}$ est ordonné suivant la règle de Johnson.
- $J_B$ l'ensemble des jobs qui passent par la machine B uniquement, ordonnés d'une manière quelconque.

La séquence optimale d'exécution est : $J_{AB}$, $J_A$, $J_{BA}$ sur la machine A, et $J_{BA}$, $J_B$, $J_{AB}$ sur la machine B.

Le cas deuxième est : le problème *J*2/*n*=*k*/*Cmax* qui est un job shop à deux machines ayant un nombre de jobs *k* fixé à l'avance avec ou sans répétition. Brucker (2007) a présenté un algorithme polynomial pour le résoudre. Pour cela, il a développé une transformation de ce

problème en un problème de plus court chemin dans un réseau sans circuit. Il a également présenté un algorithme polynomial pour la résolution du *J*2/*n*=3/*Cmax* sans répétition.

Le troisième cas particulier, c'est le job shop $J2/p_{j,k}=1/L\max$. En effet, il peut être résolu avec un algorithme polynomial en $O(r\log r)$ [15]. De plus, Brucker [15] a utilisé une technique de Hachage qui lui a permis de déduire un autre algorithme en $O(r)$.

### 4.2.2 Job shop à deux jobs

Le job shop considéré dans cette section possède *m* machines et exactement deux jobs. La résolution de $J/n=2/C\max$ peut être effectuée en temps polynomial grâce à l'approche géométrique proposée pour la première fois par Akers et Friedman, 1955 [7]. Elle consiste à réduire ce problème en un problème de recherche de plus courts chemins dans un plan encombré d'obstacles représentant l'exécution simultanée des deux jobs sur la même machine.

Il existe plusieurs autres extensions de l'approche géométrique classique dans la littérature, développées pour des problèmes à deux jobs plus généraux [15].

## 5. La Contrainte de Blocage

### 5.1. Définition

La contrainte de blocage se pose en général dans les problèmes d'ordonnancement d'atelier de type flow shop ou job shop, dans lesquels il existe une contrainte de limitation ou d'inexistence d'espace de stockage sur les machines.

Un problème d'ordonnancement de type job shop avec la contrainte de blocage est un problème d'ordonnancement de type job shop simple (déjà vu dans la section 2 de ce chapitre) avec une contrainte en plus qui est la limitation ou l'inexistence d'espace de stockage sur les machines composant l'atelier.

Généralement, dans un atelier de production, deux opérations peuvent être réalisées en séquentiel ou en parallèle. Ceci dépend du type des ressources disponibles dans l'atelier: ou les ressources sont partageables ou elles ne le sont pas. Dans le cas où elles ne sont pas partageables, alors une règle de priorité doit être définie entre ces opérations afin d'organiser leur passage sur la même machine. Jusque là, on ne parle pas d'espace de stockage des machines car on suppose qu'il est infini. Ce problème, décrit un job shop simple.

Par contre, si l'atelier de production se compose de machines non partageables qui n'admettent pas d'espace de stockage pour les opérations traitées, alors le passage séquentiel des opérations sur les machines risque d'être bloqué. En effet, une opération qui a terminé son traitement sur la machine $k$ ne peut commencer son prochain traitement sur la machine suivante $k$+1 que si celle-ci est libre. Dans ce cas, puisque les machines $k$ et $k$+1 n'admettent pas d'espace de stockage pour opérations traitées, alors l'opération se trouvant sur la machine $k$ va rester sur celle-ci, malgré l'achèvement de son traitement, jusqu'à ce que la machine suivante $k$+1 se libère. On dit alors que la machine $k$ est bloquée.

### 5.2. Quelques exemples d'application

Il existe dans la réalité des problèmes totalement différents de l'ordonnancement dans leurs structures et leurs définitions, mais qui se modélisent de la même manière que le problème d'ordonnancement avec blocage. Citons dans ce qui suit, trois exemples d'application. Le premier concerne l'organisation de la circulation des trains dans un terminus [28]. Le second concerne l'organisation des blocs opératoires dans un hôpital [71] et le dernier concerne l'organisation du chargement et déchargement de conteneurs dans un port [66].

#### 5.2.1 Organisation de la circulation des trains dans un terminus

Généralement, un terminus de trains se compose de :

- Trains : qui arrivent, qui quittent le terminus ou bien qui peuvent stationner un temps déterminé pour plusieurs raisons (panne, attente du temps de départ…).
- Blocs de différentes longueurs : un bloc est un segment de chemin de fer.
- Routes : une route est constituée d'un ensemble de blocs.
- Points d'entrée et de sortie du terminus.
- Plateformes : une plateforme correspond au lieu de descente et de montée des voyageurs.
- Signaux lumineux : qui prennent la couleur rouge ou verte.

Le terminus est un passage obligatoire pour un train, dans lequel il va changer sa destination après le chargement et/ou déchargement des voyageurs. Cette nouvelle destination a un horaire précis (départ et arrivée). Donc, chaque terminus a son propre calendrier journalier des horaires de départs et d'arrivées des trains, qui est réalisé manuellement par des tables de temps. D'autre part, un retard d'un train conduira automatiquement à un changement de ces tables de temps ce qui engendrera un effort humain considérable. Ce calendrier peut être réalisé d'une manière automatique grâce au modèle suivant:

- Modéliser chaque tronçon de chemin de fer séparé par des signaux lumineux par un *bloc*. Ces signaux lumineux contrôlent les accès aux blocs. La couleur verte du signal indique que le bloc est libre et la couleur rouge indique que le bloc est occupé par un autre train.
- Un ensemble de blocs forment une *route*. Les conditions d'accès d'un train à une route *R* sont les suivantes:
    - Ce train occupe le premier bloc de la route *R*.
    - II n'y a pas un autre train qui traverse une autre *route incompatible* avec la route $R$ : deux routes $R_1$ et $R_2$ se disent incompatibles si elles partagent au mois un bloc commun.
    - Le bloc final de *R* est libre.

Pour faire une analogie avec le problème d'ordonnancement, nous pouvons modéliser les différents trains par les différents jobs, l'ensemble des machines par les routes disponibles dans le terminus. Le chemin du train (job) dans le terminus correspond à une liste ordonnée des ressources (routes) à traverser qui représente la gamme de fabrication dans le problème d'ordonnancement.

La contrainte de blocage apparaît dans ce problème avec la condition d'accès à une route donnée. Ainsi, la résolution du problème d'organisation de la circulation des trains dans un terminus revient à résoudre un problème d'ordonnancement d'atelier à cheminement multiple avec la contrainte de blocage.

### 5.2.2 Organisation des blocs opératoires dans un hôpital

Dans un hôpital, les patients arrivent à l'hôpital d'une manière aléatoire ou sur rendez-vous pour une consultation ou une hospitalisation. Un patient (assimilé à un job) doit faire l'objet d'une occupation de ressources de différents types : humain (les médecins et les chirurgiens) et matériel (la salle de consultation, la chambre d'hôpital, etc.). Le problème de partage de la ressource risque de se poser. Si la ressource est partageable, le problème ne se pose pas (prenant en compte la capacité de la ressource). Par contre, si la ressource n'est pas partageable alors il est nécessaire d'ordonnancer l'utilisation de cette ressource pour éviter des problèmes de blocage. La résolution de ce modèle permet le suivi des flux des malades dans les hôpitaux tout en exploitant le matériel disponible d'une manière optimale.

### 5.2.3 Déchargement de conteneurs dans un port

Soit un terminal dans un port contenant plusieurs conteneurs. Il est composé d'un quai et d'un dépôt de stockage (storage yard). Le quai, à son tour, contient un ou plusieurs _postes d'amarrage pour les bateaux. Le dépôt est utilisé pour y stocker les conteneurs temporairement. Ce stockage dépend des temps d'arrivées des bateaux, de la disponibilité des grues et des véhicules dédiés à leur transport.

Les conteneurs sont stockés en blocs. Chaque bloc est composé de six lignes et chaque ligne contient une vingtaine de conteneurs avec une hauteur de quatre à cinq conteneurs.

Dès que le poste d'amarrage est désigné, un quai ayant une grue libre (disponible) est alloué pour charger les conteneurs sortants et décharger les autres entrants.

L'opération de chargement et de déchargement est effectuée par le levage, l'empilement et le dépilement des conteneurs, supervisée par un agent maritime.

Le problème qui se pose est l'attribution et la programmation des grues disponibles, des véhicules disponibles, des postes d'amarrage, de l'espace de stockage, de la manipulation des équipements et l'allocation des différentes ressources présentes, afin d'effectuer l'opération de chargement et de déchargement sans blocage.

## 5.3 Etat de l'art

Les problèmes de type job shop avec la contrainte de blocage a intéressé beaucoup de chercheurs ces dernières années à cause de ces applications dans l'industrie, l'agriculture, les hôpitaux, les ports, etc. Sa complexité est, généralement, NP-difficile [42]. Ceci est dû au fait que nous ne connaissons pas d'algorithmes polynômiaux pour les résoudre. Donc leur résolution, d'une manière optimale, passe obligatoirement par les méthodes exactes non-polynomiales.

La première méthode exacte développée pour ces problèmes, sans la contrainte de blocage, a été proposée par Fisher et Thompson [27]. Ils ont développé plusieurs méthodes, mais sans résoudre les exemples de taille 10×10 et plus. Il a fallu attendre 1986 pour voir la première méthode exacte qui a pu résoudre les benchmarks de taille 10×10 de Muth & Thompson [64]. Cette dernière a été développée par Carlier & Pinson [19] et est basée sur la résolution des sous problèmes à une machine. D'autres méthodes exactes ont été proposées par la suite par plusieurs autres auteurs [15], [59], etc., mais toutes basées sur l'idée de Carlier.

Le premier article qui a présenté une bonne récapitulation de tous les travaux effectués sur la contrainte de blocage est celui de Hall & Sriskandaraja 1996 [42], dans lequel ils ont décrits plusieurs applications des contraintes de sans attente et de blocage. Ils ont passé en revue, aussi, la complexité d'une grande variété de problèmes d'ordonnancement avec les contraintes de blocage et de sans attente. Et ils ont donné plusieurs problèmes qui restent ouverts.

Dans ce qui suit nous présentons les travaux publiés pour le problème du flow shop avec la contrainte de blocage.

Le problème d'ordonnancement flow shop avec trois machines est NP-difficile au sens fort. Pour le cas de deux machines, le problème peut être réduit en un cas spécial du problème de voyageur de commerce (PVC) et le résoudre en temps polynomial par l'algorithme de Gilmor et Gomory [33]. Pour m>2 le problème flow shop est considéré comme NP-difficile, pour cela plusieurs heuristiques sont développées pour résoudre ce problème : Xianpeng Wang et Lixin Tang [88] ont proposé un algorithme de Recherche Taboue pour résoudre le problème flow shop hybride avec capacité de stockage limitée où chaque job doit passer par N étages et chaque étage i contient $M_j$ machines parallèles. Pour la construction d'une solution réalisable, ils ont utilisé une procédure GCP (Greedy Constructive Procedure) qui considère les espaces de stockage entres deux étages successifs comme des étages contenant des machines parallèles et les temps de traitement de chaque job sur ces machines sont nuls.

Un algorithme PSO hybride utilisant la procédure NEH, pour construire une solution initiale, a été proposé par Bo Liu, Ling Wang et Yi-Hui Jin [56] pour résoudre le problème d'ordonnancement flow shop avec capacité de stockage limitée. Ronconi et Henriques [77] ont étudié la minimisation du retard total pour le problème d'ordonnancement flow shop avec contrainte de blocage. Pour cela, un algorithme (FPD : Fitting Processing timess and Due dates) et deux métaheuristiques GRASP (Greedy Randomized Adaptative Search Procedures) ont été proposés pour trouver une solution approchée.

Grabowski et Pempera [37] ont utilisé la notion des blocks et anti-blocks pour construire un algorithme de Recherche Taboue pour la résolution du problème d'ordonnancement flow shop avec contrainte de blocage.

Wang et. al. [89] ont proposé un algorithme génétique hybride pour le problème d'ordonnancement flow shop avec capacité de stockage limitée. Pour construire une population initiale, ils ont utilisé la méthode NEH où la procédure de recherche locale est

appliquée sur la population avec une probabilité $(1 - p_m)$ où $p_m$ est la probabilité de mutation.

Martinez et. al. [58] ont étudié la complexité du problème flow shop avec la nouvelle contrainte de blocage où la machine *k* reste bloquée jusqu'à ce que le job commence son traitement sur la machine k+2. Ce problème, qui se pose dans l'industrie et l'agriculture, peut être modélisé par un graphe disjonctif.

Soukhal et. al. [81] ont étudié la complexité du problème d'ordonnancement flow shop avec contrainte de transport avec deux machines. Les jobs traités sur les deux machines sont transportés par des camions de capacité limitée. Dans leur article, ils ont mis en exergue la difficulté de quelques cas particuliers avec la contrainte de blocage entre les machines.

Soukhal & Martineau [82] ont proposé un modèle mathématique et un algorithme génétique pour résoudre le problème d'ordonnancement flow shop avec un robot relié avec toutes les machines. Le robot peut traiter un seul job à la fois.

Ronconi [78] a proposé une heuristique constructive appelée MinMax pour résoudre le problème flow shop avec contrainte de blocage. Cette heuristique est comparée avec d'autres algorithmes comme l'algorithme PF, l'algorithme NEH, etc.

Thornton & Hunsucker [86] ont proposé un algorithme appelé : NIS-heuristique (No Intermediate Storage), pour la résolution du problème d'ordonnancement flow shop avec processeurs multiples et indisponibilité d'un espace de stockage. Les jobs doivent passer par *m* étages et chaque étage *l* contient $M_l$ machines parallèles. L'algorithme est comparé avec d'autres heuristiques qui résolvent le même problème.

Caraffa et. al. [17] ont utilisé un algorithme Génétique pour résoudre le problème d'ordonnancement flow shop avec contrainte de blocage. Ils ont utilisé le lien entre la contrainte de blocage et la contrainte de sans attente pour construire une fonction d'évaluation de la fitness des éléments générés par l'algorithme génétique.

***Pour le problème job shop avec contrainte de blocage :***

Mascis & Pacciarelli [59] ont proposé une modélisation par graphe alternatif avec quelques propriétés sur cette catégorie de graphes qui généralise les graphes disjonctifs. Des heuristiques et une méthode par séparation et évaluation sont décrites dans cet article pour le problème d'ordonnancement job shop avec contraintes de blocage et de sans attente.

Mati et. al. [60] ont proposé une Recherche Taboue pour le problème d'ordonnancement job shop avec contrainte de blocage.

Ait Zai, Benmejdoub & Boudhar [9], [3] ont développé une méthode heuristique en utilisant PSO pour résoudre le problème du job shop simple avec la contrainte de sans attente. En plus, ils ont adapté une méthode SEP et un algorithme génétique pour la résolution du problème d'ordonnancement avec les contraintes de blocage et de sans attente.

Gröflin & Linkert [39] et [40] ont développé un algorithme de Recherche Taboue pour le problème d'ordonnancement job shop généralisé avec contrainte de blocage où le temps d'installation et le temps de nettoyage sont présents.

**Remarque :** Toutes les méthodes décrites ci-dessus sont des méthodes séquentielles.

### 5.4. Modélisation de la contrainte de blocage

La Modélisation du problème de job shop avec blocage peut se faire de plusieurs manières différentes. Nous allons citer dans ce qui suit : la modélisation par la programmation linéaire [59], la modélisation algébrique [76] et la modélisation par la théorie des graphes [59]. Dans cette dernière, une nouvelle notion est utilisée : c'est la notion d'arcs alternatifs.

#### 5.4.1 Modèle linéaire

Soient :

- $f_i$ , La date où l'opération $O_i$ quitte sa machine.
- La variable $x_{ij} = \begin{cases} 1 & Si\ l'opération\ O_j\ est\ réalisée\ avant\ l'opération\ O_i \\ 0 & Sinon \end{cases}$
- $H$ est une constante suffisamment grande ( $H \geq \sum_{i=1}^{l} p_i$ ).
- $t_i$ : La date de début de l'opération $O_i$

Les contraintes :

(1) $t_i + p_i \leq t_j \qquad \forall (O_i, O_j) \in A$

(2) $x_{ij} \in \{0,1\} \qquad \forall (O_i, O_j) \in E_k \quad \forall k \in \{1,...,m\}$

(2') $(H + p_i)x_{ij} + (f_j - t_i) \geq 0 \qquad \forall (O_i, O_j) \in E_k \quad \forall k \in \{1,...,m\}$

(2'') $(H + p_j)(1 - x_{ij}) + (f_i - t_j) \geq 0 \qquad \forall (O_i, O_j) \in E_k \quad \forall k \in \{1,...,m\}$

(3) $t_i \geq 0 \quad \forall O_i \in O$

(4) $f_i - t_i \geq p_i \ \forall O_i \in O$

(5) $Z \geq t_i + p_i \quad \forall O_i \in O \ \min Z$

La contrainte (1) définie la précédence entre les opérations d'un même job.

Les contraintes (2), (2'), (2''), (3) et (4) définissent l'ordre des opérations sur la même machine en respectant la contrainte de blocage.

La contrainte (5) nous permet de calculer Z

### 5.4.2. Modélisation algébrique

Cette modélisation très formelle [76], est sans doute la moins utilisée dans le domaine de l'ordonnancement. Un problème d'ordonnancement est noté : ($O$, $R_I$, $R_M$,→, $P$) avec :

- $O = \{o_{ji}, (j= 1,\ldots, n), (i = 1,\ldots, n_j)\}$ est l'ensemble de toutes les opérations.
- $R_I$ est une relation d'équivalence sur $O$ définissant les opérations communes à un job.
- $R_M$ est une relation d'équivalence sur $O$ définissant les opérations communes à une machine.
- → est une relation d'ordre non réflexive sur $O$ définissant les précédences entre les opérations.
- $P=\{p_{ji}, (j=1,\ldots, ,n_i), (i =1,\ldots,n)\}$ est l'ensemble des temps opératoires.

Ce modèle mathématique est très intéressant; mais présente l'inconvénient, de ne pas posséder de méthodes de résolution connues.

### 5.4.3 Modélisation par les graphes alternatifs

Cette section présente une modélisation, par les graphes alternatifs, du problème du job shop avec blocage, développée par Mascis et Pacciarelli [59]. Tout d'abord, nous introduisons les différentes notations utilisées dans ce modèle :

Soit un problème de job shop avec blocage, ayant $n$ jobs et $m$ machines. Chaque job est composé de $v_l$ opérations, avec $l$=1,...,$n$.

$v$ : est le nombre total d'opérations, $v = \sum_{l=1}^{n} v_l$

$O$ : est l'ensemble de toutes les opérations. $|O| = v$

$i$ et $j$ : sont des indices d'opérations quelconques.

$k$ : est un indice désignant une machine.

$l$ : est un indice général.

$J$ : représente l'ensemble de tous les jobs.

$M$ : est l'ensemble de toutes les machines.

$o_{ji}$ : est la $i^{ème}$ opération du job $j$.

$\sigma(o_{ji})$ : est l'opération qui succède directement à l'opération $o_{ji}$ dans un même job.

$m(o_{ji})$ : est l'unique machine qui doit traiter l'opération $o_{ji}$, $m(o_{ji}) \in M$.

$C$ : est l'ensemble d'arcs représentant les relations de précédence entre les opérations d'un même job, c'est-à-dire les arcs conjonctifs.

$A$ : est un ensemble de pairs d'arcs exclusifs dit arcs alternatifs.

$U$ : est l'ensemble de tous les arcs, $U = C \cup A$.

$u_l$ : est le $l^{ème}$ arc de $U$.

$I(u)$ : est l'extrémité initiale de l'arc $u$.

$T(u)$ : est l'extrémité terminale de l'arc $u$.

$G$ $(V,U)$ : est un graphe, tel que $V$ est l'ensemble de tous les sommets représentant les opérations de l'ensemble $O$.

$t_{o_{ji}}$: représente le temps de début de traitement de l'opération $o_{ji}$

$p_{ji}$: est la durée de traitement de l'opération $o_{ji}$ sur la machine $m(o_{ji})$.

En prenant en considération les notations précédentes, le problème du job shop avec blocage peut être modélisé comme suit :

Soit $J$ un ensemble de $n$ jobs noté par $J = \{J_1,..., J_n\}$, qui doivent être traités sur $m$ machines disponibles dans l'atelier de production $M = \{M_1,.., M_m\}$. Chaque job $J_j$ est composé d'une séquence de $v_j$ opérations notée $J_j = o_{j1}, o_{j2},..., o_{j_{v_j}}$ qui doivent être traitées dans cet ordre.

Soit $G = (X, U = C \cup A)$ le graphe définit par l'ensemble des sommets $X$ qui représente l'ensemble de toutes les opérations de $O$, tel que $O = \{o_{jl} / j = 1..n \; et \; l = 1..v_j\}$ est l'ensemble d'arcs, $U=A \cup C$ tel que: $A = \left\{ \begin{array}{c} ((\sigma(o_{jl}), o_{hk}), (\sigma(o_{hk}), o_{jl})), j \neq h \; et \; M(o_{jl}) = M(o_{hk}) \\ (j = 1,...,n), (h = 1,...,n), (l = 1,...,v_j), (k = 1,...,v_h) \end{array} \right\}$ et $C = \{(o_{jl}, \sigma(o_{jl}))/ \; j = 1,...,n \; et \; l = 1,...,v_j\}$. Chaque arc de $C$ représente une contrainte de précédence entre deux opérations consécutives d'un même job $j$.

Pour chaque arc $u \in C$, est associé un poids égal au temps de traitement de l'opération $I(u)$. En plus, il exprime une relation de précédence entre $I(u)$ et $T(u)$. Ce qui veut dire que l'opération de $T(u)$ ne peut commencer avant que l'opération $I(u)$ ne soit achevée. Il s'en suit : $t_{\sigma(o_{jl})} \leq t_{jl} + p_{jl}$.

L'ensemble des arcs alternatifs *A* est utilisé pour représenter la situation de blocage. Une solution au problème d'ordonnancement consiste à faire un choix entre chaque couple d'arcs alternatifs.

Un graphe alternatif est une généralisation des graphes disjonctifs [59]. Soient $o_{jl}$, $o_{ih}$, $\sigma(o_{jl})$ et $\sigma(o_{ih})$ quatre opérations, telles que $o_{jl}$ et $o_{ih}$, sont des opérations concurrentes sur une même machine. Nous avons $M(o_{jl})= M(o_{ih})= M_k$. Dans le cas où la machine $M_k$ n'admet pas d'espace de stockage, le passage séquentiel de l'opération $o_{ih}$ sur $M_k$ dépend de la fin de traitement de l'opération $o_{jl}$ sur la même machine. Pendant ce temps, l'opération $o_{ih}$ restera bloquée jusqu'à ce que l'opération $o_{jl}$ quitte la machine $M_k$. Cette situation est représentée en reliant $\sigma(o_{jl})$ avec $o_{ih}$, $\sigma(o_{ih})$ avec $o_{jl}$ par une paire d'arcs alternatifs $u_1$et $u_2$ tel que : $u_1=(\sigma(o_{jl}),o_{ih})$ et $u_2=(\sigma(o_{ih}),o_{jl})$ avec $(u_1,u_2)\in A$.

La pondération des arcs alternatifs est souvent nulle. Ce qui exprime que les deux opérations $\sigma(o_{jl})$ et $o_{ih}$ peuvent commencer leur traitement en même temps. Idem pour $\sigma(o_{ih})$ et $o_{jl}$ (voir Figure 3.a). Exception faite avec la dernière opération $o_{jv_j}$ d'un job *Jj* donné qu'on appellera opération terminale.

Soit *Jj* une opération terminale d'un job quelconque. Puisque *Jj* n'admet pas de successeur alors $\sigma(o_{jv_j})$ n'existe pas. Dans ce cas, on relie directement les sommets (opérations) avec leurs concurrents sur la même machine en utilisant des arcs alternatifs pondérés (qui sont dans ce cas, des arcs disjonctifs) (voir Figure 10.b). Cette pondération est strictement positive avec une valeur égale à la durée de traitement $p_{jv_j}$ du sommet initial de chaque arc alternatif.

La Figure 10.a montre un exemple d'arcs alternatifs pour des opérations non terminales $o_{jl}$ et $o_{ih}$. Par contre, la Figure 3.b montre un exemple d'arcs alternatifs entre deux opérations $o_{ih}$ et $o_{jv_j}$. Les arcs alternatifs sont représentés par des arcs orientés en pointillé.

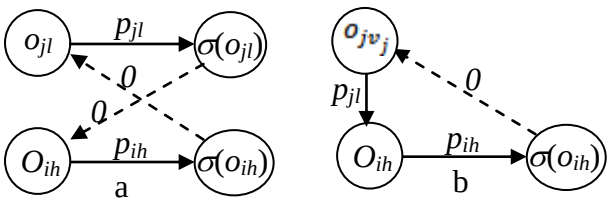


Figure 10. Arcs alternatifs d'une paire d'opérations *i* et *j*

**Définition 1:** On appelle ***sélection*** *S* un ensemble d'arcs alternatifs, choisis parmi les éléments de l'ensemble *A*, sachant que *A* est un ensemble de couples $((\sigma(o_{jl}),o_{hk}),(\sigma(o_{hk}),o_{jl}))$. Si l'arc $(\sigma(o_{jl}),o_{hk})$ est sélectionné dans *S* alors l'arc $(\sigma(o_{hk}),o_{jl})$ sera interdit dans *S*.

**Définition 2 :** Soit *S* une *sélection*, on dit que *S* est ***complète*** ssi elle comporte exactement un arc et un seul arc parmi tous les couples d'arcs constituant l'ensemble *A*. Dans ce cas, la longueur de la sélection *S* est égale au cardinal de *A* et on note $long(S)=|A|$.

**Définition 3 :** Un ordonnancement est une *sélection complète S* d'arcs alternatifs, tels que les arcs de *S* définissent un ordre de passage des différentes opérations concurrentes sur les mêmes machines. Un tel ordonnancement peut être réalisable ou non.

**Définition 4 :** Soit *S* une sélection. Pour chaque couple d'arcs alternatifs $((i,j),(h,k))\in A$, on dit que l'arc $(i,j)$ est sélectionné dans *S* ssi $(i,j)\in S$. On dit que l'arc $(i,j)$ est interdit dans *S* ssi $(h,k)\in S$. Enfin il est possible d'avoir le cas où les deux arcs alternatifs $((i,j),(h,k))\in A$ ne sont pas dans *S*.

**Définition 5 :** Soit *S* une *sélection* d'arcs alternatifs. On note $G(S)=(X,C\cup S)$ le graphe alternatif contenant tous les arcs conjonctifs de *C* et les arcs alternatifs sélectionnés dans *S*. On dit que *S* est une *sélection* ***consistante*** ssi le graphe correspondant $G(S)$ ne contient aucun circuit de longueur positive.

**Définition 6 :** Soit *S* une *sélection consistante*. On appelle *extension*, une *sélection* $S'$ tel que $S'\subseteq S$, si elle existe.

**Définition 7** : Le *makespan* (*Cmax*) d'une *sélection consistante S* est la valeur du plus long chemin du sommet *source* jusqu'au sommet *puits* dans le graphe $G(S)$.

**Définition 8 :** Soient une sélection *S* et le graphe correspondant $G(S)$. Nous noterons la valeur du plus long chemin entre deux sommets $(i,j)$ de $G(S)$ par $l(i,j)$.

**Définition 9 :** Soient *S* une *sélection*, $G(S)$ le graphe correspondant et *i* un sommet de $G(S)$. On note par $r_i = l(source,i)$ la valeur du plus long chemin reliant la *source* du graphe $G(S)$ au sommet *i*. Et on l'appelle *tête* (*head*) du sommet *i*. La valeur $r_i$ représente la date de disponibilité de l'opération correspondante au sommet *i*.

**Définition 10 :** Soient *S* une *sélection*, $G(S)$ le graphe correspondant et *i* un sommet de $G(S)$. On note par $q_i = l(i,puits)$ la valeur du plus long chemin reliant le sommet *i* au dernier sommet *puits* du graphe $G(S)$. Et on l'appelle *queue* (*tail*) du sommet *i*.

On se basant sur les définitions précédentes, un ordonnancement réalisable peut être représenté par une *sélection complète consistante* dans le graphe *G*.

**Remarques**

- Les graphes alternatifs sont le modèle le plus utilisé, car il permet la représentation des données et des contraintes du problème d'une manière permettant d'appliquer beaucoup d'algorithmes de la théorie des graphes.
- Le *makespan* d'un ordonnancement est le *makespan* de la *sélection complète consistante* associée.

### 5.4.4 La notion de circuit

Soit un problème d'ordonnancement avec blocage modélisé sous forme d'un graphe alternatif $G=(X,C\cup A)$. Une solution de ce problème est de fixer un arc de chaque paire alternative composant l'ensemble *A*. Cette solution peut engendrer trois cas possibles :

- Le graphe *G* est sans circuit ; dans ce cas, la solution est acceptée car elle représente un ordonnancement réalisable.
- Le graphe *G* admet des circuits de poids positif ; dans ce cas, la solution est rejetée car l'ordonnancement correspondant n'est pas réalisable.
- Le graphe *G* n'admet que des circuits de poids nul ; dans ce cas, la solution est acceptée. Cette situation est dite : *Swapping*. Un *swap* est nécessaire quand il y a un circuit de poids nul composé de deux opérations ou plus. Chaque opération est obligée d'attendre sur une machine qui est bloqué par l'opération suivante dans le, même circuit. Dans une telle situation, toutes les opérations, dans le circuit, doivent se déplacer simultanément à la machine suivante.

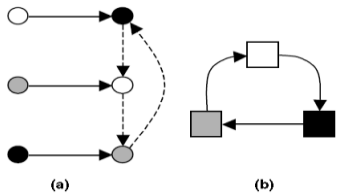


Figure 11. Le cycle de trois opérations bloquées dans un graphe alternatif

La Figure 11 montre un exemple de *Swapping*. La Figure 11.a montre le circuit de poids nul dans le graphe *G*(*S*) et la Figure 11.b montre le swapping sur les machines correspondantes.

## 6. Conclusion

Dans ce chapitre, nous avons introduit le problème du job shop simple. Nous avons exposé ensuite, la contrainte de blocage pour laquelle nous avons donné les différentes formulations

possibles. Pour résoudre ce problème par une méthode exacte, nous avons opté pour la modélisation basée sur les graphes alternatifs. Dans le chapitre suivant, nous détaillerons notre contribution dans le développement de la méthode SEP ainsi que sa parallélisation.

# Chapitre 4
# Résolution Exacte

## 1. Introduction

Une des caractéristiques d'un problème d'optimisation combinatoire est qu'il admet toujours un ensemble de solutions fini et dénombrable. Il est donc possible, en principe, d'énumérer toutes ces solutions, et ensuite prendre celles qui nous arrangent.

Le nombre de solutions possibles d'un problème de la classe NP-complet est très important et leur énumération est une opération impossible en un temps raisonnable.

Les méthodes par séparation et évaluation progressives SEP permettent d'énumérer ces solutions d'une manière intelligente, dans le sens où on utilise certaines propriétés du problème en question, afin d'éviter le parcours des solutions non intéressantes. Cette technique arrive à éliminer des solutions partielles qui ne mènent pas à la solution que l'on recherche. De ce fait, nous obtenons la solution recherchée en des temps raisonnables pour de petites instances. Bien entendu, dans le pire des cas, nous retombons toujours sur l'énumération explicite de toutes les solutions possibles du problème. Cette technique est très utilisée afin de résoudre les problèmes NP-difficile.

## 2. SEP pour le Job Shop Simple JSS

Dans cette section, nous présentons le détail de la méthode SEP développée par Brucker [15], pour résoudre le JSS. Cette méthode utilise un *arbre* de recherche ; initialement, l'arbre ne contient qu'un seul sommet qui est la *racine*, dans lequel aucun arc disjonctif n'est fixé. Ce qui veut dire, qu'il représente toutes les solutions possibles du problème. Les autres sommets, successeurs de la *racine*, sont calculés en fixant des arcs disjonctifs. Ils représentent toutes les solutions du problème respectant les disjonctions déjà fixées. Ensuite, chaque successeur est manipulé récursivement de la même manière. L'exploration des sommets de l'arbre de recherche s'arrête lorsque l'ensemble des arcs fixés dans un sommet représente une sélection complète. Or, cette solution peut ne pas être optimale. Pour cela, nous donnons les détails suivants : Un sommet $r$ du graphe de recherche correspond à un graphe $G(F_r)=(X,C\cup F_r)$, tel que $F_r$ représente l'ensemble des arcs disjonctifs fixés dans le sommet $r$. Ce sommet représente toutes les solutions $Y(r)$ ayant les arcs de $F_r$ déjà fixés. La séparation au niveau du sommet $r$ est faite en subdivisant l'ensemble $Y(r)$ en plusieurs sous ensembles $Y(s_1)$, $Y(s_2)$,...,

$Y(s_q)$. Chaque sous ensemble $Y(s_i)$ ($i$=1,…,$q$) représente l'ensemble des solutions correspondantes au sous problème relatif au graphe $G(F_{s_i}) = (X, C \cup F_{s_i})$, tel que $F_r \subset F_{s_i}$. Ceci veux dire que, $G(F_{s_i})$ est un graphe dérivé de $G(F_r)$ en fixant plus d'arcs disjonctifs. Cette séparation, va créer des sommets, $s_1,...,s_q$, successeurs immédiats au sommet $r$ dans l'arbre de recherche qui sont traités à leur tour récursivement de la même manière.

Pour chaque sommet $r$, une borne inférieure $LB(r)$ est utilisée permettant de borner inférieurement toutes les solutions de l'ensemble $Y(r)$. Nous posons $LB(r)=\infty$ si le graphe correspondant $G(F_r)$ admet un circuit.

Une borne supérieure $UB$ est utilisée : elle représente la meilleure solution réalisable connue jusqu'à présent du problème posé, elle est mise à jour à chaque fois qu'une nouvelle solution réalisable améliorant $UB$ est trouvée.

L'algorithme général est le suivant :

Initialement, nous posons $UB=\infty$ et $F_r=\varnothing$.

*Procédure Séparation-Evaluation-Progressive(r)*
*Debut*

1. *Calculer une solution* $S \in Y(r)$ *en utilisant des heuristiques.*
2. *Si* $(C_{\max}(S) < UB)$ *alors* $UB := C_{\max}(S)$;
3. *Tant que (il existe un successeur s de r non encore examiné)*

*Faire*

4. *Si* $(LB(s) < UB)$ *alors Séparation-Evaluation-Progressive(r);*

*Fin.*

Deux aspects sont importants dans cette procédure :

1- La manière avec laquelle est faite la *séparation*.

2- Le calcul des bornes.

**2.1 La séparation**

La séparation présentée dans cette partie est celle utilisée par Grabowski et al. [36]. Cette séparation est basée sur l'ordonnancement à une machine (one machine scheduling) avec les dates de disponibilité et les dates échues.

Soit $S$ une sélection et $G(S) = (X, C \cup S)$ le graphe correspondant. Soit $P$ le chemin critique trouvé dans $G(S)$ et $L(S)$ sa longueur. Une séquence composée des sommets successifs $u_1,.., u_k$ de $P$ est dite un *block*, si les deux propriétés suivantes sont satisfaites:

1. La séquence contient au moins deux sommets.
2. La séquence contient le maximum d'opérations devant être traitées sur la même machine.

Posons $B_j$, le $j^{ème}$ *block* du chemin critique *P*. La Figure 1 ci-dessous décrit la notion de *block*.

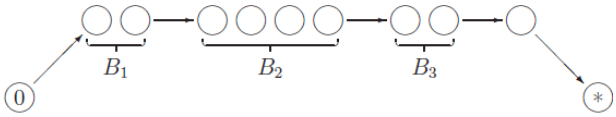


Figure 12. Exemple de *Blocks* dans un chemin critique

**Théorème** [15]: Soit *S* une *sélection* complète correspondant à une solution du problème d'ordonnancement de type job shop simple. Si une autre *sélection* $S'$ existe tel que : $L(S')<L(S)$, alors au moins une opération d'un block *B* de $G(S)$ doit être traitée dans la *sélection* $S'$ avant la première opération ou après la dernière opération du block *B*.

**Corollaire** [15]: S'il existe deux *sélections complètes* $S, S'$ avec $L(S') < L(S)$, alors l'une des deux conditions suivantes est vérifiée:

i- Au moins une opération, différente de la première, du block *B* dans $G(S)$ doit être traitée avant toutes les autres opérations de ce block dans la solution définie par $G(S')$.

ii- Au moins une opération, différente de la dernière, du block *B* dans $G(S)$ doit être traitée après toutes les autres opérations de ce block dans la solution définie par $G(S')$.

Considérons maintenant un sommet *r* de l'arborescence et une solution $y \in Y_r$ calculée généralement en utilisant une heuristique.

Soit *S* une sélection complète correspondant à la solution *y*. Le chemin critique dans $G(S)$ est défini par un ensemble de blocks $B_1,...,B_k$. Pour un block $B_j : u_1^j,...,u_{m_j}^j$, les jobs dans $E_j^B = B_j - \{u_1^j\}$ et $E_j^A = B_j - \{u_{m_j}^j\}$ sont appelés *candidats-avant* et *candidats-après*, respectivement.

Au moins l'un de ces candidats doit être déplacé afin d'améliorer *S*. De plus, nous définissons les ensembles d'arcs suivants : $F_j = \{u_1^j \to l \,/\, l = u_2^j,...,u_{m_j}^j\}$ et

$L_j = \left\{ l \to u^j_{m_j} \,/\, l = u^j_1,...,u^j_{m_j-1} \right\}$ pour $j$=1,...,$k$ et considérons la permutation suivante : $R_1$, $R_2$,..., $R_{2k}$ …..(1) de tous les $F_j$ et $L_j$.

Dans ce qui suit, nous décrivons la manière avec laquelle la séparation $Y(s_1),..,Y(s_q)$ de $Y_r$ est faite grâce à cette technique qui dépend de la formule (1).

Nous construisons les sous-ensembles $Y(s_v)$ $(v=1,...,q)$ en précisant l'ensemble des arcs de $Y_r$ à ajouter à l'ensemble d'arcs $F_r$. Ceci est fait de la manière suivante :

- Trouver, pour chaque *avant-candidat* $l \in E^B_j$ $j=1...k$, un index $m=\alpha(l)$ avec $R_m$=$F_j$ et définir: $S^B_l = R_1 \cup R_2 \cup ... \cup R_{m-1} \cup \left\{ l \to i \,/\, i \in B_j - \{l\} \right\}$.
- Trouver, pour chaque *après-candidat* $l \in E^A_j$ $j=1...k$, un index $m=\alpha(l)$ avec $R_m$=$L_j$ et définir: $S^A_l = R_1 \cup R_2 \cup ... \cup R_{m-1} \cup \left\{ i \to l \,/\, i \in B_j - \{l\} \right\}$.
- Choisir $R = R_1 \cup R_2 \cup ... \cup R_{2k}$

Afin d'illustrer au mieux la construction ci-dessus, nous présentons l'exemple suivant :

**Exemple :**

Soit un chemin critique $P$ avec deux blocks $B_1$ et $B_2$ :

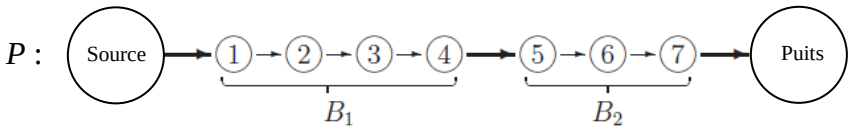


Figure 13. Exemple d'un chemin critique à deux blocks

En considérant la permutation : $R_1 = F_2$ , $R_2 = L_2$ , $R_3 = L_2$ et $R_4 = F_1$, nous obtenons les sous-ensembles d'arcs ci-dessous voir Figure 13.

Nous remarquons que $S^A_5$ et $S^B_4$ contiennent des circuits. Il est aussi possible, lors de la construction des différentes sélections précédentes, qu'un circuit soit créé. Ainsi, s'il y a des circuits qui apparaissent, alors les sommets correspondants $Y(s_v)$ ne contiennent pas de solutions réalisables.

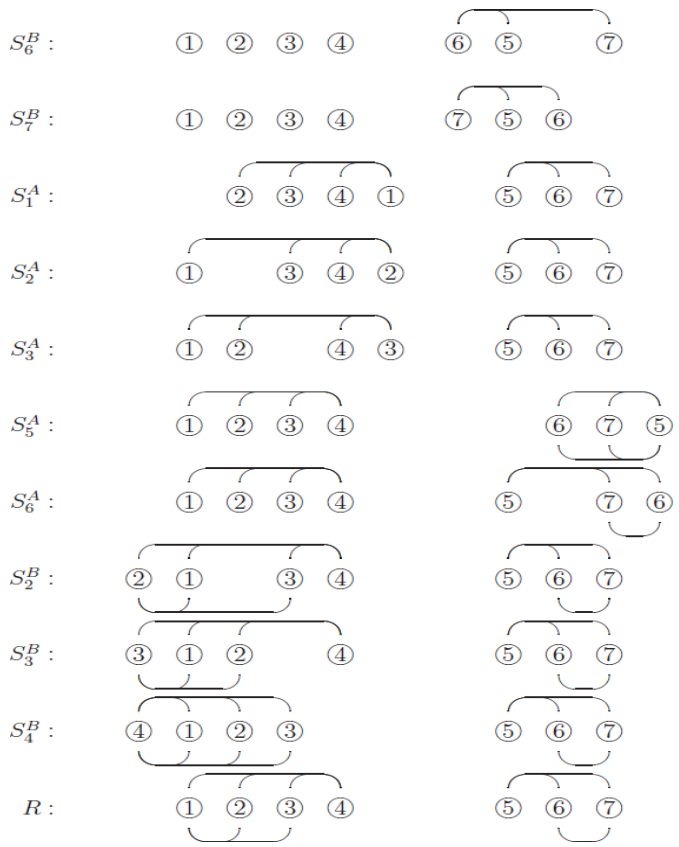


Figure 14. Exemple de séparation

Il est intéressant de vérifier, dans chaque sommet *r* de l'arbre de recherche, l'existence des circuits lors de la construction des *avant-candidats* et *après-candidats*. Ceci veut dire que pour un block $B_l : u_1^l, ..., u_{m_l}^l$ s'il existe un ars disjonctif (*i*,*j*) tel que $((i, j) \in B_l)$ déjà fixé dans le sommet actuel de l'arborescence, alors l'opération *j* (respectivement l'opération *i*) n'est pas insérée dans l'ensemble $E_l^B$ (respectivement $E_l^A$). De cette manière, les circuits trouvés dans l'exemple précédent peuvent être éliminés.

Aussi, une vérification plus rigoureuse d'existence de circuits peut être faite lors du processus de calcul des *heads* (têtes) et *tails* (queues) que nous détaillerons plus loin dans cette thèse.

Les ensembles de solutions développés avec cette technique de *séparation* sont deux à deux disjoints.

L'algorithme général de la méthode SEP en utilisant la *séparation* décrite ci-dessus, est le suivant :

*Procedure branch-and-bound(r)*
*Debut*

1. *Calculer une solution* $S \in Y(r)$ *en utilisant des heuristiques.*
2. *Si* $(C_{\max}(S) < UB)$ *alors* $UB := C_{\max}(S)$;
3. *Calculer un chemin critique P.*
4. *Calculer les blocks composants P.*
5. *calculer les sous-ensembles* $E_j^B$ *et* $E_j^A$.
6. *Tant que (il existe une opération* $i \in E_j^v \quad j = 1..k \text{ et } v = A, B$ *)*

   *Faire*
   *Supprimer i de* $E_j^v$ *;*

   *Fixer des arcs disjonctifs pour le sommet successeur s ;*
   *Calculer la borne inférieure du sommet S LB(S) ;*
   *Si (LB(S)<UB) alors Branch-and-bound(S);*

*Fin.*

Il est à noter que la stérilisation du sommet en cours *S* est effectuée si :

- *S* admet un circuit.
- Le chemin critique trouvé ne contient aucun block.
- Les deux ensembles $E_j^B$ *et* $E_j^A$ *sont vides pour tout block* $B_j$.

### 2.2 *Heads* (*Têtes*) et *tails* (*Queues*)

A chaque opération *i*, on associe une *tête* notée $r_i$ et une *queue* notée $q_i$. Ces deux notions sont indispensables pour le calcul des bornes inférieures.

- Une *tête* $r_i$ c'est la valeur qui exprime une borne inférieure pour le temps de début au plus tôt de l'opération *i*.
- Une *queue* $q_i$ c'est la valeur qui exprime une borne inférieure de la période de temps séparant le temps de fin de l'opération *i* et le temps de fin global optimal *Cmax*.

Le calcul des valeurs des *têtes* et des *queues* dépend principalement de tous les arcs conjonctifs et des ars disjonctifs fixés jusqu'à présent. Ainsi, ces valeurs changent d'un sommet *r* à un autre dans l'arborescence.

Une manière très simple pour le calcul des $r_i$ est de trouver la valeur du plus long chemin de la source jusqu'au sommet *i* dans le graphe $G = (X, C \cup F_r)$. Pour le calcul de $q_i$, il suffit de prendre le plus long chemin allant du sommet *i* jusqu'au sommet puits dans le graphe $G = (X, C \cup F_r)$.

Il existe d'autres méthodes permettant de calculer les valeurs des *têtes* et des *queues*. Ces méthodes sont présentées dans plusieurs références à savoir à titre d'exemple [15], [21], etc.

### 2.3 Sélection immédiate

Le but principal de la séparation présentée ci-dessus est de fixer, à chaque itération, le maximum possible d'arcs disjonctifs de l'ensemble $F_r$ en allant du sommet $r$ vers ses successeurs. Une fixation plus rapide des arcs disjonctifs permet d'améliorer et d'accélérer le processus de calcul de la méthode par séparation et évaluation.

Dans ce qui suit, nous présentons la méthode de Carlier et Pinson [19] qui permet de fixer des arcs disjonctifs additionnels en plus de ceux déjà fixés par la procédure de séparation. Cette méthode est appelée : *Sélection immédiate.* Elle est totalement indépendante de la procédure de séparation et elle utilise une borne supérieure du *makespan UB* et de simples bornes inférieures.

A partir de ce moment, nous supposons que $I$ est l'ensemble des opérations à traiter sur une machine donnée.

Soient : $n$ le nombre d'éléments de $I$, $J \subset I$ et $c \in I - J$.

Si la condition $\min_{j \in J \cup \{c\}} \{r_j\} + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} \{q_j\} \geq UB$ ....(2) est vérifiée alors toutes les opérations $j \in J$ doivent être ordonnées avant l'opération $c$, si on veut améliorer la borne supérieure $UB$. ss

Si $J$ se réduit à un seul élément, alors l'équation (2) peut être remplacée par une condition moins forte : $r_c + p_c + p_j + q_j \geq UB$..(3). L'équation (3) est équivalente à : $p_c + p_j > d_j - r_c$..(4).

Dans le cas où ces conditions sont vérifiées alors $j$ peut être ordonnée avant $c$. La variable $d_j$ peut être exprimée comme étant un délai d'exécution.

Comme conséquence directe aux résultats précédents, la procédure *select* suivante permet de fixer tous les arcs de l'ensemble $I$ de $n$ jobs en $O(n^2)$ :

*Procédure Select :*

1. *Pour c, j ∈I, c≠j Faire*
2. *Si $p_c+p_j>d_j-r_c$ alors*
3. *Fixer l'orientation de l'arc (j,c) de j vers c.*

*Fin-Procédure*

Dans ce qui suit, considérons le problème d'ordonnancement d'un ensemble *I* de *n* opérations sur une machine avec : des temps de traitement $p_k$, des dates de disponibilité $r_k$ et des délais $d_k$. Il est connu que ce problème avec préemption admet une solution optimale en un temps polynomial [19]. Cette solution peut être trouvée avec l'algorithme de Jackson pour un ordonnancement Préemptif (Jackson's Preemptive Schedule JPS) Carlier [19]. Le JPS calcule sa solution de Gauche vers la droite. Il existe aussi une version duale du JPS qui calcule la solution de la droite vers la gauche, appelée Backwards Jackson's Preemptive Schedule (BJPS).

L'algorithme de la *sélection immédiate* est utilisé avant le calcul d'une solution par le biais d'une heuristique, comme suit :

1- Calculer tous les arcs *primals* pour toutes les machines.
2- Calculer les nouvelles valeurs des *têtes* (*heads*) et des *délais*.
3- Calculer les arcs *duals* de toutes les machines.
4- Calculer les nouvelles valeurs des *têtes* (*heads*) et des *délais*.

Les étapes (1) et (3) permettent d'améliorer les valeurs des *têtes* (*heads*) et des *délais*, respectivement, en appliquant la procédure *select*. Le recalcul des nouvelles valeurs des *têtes* (*heads*) et des *délais* dans les étapes (2) et (4) est entrepris car l'ajout de nouveaux arcs en parallèle influence leurs valeurs. Les étapes (1) et (4) sont répétées tant qu'il y a des arcs disjonctifs à fixer.

### 2.4 Les bornes inférieures

Soit *r* un sommet de l'arborescence avec $F_r$ l'ensemble des arcs disjonctifs déjà fixés dans ce sommet. Une borne inférieure *LB*(*s*) peut être calculée pour tout sommet *s* successeur de *r* en se basant sur les arcs déjà fixés de l'ensemble $F_r$, les *têtes* $r_i$ et les *queues* $q_i$. Si la valeur de *LB*(*s*) pour un sommet donné *s* dépasse la valeur de la borne supérieure *UB*, alors il est inutile de continuer à explorer la branche descendante du sommet *s* (stérilisation).

Plusieurs méthodes permettant de calculer des bornes inférieures ont été testées dans la littérature. La meilleure méthode est celle donnée par Brucker [15]. Elle permet de calculer des bornes inférieures différentes tout au long du processus de séparation.

- La première borne Inférieure *LB* est calculée avec les ensembles $E_l^B$ et $E_l^A$. Si l'opération *i* doit être déplacée avant le block *B*, alors tous les arcs disjonctifs $(i, j) / j \in B - \{i\}$ doivent être fixés. La valeur de la borne *LB* est :

$$r_i + p_i + \max\left\{ \max_{j \in B-\{i\}} \{p_j + q_j\}; \sum_{j \in B-\{i\}} p_j + \min_{j \in B-\{i\}} \{q_j\} \right\}$$

- Par contre, si l'opération *i* est déplacée après le block *B,* alors cette borne deviendra :

$$\max\left\{ \max_{j \in B-\{i\}} \{r_j + p_j\}; \min_{j \in B-\{i\}} \{r_j\} + \sum_{j \in B-\{i\}} p_j \right\} + p_i + q_i$$

Il est clair que les valeurs retournées par les bornes inférieures *LB*(*s*) doivent être croissantes avec la fixation de nouveaux arcs disjonctifs. Donc, il est plus intéressant de calculer la *LB* à chaque fois qu'il y a un arc disjonctif fixé.

Il existe d'autres bornes inférieures, nous pouvons citer à titre d'exemple celle utilisée par Carlier & Pinson [19].

Dans cette section nous considérons l'ensemble *O* de toutes les opérations du problème, $O=\{o_1, o_2, \ldots, o_v\}$ avec *v* le nombre total des opérations de tous les jobs.

Soient :

- $E_k$ : Le sous-ensemble d'opérations qui doivent passer sur une même machine *k*,
- $r_i$ : La date de début au plutôt de l'opération *i* qui est la valeur de plus long chemin, du sommet *source* à l'opération $o_i$, $l(Source, o_i)$.
- $q_i$ : La valeur du plus long chemin de l'opération *i* au sommet *puits* $l(O_i, puits)$.

$$H(k) = \underset{i \in E_k}{Min}\{r_i\} + \sum_{i \in E_k} p_i + \underset{i \in E_k}{Min}\{q_i\}$$

**Proposition** [19] : *H(k)* est une borne inférieure des problèmes à one-machine.

**Proposition** [19] : Soit $V_k$ est la valeur optimale pour le problème à une machine associée avec la machine $M_k$.

$LB = Max\{V_k / k = 1, m\}$ est la borne inférieure pour le *makespan* du problème job shop. Il est prouvé que $LB = Max\{H(k) / E_k \subseteq O\}$ est peut être calculé en O(n *log n*).

### 3. Méthode exacte pour le Job Shop avec Blocage[1]

Les auteurs ont proposé peu de travaux en méthodes exactes afin de résoudre le problème de job shop avec la contrainte de blocage. La plupart de ces méthodes reprennent l'idée de la

[1] Les résultats de ce paragraphe sont publiés dans [3]

résolution des sous problèmes à une machine proposée par Carlier pour le JSS. Le problème majeur de ces méthodes est le fait de réutiliser les bornes inférieures déjà développées pour le JSS.

Dans ce qui suit, nous présentons la méthode exacte par séparation et évaluation (SEP) que nous avons développée pour résoudre le JSB.

### 3.1. Principe de la méthode

Par convenance, nous représentons l'exécution de la méthode SEP par une arborescence. La racine de l'arborescence représente l'ensemble de toutes les solutions du problème considéré. Pour appliquer la méthode SEP nous devons d'abord définir:

- une stratégie pour subdiviser l'espace de recherche afin de créer des espaces de recherche de plus en plus petits (Séparation).
- une fonction de calcul de la valeur d'une solution appelée évaluation. Cette fonction est généralement notée *Evaluer*().
- un moyen de calcul d'une borne supérieure et d'une borne inférieure.

La méthode commence par considérer le problème de départ avec son ensemble de solutions, appelé : *racine*. Des procédures de bornes inférieures et supérieures sont appliquées à la *racine*. Si ces deux bornes sont égales, alors une solution optimale est trouvée et l'exécution de la procédure est arrêtée. Sinon, l'ensemble des solutions est subdivisé en deux ou plusieurs sous-problèmes, devenant ainsi des enfants de la racine. La méthode est ensuite appliquée récursivement à ces sous-problèmes, engendrant ainsi une arborescence. Si une solution est trouvée pour un sous-problème, elle est réalisable mais pas nécessairement optimale pour le problème de départ. Elle peut être ensuite utilisée pour éliminer toute sa descendance : si la borne inférieure d'un nœud dépasse la valeur d'une solution déjà connue, alors on peut affirmer que la solution optimale globale ne peut être contenue dans le sous-ensemble des solutions représentées par ce nœud. La recherche continue jusqu'à ce que tous les nœuds sont soit explorés, soit éliminés.

L'algorithme général des méthodes par séparation et évaluation est décrit par l'algorithme suivant [13] :

*Algorithme général de SEP ;*
*Début*
*Evaluer (S) ; (* S étant la racine *)*
*Si S est une solution réalisable*
*Alors fin ;*
*Sinon*
*Insérer S dans F ; (* la file des sommets actifs *)*
*Meilleure solution courante = ∞ ;*
*Tant que F est non vide*
*Faire*
*Choisir S' de F ; (* exploration*)*
*F=F\{S'} ;*
*Séparer S' en $S'_1, S'_2, S'_3, ...., S'_k$.*
*Pour chacun des $S'_i$*
*Faire*
*Evaluer ($S'_i$) ;*
*Si v($S'_i$) < meilleure solution*
*Alors*
*Si $S'_i$ est une solution réalisable*
*Alors*
*Meilleure solution= v($S'_i$)*
*(* amélioration solution réalisable*)*
*Sinon*
*Insérer ($S'_i$) dans F*
*(* $S'_i$ est un sommet actif*)*
*Fsi*
*Sinon $S'_i$ sommet éliminé ;*
*(* $S'_i$ ne peut contenir la solution optimale *)*
*Fsi*
*Fait*
*Fait*
*Fsi*
*Fin.*

La fonction *v* dans l'algorithme ci-dessus est une fonction d'évaluation d'un nœud donné.

Pour résoudre le problème d'ordonnancement avec blocage, nous avons développé une méthode par séparation et évaluation qui utilise une procédure de séparation différente de celle présentée par Carlier en 1989 [19]. Elle utilise des notions de la théorie des graphes que nous décrivons dans la section suivante.

### 3.2 Quelques définitions

Lors de la conception de notre méthode SEP, nous avons utilisé plusieurs algorithmes de la théorie des graphes tels que : la mise à niveau d'un graphe, la recherche de plus long chemin dans un graphe sans circuit (Bellman), la détection de circuit dans un graphe (forte connexité) et les graphes réduits.

### 3.2.1 L'algorithme de Bellman (plus long chemin)

L'algorithme de Bellman permet de trouver les plus courts chemins reliant la source aux autres sommets d'un graphe pondéré sans circuits. Dans notre travail, nous avons utilisé cet algorithme pour le calcul des plus longs chemins dans un graphe sans circuits.

*Algorithme de Bellman (Plus long chemin)*

*(1) Soit un graphe orienté G=(X,U) avec un sommet source et un sommet puits.*
*(2)* H=∅
*(3) Soit p(x) La valeur du plus long chemin de la source vers x.*
*(4) Poser p(source)=0 et* H=H∪{source}.
*(5) Soit x un sommet quelconque et $y_i(x)$ l'ensemble de ses prédécesseurs.*
*(6) Calculer* $p(x) = {}^{Max}_{i}\{p(y_i) + d(y_i, x)\}$ tel que : $d(y_i, x)$ est la distance entre $y_i$ et *x.* et H=H∪$\{x\}$.
*(7) Si H=X alors arrêter l'algorithme Sinon aller à (5).*

*Fin-Algorithme*

Nous avons utilisé cet algorithme pour évaluer les nœuds dans l'arbre de recherche. Pour chaque nœud, l'algorithme de Bellman donne la valeur du plus long chemin $\boldsymbol{C_{max}}$ où les dates de début de chaque opération seront affectées dans cette étape.

En réalité, cette fonction d'évaluation est très importante. En effet, elle sert, grâce à la borne sup, à classer le nœud en cours de traitement dans l'arborescence comme un nœud actif ou passif. Ce qui permet de le stériliser. Cette stérilisation des sommets offre la possibilité d'éviter la recherche dans des branches non prometteuses.

### 3.2.2 L'algorithme de mise-à-niveau

L'algorithme de mise à niveau est un algorithme très utilisé avec les problèmes d'ordonnancement de projets sans contraintes dans lesquels la présence de circuits n'est pas permise. Ainsi, il permet de vérifier si le graphe, représentant ce projet, contient un circuit ou non.

L'algorithme de base de la mise à niveau est le suivant:

*Algorithme mise à niveau*
*(1) Soit un graphe orienté G=(X,U) et Z=∅ ;k=1.*
*(2) Etiqueter par k, tout sommet x, $x \in X$ tel que : $dg^{-}(x) = 0$ ;*
*Z=Z∪{x} ;*
*k=k+1 ;*
*(3) Supprimer tous les arcs sortants des sommets de l'ensemble Z.*
*(4) Allez à l'étape (1) ; On arrête la procédure lorsque Z=X.*
*Fin-Algorithme*

Dans la Figure 15, nous présentons le résultat de l'algorithme de mise à niveau appliqué au graphe de la Figure 15.a. Le résultat de la mise à niveau est présenté dans Figure 15.b.

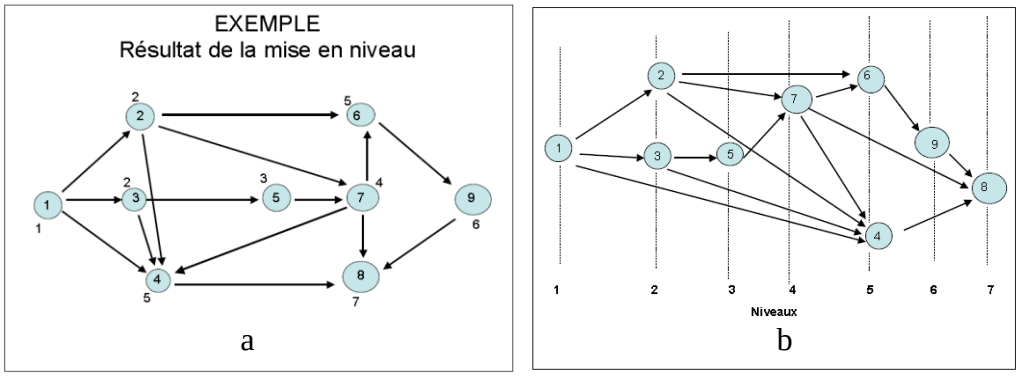


Figure 15. Exemple de mise à niveau d'un graphe sans circuits

Dans notre SEP, l'algorithme de mise-à-niveau sert à déterminer si le graphe contient des circuits ou non. Nous avons aussi utilisé cet algorithme pour rendre Bellman plus rapide, car il lui prépare les sommets par niveau. Il permet aussi de prendre la décision de continuer à appliquer l'algorithme de Bellman ou non (présence de circuits).

Dans un graphe G, si un sous-graphe de G ne peut être mis à niveau alors G contient un circuit.

La présence d'un circuit dans notre méthode a une valeur majeure et ceci apparaît dans deux cas :

- Si la longueur du circuit est nulle, alors prendre en considération le cas de swapping.
- Si la longueur du circuit est positive (solution non réalisable) alors le résultat permet de faire des stérilisations immédiates.

### 3.2.3. L'algorithme de recherche des composantes fortement connexes

Une composante fortement connexe d'un graphe $G$=(X,$U$) est un sous-graphe (maximal) $H$=($X_1$,$U_1$) tels que $X_1 \subseteq X$ et $U_1 \subseteq U$ et pour tous sommets $x$ et $y$ de $X_1$, il existe un chemin de $x$ vers $y$ et un autre chemin de $y$ vers $x$ dans $H$. Ce qui veut dire qu'une composante $f$-connexe admet toujours un circuit maximal qui passe par tous les sommets de cette composante.

L'algorithme de recherche des composantes fortement connexes est connu dans la littérature. Nous présentons dans ce sui suit une des variantes de cet algorithme:

*Algorithme Recherche Composantes f-connexes :*
Poser *G'=G*.

*(1) Choisir un sommet x de G' et le marqué par (+) et (-) en même temps;*
*(2) ∀ x marqué par (-), on marque tous les sommets $y \in \Gamma^{-}(x)$ par (-) ;*
*(3) ∀ x marqué par (+), on marque tous les sommets $y \in \Gamma^{+}(x)$ par (+) ;*
*(4) On arrête lorsqu'on ne peut plus marquer. L'ensemble des sommets marqués par (+) et (-) en même temps forme une composante fortement connexe $C_i$.*
*(5) G'= {G\ les sommets marqués par (+) et (-) en même temps lors de l'étape précédente}.*
*(6) Si G' <> Ø alors aller à l'étape (1) Sinon arrêter l'algorithme.*

*Fin-Algorithme*

Cet algorithme donne toutes les composantes fortement connexes d'un graphe quelconque. Par conséquent, il va permettre de déterminer les circuits composant un graphe *G*.

Dans notre méthode SEP, l'algorithme de recherche des composantes *f*-connexe est lancé avec l'algorithme de mise à niveau, i.e. le processus de recherche de circuits dans un graphe n'est lancé que si l'algorithme de mise à niveau est bloqué. Ce qui veut dire qu'il vient de trouver un circuit. Cette situation est débloquée par l'algorithme de forte connexité combiné avec les graphes réduits.

### 3.2.4. Graphe réduit

La réduction d'un graphe consiste à créer un nouveau graphe $\overline{G} = (\overline{X}, \overline{U})$ à partir du graphe initial *G*=(*X*,*U*) telle que toute composante fortement connexe dans *G* est réduite en un seul sommet $\bar{x}$ dans $\overline{G}$. De plus, tous les arcs de *G* appartenant à la même composante fortement connexe ne sont pas représentés dans $\overline{G}$ et les arcs incident intérieurement ou extérieurement à cette composante, resteront incident intérieurement ou extérieurement au sommet $\bar{x}$ dans $\overline{G}$. Dans le cas où plusieurs arcs sortent du sommet $\bar{x}$ vers un même quelconque, on ne garde que l'arc qui à le poids maximum.

Bellman a proposé un algorithme ayant une complexité polynomiale qui opère sur les graphes sans circuits. La réduction de graphe, grâce aux composantes fortement connexe, permet de résoudre le problème de présence de circuits de longueur nulle.

### 3.3 Description de l'algorithme séparation et évaluation

Dans cette section, nous présentons la mise en œuvre de notre méthode exacte SEP. C'est un algorithme de création et de parcours d'un arbre n-aires «arbre de recherche». La création de l'arbre est assurée par le processus de séparation alors que le parcours est assuré à chaque évaluation.

Dans la littérature, il existe plusieurs manières de parcourir l'arborescence: le parcours en largeur, le parcours en profondeur, le meilleur d'abord, etc. Des heuristiques sont même spécialement conçues pour la sélection d'un nœud parmi les autres**.**

### 3.3.1 Séparation

Le point de départ est un nœud qui représente le problème initial, avec les contraintes alternatives relaxées. Nous avons évité de porter dans ce nœud toute information qui ne change pas pendant la séparation (les arcs conjonctifs, les données, les pondérations des arcs, etc.) pour un souci d'optimisation de l'espace mémoire ; d'où d'idée de faire la différence entre les données comme suit:

- Le problème initial, qui contient toutes les données qui ne changent pas.
- Les sous-problèmes, contenant les arcs alternatifs représentant les contraintes sur lesquelles le processus de séparation va se confronter. Ce sont des données dynamiques qui changent avec le parcours de l'arbre.

L'idée de séparation que nous avons utilisée se base sur l'énumération implicite de toutes les combinaisons possibles que peut engendrer l'ordonnancement d'un ensemble d'opérations concurrentes sur une même machine.

La séparation sur l'ordre entre les opérations concurrentes sur une même machine permet de fixer un ensemble d'arcs alternatifs en même temps, contrairement à la séparation sur les arcs alternatifs qui à chaque itération permet de fixer un seul arc alternatif, donnant ainsi un espace de recherche plus important. D'ailleurs, la séparation sur les arcs alternatifs peut engendrer des cas où la solution résultante est non réalisable. Les ordonnancements ainsi générés comportent plusieurs situations de violation de contraintes de précédence. Pour cela, une vérification de l'admissibilité de la solution doit se faire après chaque séparation.

Dans les problèmes de job shop simple, le circuit est un indicateur suffisant pour dire que la solution est non réalisable. Par contre, dans notre cas, nous faisons la différence entre un circuit de longueur nulle et un circuit de longueur positive.

**Exemple :**

Soit un problème de Job-Shop à 3 machines et 2 jobs $J_1$ et $J_2$ (Figure 16) pour lequel nous cherchons un ordonnancement de durée minimale. Rappelons que dans ce type de problème, l'ordre de passage des opérations d'un même job est fixe. Résoudre ce problème revient à déterminer l'ordre de passage des opérations concurrentes sur les machines, i.e.: à déterminer

sur chaque machine, quelle opération est avant l'autre. Nous pouvons résoudre ce problème par une recherche arborescente dont l'arbre est présenté par la Figure 16. La solution optimale de ce problème est d'une durée égale à 13 (le sommet dont la valeur est entourée sur la Figure 16).

| Les jobs | La séquence des machines | Les durées sur les machines |
|---|---|---|
| $J_1$ | $M_2$, $M_1$, $M_3$ | 5,2,3 |
| $J_2$ | $M_3$, $M_2$, $M_1$ | 6,3,4 |

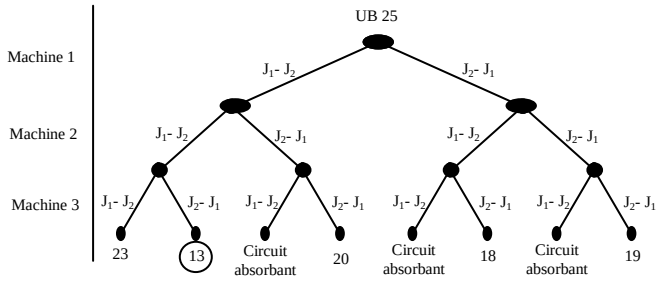


Figure 16. Arbre de recherche pour un job shop trois machines et deux jobs

La Figure 16 montre toutes les combinaisons possibles pour avoir l'ordre de passage des jobs sur les machines. La notation ($j_i$–$j_j$) s'interprète par : le job *i* avant le job *j* sur cette machine. Donc, elle donne la chance pour chaque job d'être le premier choisi à condition qu'il demande cette machine dans sa gamme de fabrication. La fixation du sens de l'arc alternatif se fait dans cette étape où un ensemble d'arcs alternatifs sera fixé à la fois représentant l'ordre choisi par la procédure de séparation.

### 3.3.2 Évaluation

La procédure d'évaluation que nous proposons utilise l'algorithme de Bellman qui va donner à chaque opération sa date de début sur la machine qu'elle demande. La date de fin de la dernière opération du graphe représente la valeur de la fonction Objectif $C_{max}$. Cette procédure, qui utilise l'algorithme de Bellman, est combinée avec l'algorithme de mise à niveau et aussi avec l'algorithme de réduction des graphes basé sur l'algorithme de recherche des composantes fortement connexes. Ce dernier algorithme n'est appelé que si l'algorithme de mise à niveau est bloqué, ce qui veut dire que le graphe admet un circuit. Nous avons aussi utilisé une autre procédure d'évaluation qui est basée sur l'algorithme de recherche de plus long chemin de Ford.

### 3.3.3 Stérilisation

La stérilisation d'un sommet se fait dans deux situations :

- Si le graphe contient un circuit positif, ce cas représente une solution non réalisable d'où il est inutile d'aller plus loin dans cette branche de l'arbre.
- Si l'évaluation d'un nœud de l'arborescence donne une valeur de *Cmax* supérieure ou égale à celle de la meilleure solution trouvée jusqu'à présent.

### 3.3.4 Bornes inférieures

Pour les bornes inférieures, il n'existe pas dans la littérature de bornes inférieures spécialement développées pour le job shop avec blocage. Donc, tous les travaux se sont orientés vers les bornes déjà utilisées pour le problème de job shop classique que nous avons détaillé dans la section 1 de ce chapitre.

Dans notre cas, nous avons testé la borne inférieure donnée par Carlier et l'évaluation réelle du nœud en cours d'exploration en utilisant la procédure d'évaluation proposée qui est basée sur l'algorithme de Bellman. Elle a donné de meilleurs résultats.

## 4. Parallélisation de la méthode par Séparation et Evaluation[2]

Le développement de la microinformatique et l'efficacité des techniques de communication entre les ordinateurs par le biais des réseaux informatiques ont permis le développement des méthodes parallèles de résolution des problèmes d'optimisation combinatoires. Ceci est fait sans avoir recours à des machines parallèles multiprocesseurs hautement sophistiquées avec des systèmes distribués compliqués. Tout ce qui est nécessaire, c'est un réseau informatique avec une bonne architecture parallèle.

L'application de la méthode SEP sur le modèle du job shop simple a permis de constater son incapacité à traiter des problèmes de taille plus de (10×10) par un seul processeur, en plus des limites observées dans l'espace mémoire et dans le temps de réponse. Ces raisons ont rendu la réalisation d'un traitement de calcul parallèle de cette méthode une nécessité plus qu'une option. Comme buts de cette parallélisation, nous avons ciblé:

- L'amélioration du temps de réponse.
- La résolution des problèmes de taille moyenne au moins à cause de l'espace de recherche qui évolue d'une manière exponentielle en raison de la complexité du

[2] Les résultats de ce paragraphe sont publiés dans [2]

problème. Ce qui rend le parcours de tout cet espace par la méthode exacte SEP un travail très délicat.

- La réalisation d'un algorithme qui remplace le travail d'un processeur puissant, assurant un envoi d'information très rapide entre les postes du réseau.

Dans la suite de cette partie, nous allons présenter une description détaillée de notre algorithme SEP parallèle. Cette description se fera comme suit :

Dans un premier temps, nous illustrons le principe de l'algorithme SEP parallèle. Puis, nous détaillons la plate-forme matérielle utilisée ainsi que les structures de données avancées que nous avons intégrées pour la réalisation de cet algorithme.

### 4.1 Principe de l'Algorithme SEP Parallèle

La méthode SEP parallèle que nous avons développée tourne sur un réseau d'ordinateurs où chaque poste à un rôle bien précis à réaliser. La communication entre ces postes est assurée par des requêtes. Le principe de base de notre algorithme parallèle utilise un poste du réseau appelé : *contrôleur*. Son rôle est:

- Le chargement des données de base du problème posé.
- L'affectation des tâches aux autres postes qui vont réaliser la recherche de la solution optimale du problème posé.
- L'écoute de toute nouvelle amélioration et sa diffusion vers tous les autres postes.
- L'écoute et la déclaration de la fin de l'algorithme.

Concernant les autres postes du réseau (autre que le *contrôleur*), ils sont dédiés aux opérations de calcul. Ces opérations englobent l'exploration et l'évaluation des différents nœuds de l'arborescence. Le rôle principal des ces postes est d'améliorer les valeurs des solutions trouvées et d'informer le poste *contrôleur* de chaque amélioration jusqu'à trouver la solution optimale.

Les demandes de travail émises par les postes calculateurs se font par un passage séquentiel sur leurs voisins (mais pas le *contrôleur*), le but étant de distribuer toutes les branches de l'arborescence de recherche sur tous les postes du réseau d'une manière équitable.

### 4.2 La topologie physique et la topologie logique

La définition ou le choix d'une topologie physique qui s'adapte à notre algorithme parallèle mène à poser la question suivante: *existe-il une topologie physique favorable à ce*

*travail de calcul parallèle sachant que la contrainte de transfert d'informations en temps réel existe entre les postes calculateurs*?

En réponse à cette question, nous pensons que la topologie dominante dans ce cas de figure est la topologie complète ou étoile citée plus loin. En d'autres termes, nous imaginons un réseau d'ordinateurs où chaque poste est connecté à un autre par un câble de connexion indépendant. Dans ce cas, chaque poste peut communiquer directement avec tous les autres postes du réseau sans aucun problème. Par contre, cet avantage peut devenir facilement un inconvénient. En effet, celui-ci peut apparaître dans la surcharge du réseau à cause d'un flux d'informations important échangé entre tous les postes et dans tous les sens.

Une autre topologie, qui parait aussi intéressante, c'est la topologie anneau. Dans cette topologie, l'échange d'informations directement en point à point n'existe pas, donc, pas de surcharge du réseau. Cependant, elle a un inconvénient majeur qui est reflété par un taux de rendement très bas. Ceci est dû au fait que la communication entre deux postes nécessite le passage sur tout les postes qui précèdent le poste destination.

Pour bénéficier des avantages des deux topologies: étoile et anneau et à fin de pallier à leurs inconvénients, nous avons proposé une nouvelle topologie que nous avons appelée : *topologie mixte*. Cette nouvelle topologie est caractérisée par la création d'une structure d'anneau logique à l'intérieure d'une topologie étoile. Donc physiquement en termes de matériel, nous disposons de tout ce qu'il faut pour faire fonctionner notre algorithme parallèle sur une topologie étoile. Mais, nous avons ajouté certaines restrictions logiques qui sont caractérisées par l'interdiction d'effectuer des communications directes entre les différents postes. Le principe est de toujours passer par l'anneau logique sauf dans certains cas particuliers qui nous permettent d'améliorer encore plus la qualité de transfert d'informations sur le réseau. Avec cette nouvelle structure, tous les types de communications possibles entre les postes du réseau sont permis. Grâce à la topologie étoile utilisée, l'algorithme peut effectuer un transfert d'informations sur tout le réseau en parallèle ou en point à point. Autrement, les informations circulent dans un seul sens avec la structure de l'anneau logique créée au sein de la topologie étoile.

La Figure 17 montre le schéma de la topologie étoile utilisée dans laquelle nous trouvons les différents postes (contrôleur et calculateurs) qui sont reliés par un Switch.

Dans cette structure, le nombre de poste calculateurs est théoriquement illimité. Mais dans la pratique, on peut mettre le nombre supporté par le système d'exploitation vis-à-vis du nombre des threads permis.

La structure de notre algorithme permet l'augmentation du nombre de postes calculateurs sans diminuer la rapidité du calcul; i.e. : elle n'alourdit pas le travail du contrôleur, au contraire elle n'a aucun effet sur ce dernier, à l'inverse des autres technologies (maitre/esclave, client/serveur). Nous avons même profité du temps d'attente du contrôleur pour le dédier à la recherche d'une amélioration de la borne supérieure *UB* par une méthode approchée. Dans notre cas, nous avons utilisé le Recuit Simulé.

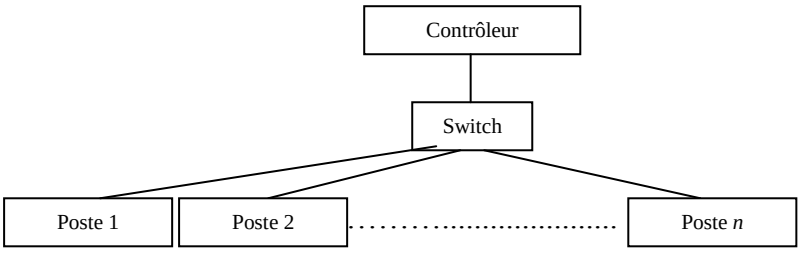


Figure 17. Topologie étoile

La Figure 18 montre la topologie anneau logique que nous avons proposée. Elle englobe tous les postes calculateurs du réseau, mais pas le contrôleur. La création de cet anneau logique est faite par notre algorithme parallèle directement au lancement, à partir du contrôleur où il envoie à chaque calculateur (poste) la structure suivante :

- Son adresse IP (adresse du contrôleur).
- Les adresses IP de ses voisins gauche et droit simultanément.
- L'information si ce poste est un lanceur ou pas.

Le contrôleur choisi au hasard (ou par une technique de sélection) un poste calculateur pour le nommer : lanceur initial du calcul. Nous avons appelé ce poste : le *lanceur*. Le contrôleur envoie les données du problème initial au *lanceur* et il lance en parallèle cinq processus :

- Le Recuit Simulé.
- L'écoute des demandes du problème initial.
- L'écoute de nouvelles améliorations.
- L'écoute de blocage du jeton.
- L'écoute de la terminaison de l'algorithme.

Après avoir reçu la structure du problème initial à partir du *contrôleur*, le *lanceur* doit créer et lancer un *jeton* dans le réseau indiquant par ça le commencement du travail. A la réception du *jeton*, chaque calculateur demande le problème initial à partir du contrôleur.

Le calculateur disposant du *jeton*, après la réception du problème initial, envoie une demande de travail au voisin droit, tout en gardant le *jeton* à son niveau. Ceci est fait pour assurer qu'un seul poste qui demande à la fois. Deux cas sont possibles :

- Si le voisin droit à un travail supplémentaire (la file des sous-problèmes contient plus qu'un sous problème). Dans ce cas, il envoie directement la moitié de cette file au demandeur grâce à son adresse IP qui apparaît dans la demande. A la réception du travail par le demandeur, il libère le *jeton* à son voisin gauche lui donnant par là, l'occasion de demander le travail.
- Dans le cas contraire, il passe cette demande à son voisin droit et la demande tourne jusqu'à l'obtention du travail ou la fin de l'algorithme.

Parallèlement au processus de séparation effectué par les postes calculateurs, ces derniers lancent les processus de gestion du *jeton* et des demandes de travail.

Nous remarquons concernant la demande de travail qu'une seule demande tourne à la fois dans l'anneau. Ceci permet les gains suivants :

1. Une gestion simple, efficace et surtout équitable du travail réalisé par les calculateurs.
2. Une nouvelle manière de recherche car avant chaque partage de la file, le poste donneur (émetteur) a réalisé un parcours assez profond dans l'arbre de recherche (en profondeur) d'où un gain très important en temps de réponse.
3. Minimisation très importante du trafic sur le réseau en comparant ceci au techniques classiques (maitre/esclave...).
4. Soulagement clair du contrôleur car l'augmentation des calculateurs n'a aucun effet sur ce dernier.
5. Autres gains à citer par exemple : la détection de la terminaison de l'algorithme, etc.

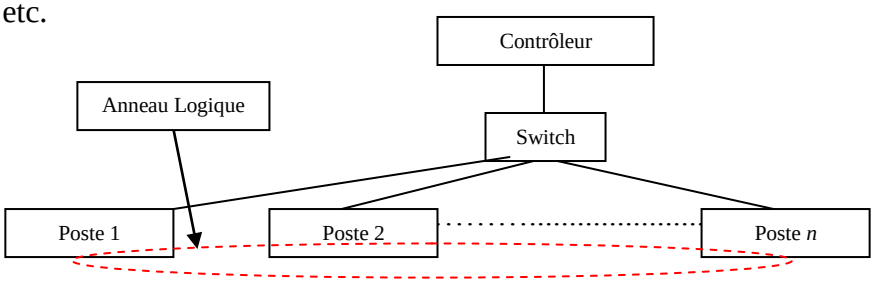


Figure18. Topologie Anneau Logique

### 4.3 Les acteurs du système

Nous avons déjà présenté dans les points précédents les deux acteurs principaux existant dans notre algorithme : le *contrôleur* et le *calculateur*.

#### 4.3.1. Le contrôleur

Cet acteur réalise les tâches d'un gérant dans le sens positif du mot (contrairement à un maître). Il est présent pour assurer une bonne gestion du processus qu'est la détermination d'une solution optimale au problème de job shop.

Une bonne gestion du système, c'est le fait de dédier les postes calculateurs uniquement pour le calcul.

En plus, l'interface utilisateur de l'application est indépendante du processus de calcul lors de l'exécution du programme (le moteur SEP).

#### 4.3.2. Le calculateur

Dans notre algorithme, le rôle principal du calculateur est la réalisation de la procédure de séparation et évaluation détaillée dans la section ci-dessus.

### 4.4. L'utilité du jeton dans l'algorithme

Dans la technologie classique de l'anneau par *jeton* (token ring), le *jeton* a été utilisé comme privilège d'accès à l'anneau. Le poste qui reçoit le jeton a le droit d'envoyer les données, sinon il rend le jeton dans l'anneau en l'envoyant à son voisin gauche ou droit selon le protocole de communication utilisé.

Dans notre travail, nous proposons d'inverser les règles c'est-à-dire celui qui reçoit le jeton a le droit de demander le travail (et non pas d'envoyer) s'il est en état de repos. Le *jeton* sera créé une et une seule fois par le lanceur. Donc, la possibilité d'avoir plusieurs jetons qui circulent dans le réseau est éliminée. Aussi, lorsqu'un poste calculateur veut demander du travail, il est obligé de garder le *jeton* à son niveau, ce qui rend la circulation de plusieurs demandes dans le réseau impossible. Ceci évite toute incohérence dans la gestion du jeton ou de la demande de travail.

Un champ (*repos*) désigne si le processus de séparation est à l'état de repos lorsqu'il porte la valeur « *vrai* ». Cette variable à la création de la demande sera mise à « *vrai* » pour dire que tous les postes sont au repos. Seuls les processus de séparation actifs ont le droit de mettre cette variable à « *faux* », pour exprimer le fait que le poste soit occupé et au moins un

processus est entrain de s'exécuter. Quand le demandeur reçoit sa demande, il vérifie l'état de cette variable si elle n'a pas changé. Il déclare la terminaison de l'algorithme au niveau du contrôleur pour que ce dernier arrête le traitement. Cependant, s'il reçoit un traitement comme réponse à sa demande, il mettra son adresse IP dans le champ (*bénéficiaire*) de l'objet *jeton* et il l'envoie à son voisin gauche.

A la nouvelle réception du *jeton*, le bénéficiaire doit s'assurer que le champ (*bénéficiaire*) contient son adresse IP. Si c'est le cas, il envoie le jeton au *contrôleur* pour le bloquer car cette situation veut dire que tout le monde travaille et la circulation inutile du *jeton* va alourdir le temps de réponse. Le *contrôleur* à son tour informe tous les calculateurs que le jeton est bloqué.

Lorsqu'un poste calculateur termine son processus de séparation, il envoie la demande de déblocage au contrôleur. Ce dernier compare le niveau actuel de blocage au niveau demandeur. Si la requête porte un niveau inférieur, il ne répondra pas à cette requête et le *jeton* va tôt ou tard arriver au demandeur. Par contre s'il y a une égalité des niveaux, le contrôleur envoie le *jeton* au demandeur et incrémente son niveau de blocage pour dire que toutes les requêtes inférieures à ce niveau ont été satisfaites.

## 5. Conclusion

Nous avons présenté dans ce chapitre une nouvelle méthode SEP pour le JSB. Celle-ci utilise les techniques de bases de la théorie des graphes, à savoir : l'algorithme de recherche du plus long chemin de Bellman, l'algorithme de mise à niveau, l'algorithme de recherche des composantes fortement connexes et enfin l'algorithme de réduction des graphes. Ensuite, nous avons développé une technique de parallélisation très efficace de cette méthode. Dans le chapitre suivant, nous allons étudier l'impact de deux méta-heuristiques, à savoir, les Algorithmes Génétiques *AG* et l'Optimisation par les Essaims de Particules *PSO* sur le problème de job shop avec blocage. De plus, nous proposons une nouvelle idée de codage des solutions, ainsi, qu'une technique de parallélisation pour les deux méthodes.

# Chapitre 5
# Résolution approchée

## 1. Introduction

L'implémentation pratique des méthodes de résolution exactes du problème du job shop avec blocage, a montré leur limite à résoudre les problèmes de grandes tailles. Ceci est dû à la complexité du problème déjà détaillée dans les premiers chapitres de cette thèse.

L'importance de ce problème et sa présence dans beaucoup de situations concrètes ont incité les chercheurs à orienter leurs travaux vers les heuristiques. Ces méthodes sont des techniques généralement basées sur le bon sens qui oriente vers de bonnes solutions. En effet, des chercheurs ont travaillé sur ce problème en utilisant les métaheuristiques connues dans la littérature comme, les Algorithmes Génétiques, l'Optimisation par les Essaims de Particules, les Colonies de Fourmies, la Recherche Taboue, le Recuit Simulé, etc. Au vu des résultats encourageants obtenus dans ce domaine, les chercheurs ont centré leurs efforts sur l'étude de ce type de méthodes de résolution.

De même, nous nous sommes intéressés au développement de ce volet par l'exploration de certaines métaheuristiques et l'étude de leurs impacts sur la résolution du problème de job shop avec blocage (JSB).

Dans ce chapitre, nous décrivons en détail l'application de deux métaheuristiques connues dans la littérature appliquées au JSB. Ces deux méthodes sont les Algorithmes Génétiques (AG) et l'Optimisation par les Essaims de Particules (PSO). Pour l'implémentation des deux méthodes, nous avons orienté nos efforts vers deux axes principaux : le premier axe consiste en l'exploration profonde des différentes possibilités de codages des solutions du problème du JSB, qui est en fait la clé du problème ; le deuxième axe consiste en la parallélisation de ces méthodes. Le but de cette dernière n'est pas le gain en temps d'exécution car les métaheuristiques sont connues pour leur rapidité à atteindre des solutions réalisables de bonne qualité, mais plutôt de pouvoir explorer le maximum de solutions réalisables possibles afin d'améliorer l'efficacité de ces méthodes.

La complexité de notre problème et l'ampleur de la taille de l'ensemble des solutions réalisables de ce problème, imposent le choix d'un codage efficace des solutions qui doit éviter l'exploration des solutions non réalisables. C'est pour cette raison que nous avons

concentré nos efforts sur les différentes techniques de codage des solutions pour les deux métaheuristiques (AG et PSO).

En se basant sur la modélisation par les graphes alternatifs, nous avons développé notre premier codage qui est un code binaire. Dans ce dernier, nous avons représenté uniquement les arcs alternatifs ; son implémentation a donné des résultats satisfaisants. Cependant, lors du processus de résolution par les deux métaheuristiques AG et PSO, le nombre de solutions non réalisables explorées, en utilisant ce premier codage, reste important. Ce qui explique la lenteur de ces algorithmes pour certains exemples.

Pour pallier à cet inconvénient, nous avons adapté deux autres codages. Le premier codage est basé sur les règles de priorité [25], pour lequel nous avons proposé un algorithme très efficace de décodage de solutions. Ce codage a donné de meilleurs résultats surtout avec la version parallèle. Le deuxième codage adapté est le codage par rang [32] que nous détaillons dans la section 2.3. L'inconvénient de ce codage est qu'il nécessite un algorithme de correction à chaque génération de nouvelles solutions.

## 2. Le codage des solutions[3]

Le codage d'une solution est un déterminant important de l'efficacité d'une méthode d'optimisation. C'est une chaîne de bits contenant toute l'information nécessaire à la description d'un point dans l'espace d'état d'un problème donné. Son but est la manipulation d'opérateurs simples qui permettront d'explorer l'espace des solutions d'une manière efficace.

Dans notre cas, un codage d'une solution signifie la transcription d'un ordonnancement réel en représentation adéquate permettant la réalisation des différents opérateurs de la métaheuristique en question.

Dans ce qui suit, nous présentons trois stratégies de codage qui sont : le codage basé sur les arcs alternatifs, le codage basé sur les règles de priorité et le codage par rang. Ces trois stratégies sont communes aux deux métaheuristiques AG et PSO.

### 2.1 Le codage basé sur les arcs alternatifs

Le codage binaire s'applique sur la modélisation par les graphes alternatifs. Dans ce cas, une orientation des arcs alternatifs définit un ordonnancement. Nous avons utilisé cette caractéristique pour développer le codage binaire suivant : une solution est codée par un

[3] Les résultats de cette partie sont publiés dans [2] et [3]

vecteur binaire *Ch* comportant *n* gènes où chaque gène représente une paire d'arcs alternatifs du graphe, *n* étant le nombre total d'arcs alternatifs.

Nous affectons au gène la valeur 0 ou 1, comme suit :

- Le gène reçoit la valeur 0 pour désigner le sens positif de l'arc alternatif.
- Le gène reçoit la valeur 1 pour désigner le sens négatif de l'arc alternatif.

Le parcours de tous les gènes permet de fixer les arcs alternatifs nécessaires sur le graphe $G=(X,C\cup A)$. L'évaluation d'une telle solution peut se faire par l'algorithme de recherche de plus long chemin de Ford ou bien par la même méthode d'évaluation basée sur l'algorithme de Bellman (combiné avec les algorithmes de base de la théorie des graphes), décrit dans le chapitre précédent. Ce codage est très approprié pour les deux métaheuristiques AG et PSO.

**Exemple**

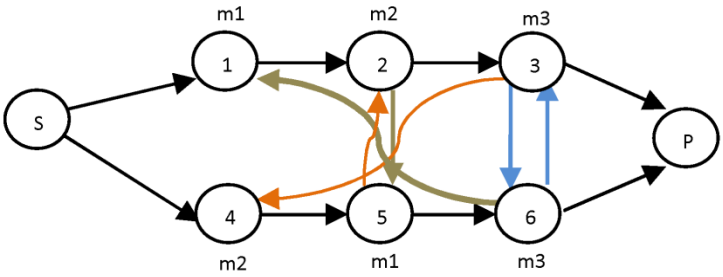


Figure 19. Exemple de graphe comportant trois paires d'arcs alternatifs.

Les arcs conjonctifs décrivant un ordre de passage pour les opérations d'un même job sont présentés dans le graphe ci-dessus avec une couleur noire. Par contre, les autres arcs alternatifs sont colorés. Chaque paire d'arcs alternatifs est colorée par une couleur distincte.

Soit *ch* le codage binaire d'une solution de l'exemple ci-dessus. *Ch*=[1,0,1]. Dans le Tableau 2 suivant, nous donnons les arcs alternatifs choisis suivant le codage *ch*.

| *Paires* | *Arcs* | *Ch* | *Arc sélectionné* |
|---|---|---|---|
| Paire 1 (rouge) | Arc (3,4) | 1 | Arc (5,2) |
| | Arc (5,2) | | |
| Paire 2 (vert) | Arc (6,1) | 0 | Arc (6,1) |
| | Arc (2,5) | | |
| Paire 3 (bleu) | Arc (3,6) | 1 | Arc (6,3) |
| | Arc (6,3) | | |

Tableau 2. Arcs alternatifs sélectionnés

Ce codage engendre la solution suivante :

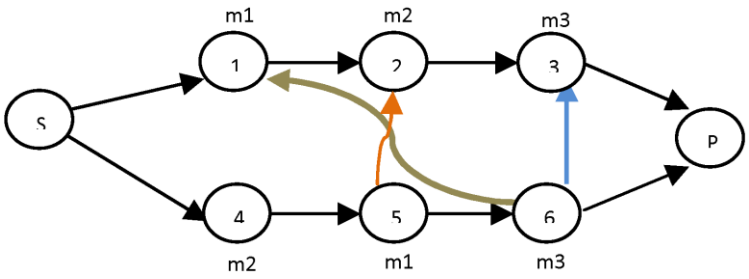


Figure 20. Solution par graphe alternatif du codage *Ch*

L'inconvénient de ce codage, lors de l'application des opérateurs de génération des solutions, est qu'il génère souvent des solutions non réalisables ce qui engendre, à un moment donné, des populations de solutions non réalisables. Ce qui contribue à la lenteur de la méthode pour converger vers la solution optimale. Pour pallier cet inconvénient, nous avons adapté un deuxième codage basé sur les règles de priorité.

### 2.2 Le codage basé sur les règles de priorité

Les règles de priorité sont une sorte d'heuristique qui tente de guider le processus de recherche de solutions. Dans ce codage utilisé d'abord par Dorndorf et Pesch [25], le chromosome est une suite de gènes qui représentent des règles de priorité et non pas la solution. La construction de l'ordonnancement dans notre approche à partir d'un chromosome ainsi codé, se fait selon l'algorithme de décodage qui est une méthode constructive et dont la décision sur la prochaine opération à chaque étape est guidée par les gènes du chromosome.

Les règles de priorités se classent en deux familles :

- Les règles statiques qui ne dépendent que des données du job (Durée, ordre de précédence, etc.). Parmi les règles statiques citons : les règles SPT (Shortest processing time), LPT (Longest Processing Time) et SWPT (Shortest weighted processing time).
- Les règles dynamiques tiennent compte de l'ordonnancement construit jusqu'à présent. Parmi les règles dynamiques citons : les règles MOPNR (Most Operations Remaining ou MTR Most Task Remaining) et MWKR (Most Work Remaining ou LRT Longest remaining processing Time).

Contrairement au codage binaire basé sur les arcs alternatifs, les méthodes constructives se distinguent par leur rapidité et leur grande simplicité. Nous obtenons en effet très rapidement

une solution admissible pour un problème donné sans avoir recours à d'autres techniques de correction.

Les règles de priorité entre les opérations à traiter sont nombreuses. Nous avons utilisé dans notre application dix règles qui sont largement employés.

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

Tableau 3. Différentes règles de priorité utilisées

Dans ce codage, la longueur du chromosome est égale au nombre total d'opérations. Par exemple, le chromosome suivant : *Ch*= [LPT, LFT, EFT, SPT, LST, EST, Random, LWKR], représente un ordonnancement pour un problème de huit opérations.

La partie la plus importante de ce codage est l'algorithme de décodage que nous avons développé. Le but de cet algorithme, présenté ci-dessous, est le passage du codage avec les règles de priorités vers l'ordonnancement physique.

*Algorithme décodage*
*Entrée : Un chromosome ch=*[$R_1$,..., $R_r$] tel que*: $R_i$ : la $i^{ème}$ règle de priorité;*
*Ensemble d'opérations prêtes C;*
*Sortie : un ordonnancement actif ;*
*Initialiser C par l'ensemble d'opérations prêtes ;*
*Pour (i :=0 à Nb_Opérations) Faire*
*Début*
*1. Choisir selon ch[i] une opération de C; /* l'opération à ordonnancer */*
*2. Déterminer la machine correspondante;*
*3. Si (la machine est libre) alors Allouer;*
*Sinon Bloquer ;*
*4. Mettre à jour l'ensemble des opérations prêtes C;*
*Fin ;*
*Ordonnancer la dernière opération ;*
*Fin-Algorithme.*

L'avantage primordial de ce codage est le fait qu'il ne génère que des solutions réalisables. Ce qui permet une exploration efficace de l'espace des solutions.

### 2.3 Codage par rang

Le codage par rang est une représentation efficace du problème d'ordonnancement de type job shop. En effet, chaque opération $o_i$ est représentée avec deux paramètres très importants : le premier paramètre est le numéro de la machine *k* sur laquelle elle doit être traitée et le second est l'ordre de son passage sur cette machine. Bien sur, cet ordre est plus important dans le cas où il existe plusieurs opérations concurrentes sur une même machine *k*.

La structure du codage est présentée dans le Tableau 4, qui reprend l'exemple de la Figure 20 présenté ci-dessus.

| N° Opération | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| N° Machine | 1 | 2 | 3 | 2 | 1 | 3 |
| Rang | 2 | 2 | 2 | 1 | 1 | 1 |

Tableau 4. Exemple du codage par rang

Les inconvénients majeurs de ce codage sont :

- Il s'adapte mal pour la métaheuristique PSO, par contre, il s'adapte très bien avec les AG.
- L'application du croisement dans les AG ne donne pas automatiquement des solutions réalisables par rapport au rang des opérations sur les mêmes machines. Une manière de régler ce problème serait de corriger les erreurs du rang d'une manière aléatoire en gardant l'ordre du tri sur la même machine *k* dans les chromosomes fils.

## 3. Algorithme Génétique proposé[4]

Les AG's sont connus dans la littérature pour être largement utilisés dans la résolution des problèmes d'optimisation. Ils sont considérés par plusieurs chercheurs comme des méthodes bien adaptées au problème de Job Shop, même s'ils ne garantissent pas l'optimum dans plusieurs cas difficiles. Dans cette section, nous allons décrire l'AG que nous avons développé. Son principe de base utilise l'algorithme génétique standard développé par Goldberg en 1989 [34].

L'algorithme implémenté est le suivant :

*Algorithme Génétique ;*
*Début*
*Génération d'une population initiale $P_t$ (t=0) de n individus;*
*Répéter*
*1. Evaluation de la fonction objectif f de chaque individu;*
*2. Sélectionner les 10% meilleurs de la population $P_t$ et les mettre dans $P_{t+1}$;*
*3. Remplir 70% de la population $P_{t+1}$ en croisant des chromosomes de $P_t$ sélectionnés aléatoirement;*
*4. Mutation de 20% de la population $P_t$ et les mettre dans $P_{t+1}$;*
*Jusqu'à (condition d'arrêt satisfaite)*
*Retourner la meilleure solution;*
*Fin-Algorithme.*

La mise en œuvre de cet algorithme passe par :

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

### 3.1 Le Croisement

Le croisement est un opérateur très important des algorithmes génétiques. L'utilisation des croisements simples connus (uni-point ou multipoints) ne semble pas fiable pour le problème de Job shop, car des fois ceci n'engendre que des solutions non réalisables au problème posé. Pour cela, nous avons opté pour la philosophie du croisement *uniforme* qui consiste à combiner les meilleurs partis de chaque chromosome père afin d'avoir deux meilleurs chromosomes fils. Ce croisement uniforme favorise le meilleur chromosome lors de l'échange

[4] Les résultats de ce paragraphe sont publiés dans [3]

des gènes.

Dans ce type de croisement, le masque est construit en favorisant le chromosome qui a le plus petit $C_{max}$ c'est-à-dire, on lui donne plus de probabilité d'apparaître.

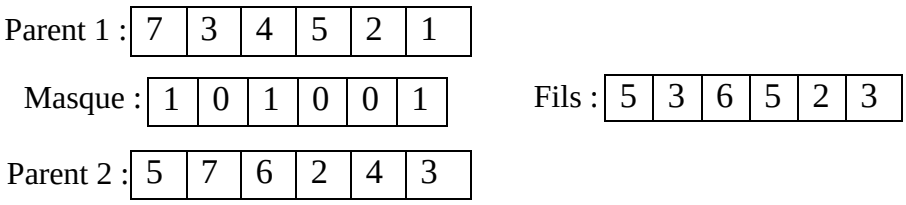


Figure 21. Exemple de croisement uniforme

### 3.2 La Mutation

Le rôle de la mutation est d'apporter une certaine diversité à la population et d'empêcher que celle-ci converge trop vite vers un seul type d'individu. En d'autres termes, il faut éviter d'avoir des populations où se répètent les mêmes individus.

Dans notre application, nous avons utilisé la mutation simple uni-point qui se fait par une altération d'un gène, choisi aléatoirement, d'un chromosome donné.

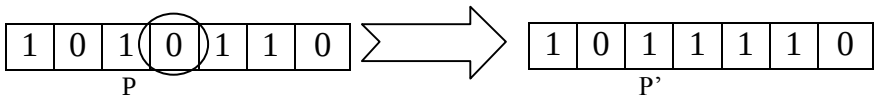


Figure 22. Exemple Mutation uni-point

### 3.3 La sélection

La sélection est appliquée afin de favoriser au cours du temps les individus les mieux adaptés et leur permettre de se reproduire. Dans les stratégies de sélection les plus connues, un meilleur chromosome dans l'ancienne population peut ne pas se reproduire dans la nouvelle population même si sa probabilité est élevée. Pour cela, nous avons choisi dans notre application d'implémenter une stratégie qui sélectionne les dix meilleurs pour cent de l'ancienne population pour se reproduire dans la nouvelle.

### 3.4 La population initiale

Avant de lancer l'algorithme génétique, une population initiale doit être générée. Cet ensemble d'individus qui servira de base pour les générations ultérieures doit être non homogène. Afin de permettre cette qualité, la génération de la population initiale dans notre application se fait selon le type de codage:

- Dans le codage binaire, nous avons utilisé deux stratégies :
  - ✓ l'algorithme heuristique.
  - ✓ la méthode par séparation et évaluation progressive.
- Dans le codage basé sur les règles de priorités, les chromosomes sont générés aléatoirement.

### 3.4.1 Algorithme heuristique

Cet algorithme heuristique a été utilisé la première fois par Mascis & Pacciarelli (2002) [59]. Il consiste à développer une solution au problème du job shop avec blocage en utilisant une fonction ***select*** qui sélectionne un sens pour un arc alternatif au hasard. Nous l'avons repris afin de l'exploiter dans la génération d'une population initiale. Mais malheureusement, cet algorithme est efficace pour les petits exemples où il génère une population initiale en un temps raisonnable. Cependant et par moment, il prend plusieurs heures d'exécution pour les exemples de grande taille pour générer un seul chromosome. Ce qui nous a incités à trouver une autre stratégie pour générer une population initiale qui ne comporte que des solutions réalisables. Nous avons testé la méthode la plus utilisée qui est la génération d'une population initiale d'une manière aléatoire, mais le spectre des solutions non réalisables apparaît toujours. Eviter les solutions non réalisables dans une population initiale est indispensable surtout pour le codage binaire. Sinon, nous risquons de ne parcourir que des solutions non réalisables et ceci va nuire à la qualité de notre algorithme génétique. C'est pour cette raison que nous avons opté pour une autre méthode décrite dans la prochaine section.

L'algorithme 3 ci-dessous, présente la méthode heuristique développée par Mascis & Pacciarelli [59]:

*Algorithme heuristique*
*(1) : Entrée : Un graphe alternatif G=(N,F,A)*
*Début*
*(2) S :=∅*
*(3) Tant que (A≠∅) faire*
*Sélectionner une paire ((h,k),(i,j))∈A,*
*Sélectionner (i,j), i.e. S :=S∪ {(i,j)}, A:=A- {((h,k),(i,j))},*
*Si ∃((h,k),(i,j))∈A : l(k,h)+$a_{hk}$ > 0, l(j,i)+$a_{ij}$ >0*
*Alors STOP, la procédure a échoué à trouver une solution réalisable.*
*Sinon tant que (∃((u,v),(p,q))∈A : l(v,u)+$a_{uv}$ > 0)*
*Faire S :=S∪{(p,q)}, A :=A-{((u,v),(p,q))}*
*(Tous les arcs induits par S sont sélectionnés)*
*Fait*
*Fait*
*Fin.*

### 3.4.2. Algorithme par Séparation et Evaluation

La deuxième technique utilisée pour générer une population initiale est l'algorithme par Séparation et Evaluation que nous avons développé dans le chapitre 4 de cette thèse.

Le principe de la méthode est simple : laisser l'algorithme SEP travailler pour avoir $N$ solutions réalisables (chromosomes), ensuite nous l'arrêtons. Nous sélectionnons parmi ces $N$ chromosomes les 50% meilleurs qui seront utilisés comme une population initiale. Avec cette méthode, nous pouvons garantir que la population initiale, ainsi générée, ne comporte que des solutions réalisables.

## 3.5 Evaluation des individus

La fonction *fitness* mesure les performances de chaque individu. Elle correspond à l'utilité de la solution par rapport au problème. Elle est utilisée pour déterminer la force de reproduction de chaque individu dans une population.

Dans notre application, la fitness d'un chromosome est représentée par son évaluation qui se fait de deux manières, selon le codage adapté, comme suit :

- en utilisant l'algorithme de Ford dans le codage binaire basé sur les arcs alternatifs.
- en utilisant une fonction d'évaluation *Eval* que nous avons développée pour le codage basé sur les règles de priorité.

### 3.5.1 Algorithme de Ford

L'algorithme de Ford permet d'évaluer chaque chromosome de la population en cours. Cette évaluation est la valeur du plus long chemin dans le graphe *G*, comportant des arcs alternatifs fixés suivant la solution donnée par ce chromosome.

L'algorithme de Ford n'est pas le seul algorithme utilisé pour le calcul du plus long chemin. Nous avons également utilisé l'algorithme de Bellman qui est largement répandu dans la littérature. Cependant, nous avons opté pour l'algorithme de Ford car il offre plus de fonctionnalité.

Par exemple: l'algorithme de Bellman travaille dans un graphe sans circuit (longueur positive et/ou nuls). Pour détecter un circuit de longueur nulle, nous somme obligés de passer par plusieurs algorithmes : composantes fortement connexes, mise à niveau, graphe réduit (voir chapitre 4 de cette thèse). Par contre, l'algorithme de Ford permet de détecter la présence d'un circuit de longueur positive. De plus, il fonctionne même avec la présence d'un circuit de longueur nulle dans le graphe.

Dans notre algorithme génétique, le module de détection de circuits de longueur positive est très important. En effet, il permet de détecter si un chromosome généré, après un croisement ou une mutation, est réalisable ou pas. Par conséquence, il offre la possibilité d'éliminer les chromosomes non réalisables de la nouvelle population.

*Algorithme de Ford (Recherche de plus long chemin)*
*Début*
*(1) Soit un graphe pondéré G=(X,U,poids) ayant une source s et un puits t.*
*(2)* Soient *d(s,x)la distance entre la source s et le sommet x et p(x,u) le poids de l'arc (x,u) dans G.*
*(3) Initialiser : d(s,s) :=0 et d(s,x) :=+∞ ∀x∈X.*
*(4) Répéter*
*Pour tout sommet x*
*Faire*
*Si d(s,x) <+∞*
*Alors*
*Pour (tout successeur u de x)*
*Faire Si d(s,u) < d(s,x)+p(x,u)*
*Alors d(s,u):= d(s,x)+ p(x,u)*
*Fin si*
*Fait*
*Fin si*
*Fait*
*Tantque (on modifie quelque chose)*
*Fin-Algorithme*

### 3.5.2. La Fonction *Eval*

La fonction Eval, présentée ci-dessous, permet de calculer le *makespan* d'un ordonnancement déjà codé par les règles de priorités.

*Algorithme Eval*
*Entrée : 1. un ordonnancement actif $O= \{o_1, o_2, o_5, \ldots o_n\}$.*
*2. la gamme opératoire.*
*Sortie : l'évaluation de l'ordonnancement $C_{max}$.*
*Début*
*Pour i=1 à n*
*Faire*
*Ordonnancer $o_i$ sur la machine $M_k$ qui lui correspond;*
*Mettre à jour les dates de début et de fin de l'opération $o_i$ ;*
*Fin pour;*
*$C_{max}$ reçoit la date de fin de l'opération $o_n$ ;*
*Fin.*
*Fin-Algorithme*

L'application numérique de l'Algorithme Génétique proposé qui correspond aux différents détails donnés dans cette partie, ainsi que les résultats obtenus sont résumés dans le chapitre 6. La qualité des solutions obtenues par l'AG nous a encouragé à proposer une parallélisation de

sa structure. Cette parallélisation se base sur le principe maître-esclave que nous détaillerons dans la section suivante de ce chapitre.

### 4. Algorithme Génétique Parallèle [5]

La conception de base des métaheuristiques offre de meilleures stratégies pour la recherche des solutions des problèmes d'optimisations combinatoires. Le parallélisme offre une amélioration de ces méthodes en explorant d'une manière intelligente beaucoup plus de solutions en des temps très satisfaisants. En appliquant des paramètres commodes, les métaheuristiques parallèles sont plus robustes que leurs versions séquentielles en termes de qualité des solutions obtenues.

Lors de la résolution du problème d'ordonnancement avec blocage d'une manière exacte, nous avons constaté le nombre très important de solutions explorées pour atteindre la solution optimale. Par contre, pour les problèmes de tailles supérieures à 10 jobs ×10 machines, nous devons aller vers des heuristiques.

Malgré cela, la qualité des solutions retournées par ces méthodes approchées nécessite encore des améliorations. Pour cette raison, nous avons pensé à paralléliser notre méthode basée sur les AG en utilisant le codage basé sur les règles de priorité pour voir l'impact du parallélisme sur la qualité des solutions retournées par notre AG.

Effectivement, l'utilisation du parallélisme permet d'accélérer de plus en plus le temps d'exécution des métaheuristiques. De plus, on c'est rendu compte que les métaheuristiques parallèles peuvent être beaucoup plus robustes que leurs versions séquentielles en termes de qualité de solutions trouvées. D'ailleurs, certains mécanismes parallèles permettent de trouver de meilleures solutions que les méthodes séquentielles, sans nécessiter un grand nombre d'itérations. Le parallélisme peut donc représenter un apport considérable aux métaheuristiques et un nombre grandissant de travaux sont effectués afin d'exploiter ce potentiel.

Les algorithmes génétiques s'adaptent très bien par leur structure à l'exécution parallèle. Avec l'avènement des machines de calcul parallèle, beaucoup de versions parallèles des Algorithmes génétiques ont vu le jour et ont fait l'objet de plusieurs études. Parmi les travaux les plus récents sur les algorithmes génétiques parallèles, citons l'apport de Guo [41] qui ont utilisé cette métaheuristique pour la résolution du problème de "*Conduction de chaleur*

[5] Les résultats de ce paragraphe sont publiés dans [3]

*inverse*" (*Inverse Heat Conduction*). Citons également Yeh et al. [94] qui ont proposé deux algorithmes génétiques parallèles pour la gestion de configuration de produit (Product Configuration Management).

La parallélisation des AG peut être réalisée de deux manières différentes : La première manière utilise le principe des populations à communication multiple et la seconde utilise le principe du maître-esclave. Les deux types d'AG parallèles ont été largement utilisés pour réduire le temps d'exécution de nombreuses applications. Le choix entre les deux méthodes parallèles est déterminé par plusieurs facteurs, tels que la facilité d'utilisation ou d'exécution et leur potentiel pour réduire les temps d'exécution. Généralement, les AG parallèles avec la méthode maître-esclave sont plus simples, car il est plus facile de configurer leurs paramètres de contrôle et leur mise en œuvre. En revanche, les AG avec les populations à communication multiple permettent _plus de parallélisme, mais sont plus difficiles à mettre en œuvre car dans ce type de parallélisme, il est nécessaire de définir les valeurs de plusieurs paramètres supplémentaires, outre les paramètres habituels de l'AG lui-même qui affectent l'efficacité et la qualité des solutions trouvées par cette méthode.

Lors de notre implémentation parallèle, notre intérêt s'est porté sur l'exploitation des réseaux d'ordinateurs et non pas sur les machines (ordinateurs) parallèles multiprocesseurs. En effet, un réseau relié par une topologie étoile, nous permet d'assurer la communication entre les postes, soit en poste à poste (communication directe entre deux machines) ou en multicast (diffusion des informations sur l'ensemble du réseau).

Dans notre modèle de parallélisation, un poste maître est dédié aux étapes : de sélection, de croisement et de mutation. Le reste des postes esclaves s'occupent du calcul de la fitness des individus (voir Figure 23). Ce choix est motivé par le fait que le temps, alloué au calcul des fitness des solutions générées, occupe la majorité du temps global de calcul de cette méthode, dans la version AG séquentiel.

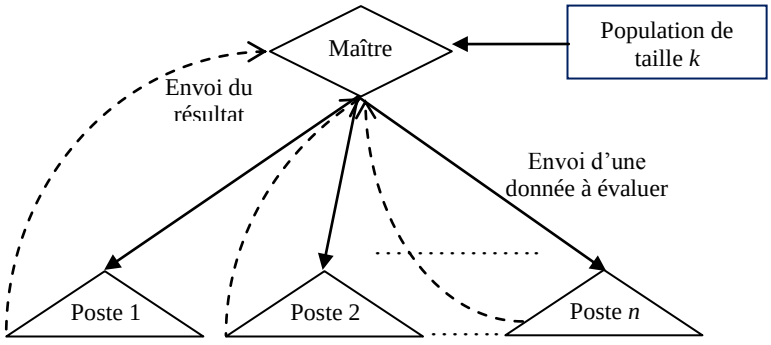


Figure 23. Principe de fonctionnement de la parallélisation des calculs

Les avantages de cette technique sont nombreux :

- L'AG en mode maître-esclave explore l'espace de recherche exactement de la même manière qu'un AG séquentiel. Donc, les structures de base utilisées dans la conception de l'AG séquentiel restent valables.
- Sa mise en œuvre est très simple.
- Dans de nombreuses applications, l'AG en mode maître-esclave offre des améliorations significatives aux résultats finaux.

Un ordinateur maître assure la génération des populations d'individus en utilisant la sélection, le croisement et la mutation. Ensuite les solutions générées sont envoyées vers les stations esclaves pour effectuer le calcul des fitness. De cette façon, le temps global de calcul est partagé entre tous les ordinateurs du réseau.

Cette méthode de parallélisation conserve toutes les caractéristiques que l'AG séquentiel sans aucune modification des paramètres déjà fixés. Le seul paramètre qui a subi des changements dans l'algorithme parallèle c'est la taille de la population qui dépend directement du nombre d'ordinateurs de notre réseau, puisque l'ordinateur maître partage la population en cours de traitement en un certain nombre proportionnel au nombre d'ordinateurs disponibles sur le réseau. Il envoie alors les sous-populations aux différents ordinateurs esclaves où le calcul de la fitness est réalisé.

Les étapes de l'algorithme parallèle, que nous avons proposé et implémenté sur le serveur, sont les suivantes :

*AG parallèle implémenté sur l'ordinateur maître.*
*Début*
*Soit un réseau de k+1 ordinateurs.*
*Générer une population $P_t$ initiale de taille n, la valeur de n doit être proportionnelle au nombre de postes k du réseau.*
*Répéter*
*1. Partager la population $P_t$ en k sous populations de tailles égales.*
*2. Dispatcher les k sous populations sur les différents postes esclaves.*
*3. Attendre une réponse des postes esclaves.*
*4. A la réception des résultats du calcul de la fitness de chaque poste esclave.*
*5. Sectionner les meilleures 10% de la population $P_t$ et les mètres dans la population $P_{t+1}$.*
*6. Remplir 70% de la population $P_{t+1}$ en utilisant l'opérateur de croisement.*
*7. Muter 20% de la population $P_t$ et les mettre dans $P_{t+1}$;*
*8. $P_t$- $P_{t+1}$.*
*Jusqu'à (condition d'arrêt satisfaite)*
*Retourner la meilleure solution trouvée*
*Fin-Algorithme.*

Les étapes de l'algorithme parallèle qui fonctionne sur les différents ordinateurs esclaves sont données comme suit:

*Algorithme implémenté sur les postes esclaves*
*Début*
*(1) A la réception d'une sous-population $P_i$ 1<=i<=.k.*
*(2) Evaluer la fonction objectif de chaque individu de $P_i$.*
(3) *Envoyer les résultats du calcul à l'ordinateur maître*
*Fin-Algorithme*

## 5. Optimisation par Essaims Particulaires OEP[6]

L'optimisation par Essaim de Particules (PSO) est une métaheuristique originale inspirée du comportement d'un essaim d'oiseaux. Dans cette partie, nous proposons un algorithme d'optimisation basé sur la métaheuristique PSO adaptée au cas discret, pour la résolution du problème d'ordonnancement Job Shop avec blocage. Dans notre algorithme, les notions de particule et de vitesse sont redéfinies, et une approche efficace est développée pour le mouvement des particules. Pour vérifier l'algorithme proposé de PSO, des comparaisons avec un algorithme génétique et une méthode exacte sont faites. Les résultats informatiques obtenus prouvent que l'algorithme proposé de PSO est très concurrentiel.

Le modèle de base PSO se compose d'un essaim de particules se déplaçant dans un espace à *n* dimensions. En cours de leurs mouvements, les particules ajustent leurs positions en fonction de leurs propres expériences et les expériences des autres particules voisines, faisant usage de la meilleure position rencontrée par elles-mêmes et leurs voisines. Chaque particule *i* au temps *t* a les caractéristiques suivantes:

- $X_i(t)$ est le vecteur position,
- $V_i(t)$ est le vecteur vitesse,
- $Pbest_i(t)$ est la meilleure position rencontrée par la particule *i* jusqu'ici et
- $Gbest(t)$ est la meilleure position globale obtenue en communiquant avec ses particules voisines (de la même population).

Ce flux d'information est réalisé par la définition d'une topologie de notre essaim. A chaque itération, la vitesse est mise à jour et la particule se déplace vers une nouvelle position. Cette nouvelle position est calculée à l'aide de la formule (1) suivante qui combine entre la position précédente et la nouvelle vitesse : $X_i(t+1) = X_i(t) + V_i(t+1)$.....(1)

[6] Les résultats de ce paragraphe sont publiés dans [2]

La vitesse est mise à jour en utilisant la formule suivante:

$$V_i(t+1) = \alpha \times V_i(t) + \beta \times (Pbest_i(t) - X_i(t)) + \gamma \times (Gbest(t) - X_i(t)).....(2)$$

Où les paramètres $\alpha$ , $\beta$ et $\gamma$ sont des paramètres fixés de manière empirique.

L'algorithme implémenté est le suivant :

*Algorithme PSO*
*Début*
*(1) Initialisation des paramètres*
*(2) Initialisation de l'essaim (position et vitesse)*
*(3) Evaluation*
*(4) Répéter*
*{ Déterminer la meilleure position P*
*Déterminer la meilleure position de tout ou partie de l'essaim G*
*Mettre à jour les vitesses*
*Mettre à jour les positions*
*Evaluation*
*Recherche locale (optionnel)*
*} Jusqu'à (Critère d'arrêt satisfaisant).*
*Fin-Algorithme.*

La mise en œuvre de cet Algorithme passe par :

- L'implémentation de la représentation convenable des solutions du problème posé (codage);
- L'implémentation des opérateurs de changement de position et de vitesse;
- La détermination des différents paramètres de la méthode: taille de la population, nombre de générations, les coefficients, etc.

### 5.1 Le codage des solutions

Le codage des solutions est un déterminant important de l'efficacité de la méthode PSO. Il signifie la transcription d'un ordonnancement réel en représentation adéquate permettant la réalisation des différents opérateurs de PSO. Les méthodes de codage des particules utilisées avec PSO sont les mêmes techniques (Codage binaire des arcs alternatifs, codage par les règles de priorité et le codage par rang) utilisées avec les AG (voir section 2 de ce chapitre).

### 5.2 Opérateurs vitesse

L'algorithme PSO a été développé pour résoudre les problèmes d'optimisation non-linéaires avec des fonctions objectifs continues. Dans la littérature, plusieurs auteurs ont proposé d'adapter certains opérateurs, ayant trait aux calculs de la vitesse et des nouvelles

positions, pour pouvoir appliquer PSO aux problèmes d'optimisation combinatoire discrets. En particulier, le problème du voyageur de commerce (PVC).

Soit *x* un vecteur de *n* cases représentant une solution du problème de voyageur de commerce PVC. Chaque case du vecteur *x* représente une ville à visiter par le voyageur du commerce. L'ordre des villes décrit par le vecteur *x* défini une solution au problème.

Afin d'explorer l'espace des solutions du PVC, une technique connue dans la littérature et très simple est utilisée : c'est la technique des permutations aléatoires. Son principe est caractérisé par la permutation entre deux cases de *x* choisies d'une manière aléatoire. Ceci nous permet de déduire une nouvelle solution au problème qui est en plus réalisable. Ce principe a été l'axe adopté par la plupart des chercheurs ayant voulu adapter PSO aux problèmes discrets d'optimisation combinatoire. Dans cette thèse, nous avons repris cette manière de faire, en développant la première variante de l'algorithme PSO qui se base sur ce principe. Ensuite, nous avons critiqué cette manière de faire et nous avons proposé une autre méthode plus originale. Cette méthode permet d'adapter PSO dans sa version continue au problème d'ordonnancement job shop avec blocage. Dans ce qui suit, nous allons détailler les deux variantes de PSO.

### 5.2.1. 1ère Variante de PSO

Dans cette variante, une position *X* est soit une suite de 0 et 1 représentant un choix des arcs alternatifs ou bien une suite de chiffres entre 0 et 9 représentant les règles de priorité. Dans les deux cas, *X* représente une solution de notre problème. Une vitesse *V* est une suite de couples représentant les différentes permutations effectuées sur une position *X*.

Les opérateurs appliqués dans cette variante sont [95] :

**a. Soustraction (position, position) → vitesse :** La soustraction entre deux positions $P_1$ et $P_2$ donne comme résultats une vitesse qui est représentée par les permutations nécessaires appliquées sur $P_2$ pour passer à $P_1$.

**b. Multiplication externe (réel, vitesse) → vitesse :** Multiplication externe d'une vitesse *V* par un scalaire $c$.

*Exemple:* Vitesse $V$ = ((2,6), (3,4), (2,8))

$c = 0$ : $c \times V = \emptyset$

$0 < c < 1$: $c \times V$ est une vitesse issue de la troncature de $V$ Exemple : $c$ = 0.5, $c \times V$ = ((2,6))

$c > 1$ : $c \times V$ est une vitesse issue de la duplication de tout ou d'une partie de $V$ .

*Exemple :* $c$ = 2.5 alors $c \times V$ = ((2,6),(3,4),(2,8),(2,6),(3,4),(2,8),(2,6))

$c < 0$ : $c \times V$ vitesse issue d'une inversion de la liste des permutations

Exemple $c$ = -1 alors $c \times V$ = ((2,8),(3,4),(2,6))

**c. Addition (vitesse, vitesse) → vitesse :** Concaténation des deux listes de permutation.

*Exemple :*

Si $V_1$ = ((2,6),(3,4),(2,8)) et $V_2$ = ((3,5),(2,7)) Alors

V1 + V2 = ((2,6),(3,4),(2,8),(3,5),(2,7))

**d. Déplacement (position, vitesse) → position :** Application de l'ensemble des transformations de la vitesse sur la position initiale afin de trouver comme résultat une position finale.

*Exemple :*

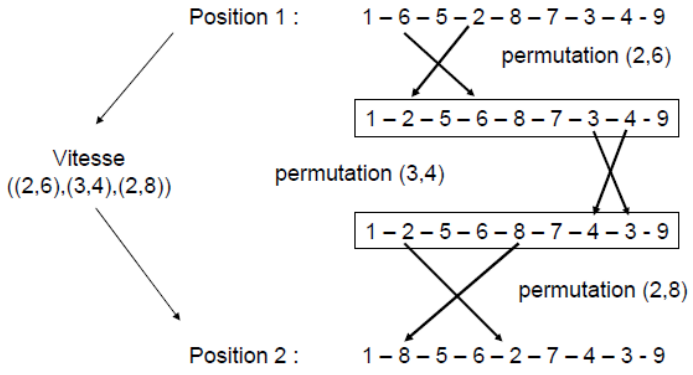


Figure 24. Application de la vitesse sur une position

### 5.2.2. 2ème Variante de PSO

Pour le codage binaire, la particule est représentée par une suite de 0 et de 1 indiquant l'orientation choisie pour chaque arc alternatif. L'idée de cette variante découle du fait que nous considérons notre particule comme un nombre binaire qui peut être traduit en un nombre décimal que nous allons utiliser dans le calcul des nouvelles positions. A la fin de tous les calculs, nous allons procéder à la conversion de la partie entière du nombre décimal résultant en binaire qui donnera la nouvelle position. Chaque position est évaluée par l'algorithme de Ford. L'inconvénient de ce codage est le fait qu'il peut tomber souvent sur des solutions non

réalisables lors de l'application du processus de calcul de la nouvelle position. Les solutions retournées par moment ne sont pas bonnes, d'où la migration vers le codage basé sur les règles de priorité. Cependant, pour ce dernier codage, une particule est représentée comme une séquence de chiffres entre 0 et 9. Chaque valeur exprime l'une des règles de priorité donnée dans le Tableau 3. Cette particule peut être, donc, considérée comme une valeur décimale qui peut être manipulée dans des opérations arithmétiques standards. Cela nous permettra d'appliquer les équations (1) et (2) et de terminer le processus de PSO sans aucun problème. Pour les autres paramètres, il n'y a pas de règle pour les déterminer; il faudra réaliser de nombreux essais afin d'acquérir l'expérience nécessaire pour l'estimation de ces paramètres.

### 6. Algorithme PSO parallèle proposé[7]

Nous avons tenté une parallélisation de notre méthode PSO avec le codage basé sur les règles de priorité. Ceci a été fait dans le but de voir l'impact du parallélisme sur la qualité des solutions retournées.

L'idée principale utilisée avec la parallélisation des AG est réutilisée avec la parallélisation de PSO. Dans la littérature, il y a un manque flagrant de documentation traitant ce coté de la méthode. Donc pour développer l'algorithme PSO parallèle, nous avons suivi la démarche de parallélisation à gros grains. Elle repose sur le paradigme maitre/esclave. Dans notre cas, chaque client évalue sa propre population en parallèle sur des machines différentes, c'est-à-dire que chaque client exécute l'algorithme PSO Séquentiel.

Un programme maître (au niveau du Serveur) se charge de lancer les paramètres nécessaires (nombre de particules, nombre d'itérations) au fonctionnement des postes clients. A chaque itération, chaque client renvoie sa meilleure particule avec son *Cmax* correspondant vers le serveur (poste maître) dont l'une des fonctionnalités principales est l'écoute des résultats de chaque client. Le serveur choisit parmi les particules reçues la meilleure d'entre elles et la renvoie à son tour à tous les clients pour les informer de la meilleure particule dans le groupe (Gbest). Ensuite chaque client effectue la mise à jour à son niveau de la valeur de $G_{best}$ si nécessaire.

Le but de cette parallélisation est principalement d'essayer de parcourir, d'une manière intelligente, le plus grand nombre possibles de solutions.

[7] Les résultats de ce paragraphe sont publiés dans [2]

La topologie physique utilisée est la topologie étoile, car c'est la topologie qui convient le mieux à notre application PSO parallèle. Dans cette topologie, chaque poste du réseau peut communiquer directement avec un autre poste ou bien il a aussi la possibilité de dispatcher une information sur tout le réseau sans aucun problème.

Dans un système client/serveur, le fonctionnement est comme suit :

- Le client émet une requête vers le serveur grâce à son adresse IP qui désigne un service particulier du serveur.
- Le serveur reçoit la demande et répond à l'aide de l'adresse IP de la machine cliente qui est incluse dans la requête initiale.

Les différentes informations communiquées par le poste *maître* vers un *calculateur* sont :

- Le nombre de particules composant la sous population qu'il va évaluer.
- Le nombre d'itérations à réaliser.
- La meilleure particule $G_{best}$ calculée jusqu'à présent par tous les postes.

A chaque itération, chaque poste calculateur évalue la sous population en cours et il envoie sa meilleure position $P_{best}$ au sélectionneur.

Dans ce qui suit, nous présentons les algorithmes décrivant les étapes de la méthode PSO Parallèle. Le premier tourne sur le serveur et le deuxième tourne sur les postes clients.

Les étapes de l'algorithme parallèle PSO implémenté sur le poste maître sont:

*Algorithme PSO Parallèle pour le poste maître.*
*Début*
*Soit un réseau informatique avec (K+1) postes.*
*Communiquer à tous les postes clients:*
*1/ Le Nombre de particules par population.*
*2/ Le nombre d'itérations global.*
*3/ Gbest*=∅, Cmax*= ∅.*
*4/ Lancer le processus d'exécution.*
*Répéter*
*1/ A la réception d'une nouvelle valeur $Gbest_i$ du poste client i.*
*2/ Si ($Gest_i$ reçu est meilleure que Gbest*)*
*Alors*
*Gbest*= $Gbest_i$*
*Cmax*=$Cmax_i$ ;*
*Diffuser la nouvelle Gbest* sur tous les postes clients;*
*Jusqu'à (Satisfaction du critère d'arrêt) ;*
*Retourner la meilleure solution.*
*Fin-Algorithme.*

L'algorithme parallèle PSO implémenté sur les postes clients (esclaves) est le suivant:

*Algorithme PSO parallèle pour les postes clients;*
*Début*
*A la réception du :*
*1/ Nombre de particules par population.*
*2/ le nombre d'itérations global.*
*3/ Gbest*.*
*Faire*
*1/Initialisation de l'essaim;*
*2Initialisation des positions et des vitesses;*
*3/Evaluation;*
*4/Envoyer Gbest au serveur.*
*Répéter*
*Déterminer la meilleure position Pbest;*
*Déterminer la meilleure position de l'essaim Gbest;*
*Envoyer Gbest au serveur;*
*Mettre à jour les vitesses et les positions;*
*Evaluation*
*Recherche Locale (optionnelle)*
*Mettre à jour Gbest à partir du serveur;*
*Jusqu'à (Satisfaire le critère d'arrêt)*
*Fin-Algorithme.*

## 7. Conclusion

Dans ce chapitre, nous avons présenté les différentes techniques que nous avons développées pour pouvoir appliquer les deux métaheuristiques AG et PSO sur le problème de job shop avec blocage. Nous avons constaté que le codage basé sur les règles de priorité est plus performant que le codage binaire. Cette meilleure performance est motivée par le fait que ce codage n'explore pas de solutions non réalisables. Nous avons aussi, développé deux techniques de parallélisation permettant d'améliorer les performances des deux métaheuristiques. Une étude comparative entre ces deux méthodes appuyée des résultats obtenus sur des instances connues dans la littérature fera l'objet du prochain chapitre.

# Chapitre 6
# Expérimentations numériques

Le présent chapitre est dédié à la présentation et à l'analyse des résultats retournés par les différentes méthodes de résolutions présentées tout au long de cette thèse. Afin de rendre le processus de comparaison plus efficace, nous avons choisi un ensemble d'exemples, très utilisés dans la littérature, nommés les Benchmarks de Lawrence [51].

Toutes les méthodes proposées dans cette thèse sont testées sur ces exemples. Une comparaison des résultats obtenus est ensuite effectuée avec ceux donnés par les méthodes développées dans la littérature et bien entendu appliquées sur les mêmes exemples.

Pour mieux structurer la présentation de nos résultats, nous divisons ce chapitre en deux parties : Dans la première partie, nous présentons les résultats numériques retournés par nos méthodes et dans la deuxième partie, nous exposons l'analyse des comparaisons entre les différentes méthodes de résolution.

## 1. Résultats des méthodes par Séparation et Evaluation

### 1.1 Résultats issus des méthodes séquentielles

Nous présentons dans le Tableau 5, les résultats de notre méthode SEP séquentielle avec l'évaluation de Bellman. La première colonne de ce tableau représente la taille du problème exprimée en termes de nombre de machines et de nombre de jobs. Nous attirons l'attention du lecteur que le nombre de solutions générées par un exemple de *n* jobs et *m* machines, au pire des cas, est: $(n!)^m$. Cette valeur représente la taille de l'espace de recherche i.e. le nombre de combinaisons possibles.

La deuxième colonne, contient les résultats retournés par la méthode du Recuit Simulé, appliquée au problème d'ordonnancement avec blocage, que nous avons développé. Nous avons utilisé les résultats de cette technique afin d'avoir une bonne borne supérieure BU initiale au problème posé. Ceci permettra de faire des stérilisations de branches dans l'arbre de recherche. La troisième colonne, dévoile les résultats des *Cmax* retournés par notre SEP séquentielle. La quatrième colonne, donne le temps de réponse de notre méthode. Dans la $5^{ème}$ colonne, nous avons calculé le nombre de nœuds stérilisés dans l'arborescence de recherche dans le cas où l'évaluation du nœud dépasse la valeur de la *UB*. La $6^{ème}$ colonne représente le

nombre de nœuds contenant des solutions non réalisables de l'arborescence. Ces solutions contiennent des circuits de longueur positive. Notons que la détection de ces circuits est immédiate grâce à l'algorithme des composantes fortement connexes. Ce type de nœuds fait aussi l'objet d'une stérilisation.

| **Nbjobs ×Nbmachines** | **Recuit Simulé (UB)** | $C_{max}$ | **Temps** | **Nb_noeuds Eval>=UB** | **Nb_noeuds non réalisables** |
|---|---|---|---|---|---|
| 4×5 | 395 | 395 | 188 (ms) | 81 | 105 |
| 4×10 | 633 | 633 | 4,875 (s) | 332 | 321 |
| 4×15 | 957 | 955 | 25,578 (s) | 259 | 429 |
| 5×10 | 778 | 681 | 77,250 (s) | 1695 | 3031 |
| 5×15 | 994 | - | - | - | - |
| 6×5 | 435 | 427 | 2,73 (mn) | 8904 | 48667 |
| 6×10 | 765 | 758 | 9,70 (h) | 264224 | 343350 |
| 6×15 | 1247 | 1229 | 7,30 (h) | 40951 | 119193 |
| 7×5 | 679 | 638 | 36,35 (mn) | 113484 | 416412 |
| 8×5 | 677 | 590 | 29,64 (mn) | 3347340 | 7516371 |
| 9×5 | 722 | 690 | 3,87 (h) | 397858 | 1679816 |

Tableau 5. Résultats de notre méthode SEP séquentielle

Nous constatons, à partir de ces résultats que:

- Le Recuit Simulé donne généralement de bonnes solutions approchées. Ces solutions sont égales aux solutions exactes pour les exemples de petite taille.
- L'unité de temps de réponse varie entre secondes, minutes et heures, selon les caractéristiques particulières de l'exemple en question.
- On peut traiter deux exemples de même taille avec un temps de réponse pour le premier exemple beaucoup plus petit que le temps de réponse pour le deuxième exemple. Ceci peut être expliqué par la complexité de l'exemple lui-même. Par complexité d'un exemple, nous entendons le nombre de circuits de longueur nulle qu'il comporte. En effet, la détection de ces circuits prend plus de temps.
- Nous remarquons entre les deux colonnes cinq et six que le nombre de nœuds stérilisés en raison de la non réalisabilité des solutions générées est plus important que le nombre de nœuds stérilisés par la borne supérieure. Ceci est dû à la qualité médiocre de la borne inférieure utilisée.

Dans le Tableau 6 ci-dessous, nous présentons une comparaison entre les deux méthodes d'évaluation que nous avons utilisées avec notre méthode SEP. La première évaluation est basée sur l'algorithme de Bellman de recherche de plus long chemin. Nous avons combiné ce dernier avec les algorithmes : de mise à niveau d'un graphe pour détecter les circuits, de

recherche de composantes fortement connexes pour trouver les circuits et de réduction des graphes pour passer aux graphes sans circuits. La deuxième évaluation est basée sur l'algorithme de Ford pour la recherche des plus longs chemins dans un graphe.

| **nb_jobs × nb_machines** | **Recuit Simulé** | ***Cmax* Bellman** | ***Cmax* Ford** | **Temps (secondes) Bellman** | **Temps (secondes) Ford** |
|---|---|---|---|---|---|
| 4×5 | 395 | 395 | 395 | 188×10-3 | 0,031 |
| 4×10 | 633 | 633 | 633 | 4,875 | 0,516 |
| 4×15 | 957 | 955 | 955 | 25,578 | 0,531 |
| 5×10 | 778 | 681 | 681 | 77,250 | 7,28 |
| 5×15 | 994 | - | 923 | - | 2 329,36 |
| 6×5 | 435 | 427 | 427 | 168,8 | 21,18 |
| 6×10 | 765 | 758 | 758 | 34920 | 1715,70 |
| 6×15 | 1 247 | 1 229 | 1 229 | 26289,5 | 45020,96 |
| 7×5 | 679 | 638 | 638 | 2181 | 424,07 |
| 8×5 | 677 | 659 | 659 | 1778,4 | 12059,7 |
| 9×5 | 722 | 690 | 690 | 13932 | 2580 |

Tableau 6. Résultats du SEP en utilisant Bellman et Ford.

Nous pouvons noter d'après le Tableau 6 qu'en termes de temps d'exécution, l'évaluation de Ford donne de meilleurs temps sauf pour l'exemple 6×15 dans lequel l'évaluation de Bellman semble meilleure. Ceci peut être interprété facilement par le fait que le graphe alternatif correspondant à l'exemple en question est composé de beaucoup de circuits ou non. En effet, si le graphe n'admet pas beaucoup de circuits, l'évaluation de Bellman serait meilleure par contre l'évaluation de Ford l'emporterait dans le cas contraire.

### 1.2 Résultats issus de la méthode parallèle

Nous avons réalisé les mêmes tests en appliquant notre méthode par séparation et évaluation parallèle sur les exemples cités ci-dessus. Les résultats obtenus sont présentés dans le Tableau 7, qui est similaire au Tableau 5 avec quelques petites modifications. Nous avons enlevé la colonne «borne supérieure » et nous l'avons remplacé par la colonne « nombre de postes ». Dans cette dernière, nous avons donné le nombre de postes utilisés dans le réseau afin de résoudre l'exemple en question. Par contre, le nombre de sommets est calculé à la base de la moyenne des nombres des solutions stérilisées par poste de calcul dans le réseau.

L'apport du parallélisme apparaît très bien dans l'exemple 5×10. En effet, dans cette instance, le temps de réponse pour 3 postes est de 22,922(s) et pour 4 postes 6,047(s) seulement contre 77,250(s) dans la partie séquentielle.

L'avantage du parallélisme apparaît encore mieux avec les exemples 6×5, 6×10 et 9×5 qui ont les temps d'exécution de : 7.30(h), 9.70(h) et 3.87(h), respectivement, dans la version

séquentielle. Ces temps ont diminués jusqu'à : 22.531(s), 1.20(h) et 25(mn), respectivement, avec 3, 4 et 3 postes de calcul dans le réseau.

La remarque la plus importante est que la méthode SEP parallèle est beaucoup plus rapide en augmentant le nombre de postes dans le réseau, grâce à l'astuce de l'anneau logique, par rapport aux autres techniques de parallélisation basées sur le principe maître-esclave. Ces dernières s'alourdissent avec l'augmentation du nombre de postes à cause des problèmes de communication entre eux.

| **Nb jobs × Nb machines** | $C_{max}$ | **Temps de réponse (seconde)** | **Nombre de nœuds Eval >=UB** | **Nombre de nœuds non réalisables** | **Nombre de PCS** |
|---|---|---|---|---|---|
| 4×5 | 395 | 9,547 | 40 | 99 | 3 |
| 4×10 | 633 | 18,360 | 157 | 163 | 3 |
| 4×15 | 955 | 16,656 | 20 | 6 | 3 |
| 5×10 | 681 | 22,922 | 155 | 106 | 3 |
| 5×10 | 681 | 6,047 | 31 | 122 | 4 |
| 5×15 | 806 | 87 413,531 | 340 269 | 799 159 | 3 |
| 6×5 | 427 | 22,531 | 2 146 | 9 578 | 3 |
| 6×15 | 1 229 | 46,046 | 319 | 743 | 3 |
| 6×10 | 758 | 4 325,672 | 65 397 | 79 455 | 4 |
| 7×5 | 638 | 260,453 | 28 073 | 84 415 | 3 |
| 8×5 | 659 | 17 139,406 | 811 380 | 1 885 852 | 4 |
| 9×5 | 690 | 1 501,250 | 91 790 | 331 750 | 3 |

Tableau 7. Résultats de la méthode SEP Parallèle

## 2. Résultats des métaheuristiques

Les résultats des métaheuristiques donnés ci-dessous ont été obtenus après avoir fixé les paramètres empiriques de celles-ci. Le choix des valeurs de ces paramètres est décrit dans la section 3 de ce chapitre.

### 2.1 Résultats issus des méthodes séquentielles

Dans le but de voir les performances de nos méthodes AG et PSO séquentielles, nous avons effectué une série d'essais sur 20 problèmes de taille et de complexité diverses. Pour chaque problème, cinq expériences ont été réalisées et nous prenons la meilleure.

Le Tableau 8 et la Figure 25 ci-dessous rapportent les résultats obtenus. La première colonne du Tableau 8 donne le numéro de l'exemple comme il a été cité dans la littérature (Exemples de Lawrence). La colonne 2 donne la dimension de l'exemple en termes de nombre de jobs et de nombre de machines. Les colonnes 3 et 4 donnent les résultats obtenus par nos méthodes séquentielles AG et PSO. La 5$^{ème}$ et dernière colonne donne une comparaison avec la méthode Recherche Taboue réalisée par Gröflin & Klinkert [40]. La figure 1 décrit une comparaison entre les métaheuristiques AG, PSO et Recuit Simulé que

nous avons développé avec la dernière solution donnée par la Recherche Taboue de Gröflin & Klinkert [40].

| Exemple | Nb jobs × Nbmachines | AG | PSO | Recherche Taboue |
|---|---|---|---|---|
| la01 | 10×5 | 852 | 831 | 836 |
| la02 | 10×5 | 830 | 799 | 824 |
| la03 | 10×5 | 790 | 812 | 765 |
| la04 | 10×5 | 799 | 832 | 774 |
| la05 | 10×5 | 698 | 730 | 711 |
| la06 | 15×5 | 1 224 | 1 284 | 1 191 |
| la07 | 15×5 | 1 221 | 1 217 | 1 119 |
| la08 | 15×5 | 1 206 | 1 277 | 1 161 |
| la09 | 15×5 | 1 262 | 1 382 | 1 269 |
| la10 | 15×5 | 1 232 | 1 326 | 1 222 |
| la11 | 20×5 | 1 666 | 1 784 | 1 615 |
| la12 | 20×5 | 1 449 | 1 484 | 1 433 |
| la13 | 20×5 | 1 622 | 1 711 | 1 576 |
| la14 | 20×5 | 1 620 | 1 763 | 1 648 |
| la15 | 20×5 | 1 657 | 1 801 | 1 667 |
| la16 | 10×10 | 1 238 | 1 245 | 1 175 |
| la17 | 10×10 | 1 050 | 1 048 | 1 040 |
| la18 | 10×10 | 1 130 | 1 130 | 1 112 |
| la19 | 10×10 | 1 179 | 1 180 | 1 124 |
| la20 | 10×10 | 1 188 | 1 252 | 1 184 |
| la21 | 15×10 | 1 775 | 1 848 | 1 617 |

Tableau 8. Résultats de l'AG et PSO.

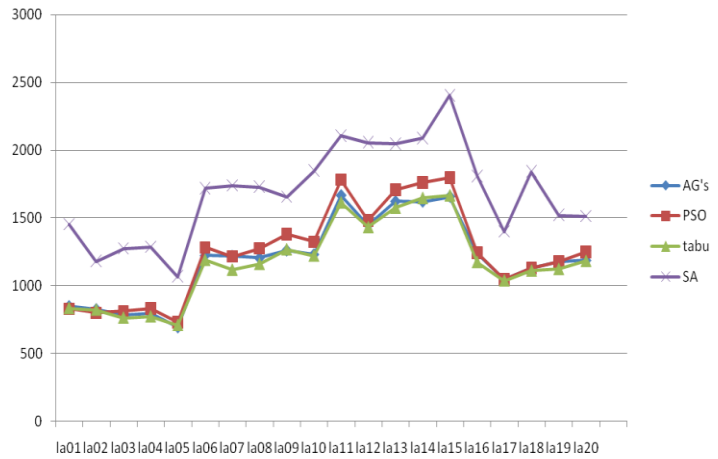


Figure 25. Résultats de nos métaheuristiques comparés avec la Recherche Taboue

Il peut être intéressant de voir les relations de dominance empirique afin de déduire un

classement entres ces méthodes. On dit qu'une métaheuristique domine empiriquement une autre si, pour tous les problèmes tests, les solutions données par la première sont égales ou meilleures que celles obtenues par la deuxième, avec au moins une solution strictement meilleure. On remarque bien d'après le Tableau 8 qu'il n'existe pas de relation de dominance entre les deux métaheuristiques AG et PSO. Tandis que la Recherche Taboue domine les deux méthodes AG et PSO séquentielles.

Nous avons aussi, comparé nos méthodes avec les heuristiques SMCP, SMBP et SMSP développées par A .Mascis a, D. Pacciarelli [59] en utilisant les benchmarks de Lawrence. Le Tableau 9 donne ces résultats. Nous remarquons clairement que l'Algorithme Génétique basé sur les règles de priorités AGP domine empiriquement toutes les autres heuristiques.

| **Instance** | **Taille** | **SMCP SMBP** | **SMSP** | **AGP** |
|---|---|---|---|---|
| La 01 | 10×5 | 1 137 | 1 197 | 852 |
| La 02 | 10×5 | 1 055 | 1 087 | 830 |
| La 03 | 10×5 | 1 088 | - | 790 |
| La 04 | 10×5 | 1 378 | 971 | 830 |
| La 05 | 10×5 | 1 116 | - | 732 |
| La 06 | 15×5 | 1 493 | - | 1 224 |
| La 07 | 15×5 | 1 737 | - | 1 240 |
| La 08 | 15×5 | 1 569 | - | 1 240 |
| La 09 | 15×5 | 1 943 | - | 1 354 |
| La 10 | 15×5 | 1 833 | 1 533 | 1 232 |
| La 11 | 20×5 | 2 189 | - | 1 666 |
| La 12 | 20×5 | 2 122 | - | 1 465 |
| La 13 | 20×5 | 2 296 | - | 1 659 |
| La 14 | 20×5 | 2 423 | - | 1 705 |
| La 15 | 20×5 | 2 371 | - | 1 693 |
| La 16 | 10×10 | 1 593 | 1 592 | 1 255 |
| La 17 | 10×10 | 1 719 | - | 1 079 |
| La 18 | 10×10 | - | - | 1 180 |
| La 19 | 10×10 | 1 786 | - | 1 179 |
| La 20 | 10×10 | 1 624 | - | 1 204 |
| La 21 | 15×10 | 2 701 | - | 1 894 |
| La 22 | 15×10 | 2 566 | - | 1 603 |
| La 23 | 15×10 | 3 044 | - | 1 806 |
| La 24 | 15×10 | 3 350 | - | 1 780 |
| La 25 | 15×10 | 2 211 | - | 1 767 |
| La 26 | 20×10 | 4 024 | 4 024 | 2 366 |
| La 27 | 20×10 | 3 488 | - | 2 414 |
| La 28 | 20×10 | 3 070 | - | 2 415 |
| La 29 | 20×10 | 3 792 | - | 2 373 |
| La 30 | 20×10 | 3 173 | - | 2 382 |
| La 31 | 30×10 | - | - | 3 573 |
| La 32 | 30×10 | - | - | 3 826 |
| La 33 | 30×10 | - | - | 3 403 |
| La 34 | 30×10 | 4 561 | - | 3 571 |
| La 35 | 30×10 | - | - | 3 668 |
| La 36 | 15×10 | 3 369 | - | 2 150 |

| | | | | |
|---|---|---|---|---|
| La 37 | 15×10 | 3 009 | - | 2 246 |
| La 38 | 15×10 | 3 187 | - | 2 132 |
| La 39 | 15×10 | 3 787 | - | 2 135 |
| La 40 | 15×10 | 3 345 | - | 2 011 |

Tableau 9. Résultats de notre AGP avec les autres heuristiques

### 2.2 Résultats issus de la méthode parallèle

Dans le Tableau 10, nous présentons les résultats de notre AG parallèle, appliquée sur 40 exemples représentant les Benchmarks de Lawrence [51]. Ces résultats sont comparés aux meilleures solutions obtenues avec l'heuristique Recherche Taboue de Gröflin & Klinkert [40]. Nous pouvons constater que les résultats de la Recherche Taboue sont meilleurs sauf dans certains cas où les résultats sont identiques comme dans les exemples la_05 et la_14, et d'autres cas où l'algorithme génétique parallèle est meilleur comme les exemples la_12 et la_15.

| **Exemple** | **Taille** | **AG Parallèle** | **Recherche Taboue** |
|---|---|---|---|
| la 01 | 10x5 | 852 | 832 |
| la 02 | 10x5 | 830 | 793 |
| la 03 | 10x5 | 790 | 747 |
| la 04 | 10x5 | 799 | 769 |
| la 05 | 10x5 | 698 | 698 |
| la 06 | 15x5 | 1 224 | 1 180 |
| la 07 | 15x5 | 1 221 | 1 091 |
| la 08 | 15x5 | 1 206 | 1 125 |
| la 09 | 15x5 | 1 262 | 1 223 |
| la 10 | 15x5 | 1 232 | 1 203 |
| la 11 | 20x5 | 1 666 | 1 584 |
| la 12 | 20x5 | 1 375 | 1 391 |
| la 13 | 20x5 | 1 611 | 1 548 |
| la 14 | 20x5 | 1 620 | 1 620 |
| la 15 | 20x5 | 1 613 | 1 650 |
| la 16 | 10x10 | 1 214 | 1 142 |
| la 17 | 10x10 | 1 050 | 1 026 |
| la 18 | 10x10 | 1 130 | 1 078 |
| la 19 | 10x10 | 1 157 | 1 093 |
| la 20 | 10x10 | 1 188 | 1 154 |
| la 21 | 15x10 | 1 767 | 1 545 |
| la 22 | 15x10 | 1 545 | 1 458 |
| la 23 | 15x10 | 1 767 | 1 611 |
| la 24 | 15x10 | 1 714 | 1 571 |
| la 25 | 15x10 | 1 711 | 1 499 |
| la 26 | 20x10 | 2 236 | 2 162 |
| la 27 | 20x10 | 2 344 | 2 175 |

| | | | |
|---|---|---|---|
| la 28 | 20x10 | 2 368 | 2 071 |
| la 29 | 20x10 | 2 242 | 2 124 |
| la 30 | 20x10 | 2 409 | 2 171 |
| la 31 | 30x10 | 3 558 | 3 167 |
| la 32 | 30x10 | 3 759 | 3 418 |
| la 33 | 30x10 | 3 424 | 3 131 |
| la 34 | 30x10 | 3 610 | 3 205 |
| la 35 | 30x10 | 3 699 | 3 311 |
| la 36 | 15x15 | 2 092 | 1 932 |
| la 37 | 15x15 | 2 200 | 2 053 |
| la 38 | 15x15 | 2 006 | 1 875 |
| la 39 | 15x15 | 2 059 | 1 950 |
| la 40 | 15x15 | 2 056 | 1 936 |

Tableau 10. Résultats des méthodes parallèles

### 3. Analyse et discussion des résultats

Pour l'analyse des résultats, nous commençons par présenter une comparaison graphique pour montrer l'apport du parallélisme à la méthode par Séparation et Evaluation SEP que nous avons développée. Les résultats sont maintenant exprimés par des diagrammes (voir Figure26).

La remarque la plus intéressante est le fait que la SEP séquentielle admet des temps de calculs plus petits que ceux de la SEP parallèle pour les petits exemples (4×5, 4×10, etc.) (voir Figure26.a). Ceci est expliqué par le fait que l'augmentation du temps de calcul de la partie parallèle est due aux différents échanges d'informations entre les postes de calcul. Par contre, le temps de transfert d'informations est négligeable lors du traitement des grands exemples (5×15, 9×5) (voir figure 26.b). Donc pour des petits exemples, il est préférable d'utiliser un calcul séquentiel.

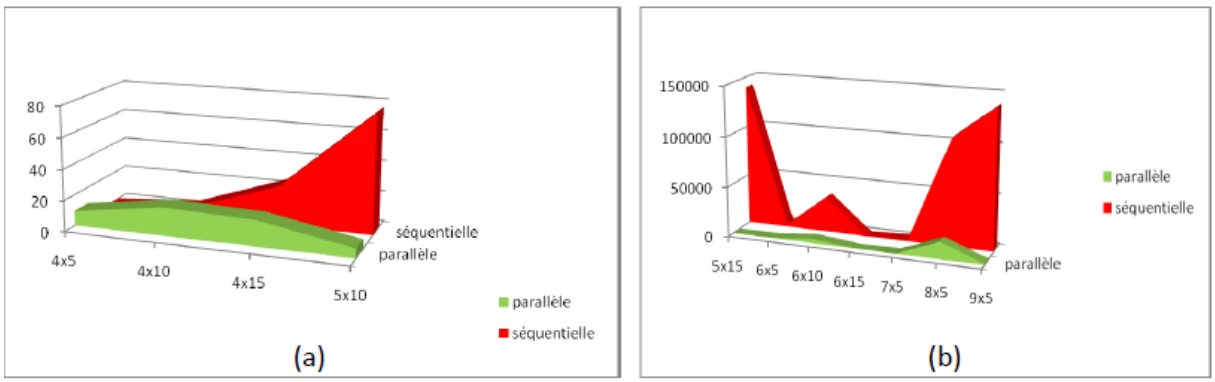


Figure 26. Exemple des temps de Réponse

Nous analysons dans cette section, l'impact de deux paramètres importants qui sont : la

taille de la population et le nombre d'itérations sur la qualité de notre Algorithme Génétique.

Dans un premier temps, nous allons voir l'effet de la taille de la population par rapport à la valeur de *Cmax*. Ce premier test est réalisé sur un exemple assez difficile qui est le la_39. La taille de la population varie de 100 à 10 000 et le nombre d'itérations est fixé à 100.

Les résultats sont donnés dans le Tableau 11, qui exprime la taille de *makespan* (*Cmax)* en fonction de la taille de la population.

| **Taille population** | **100** | **300** | **500** | **800** | **1000** | **1100** | **1500** | **2000** |
|---|---|---|---|---|---|---|---|---|
| ***Cmax*** | 2 449 | 2 264 | 2 249 | 2 152 | 2 162 | 2 126 | 2 160 | 2 135 |

Tableau 11. Résultats obtenus en variant la taille de la population pour l'AG.

La Figure27 représente la courbe correspondante au Tableau 11. Nous remarquons clairement que cette courbe se divise en deux phases:

- Une première phase qui apporte un changement positif à la qualité de la solution toute en augmentant la taille de la population.
- Une deuxième phase, qui commence à partir d'une certaine position où on constate clairement que l'augmentation de la taille de la population n'a aucun effet sur la qualité de la solution trouvée.

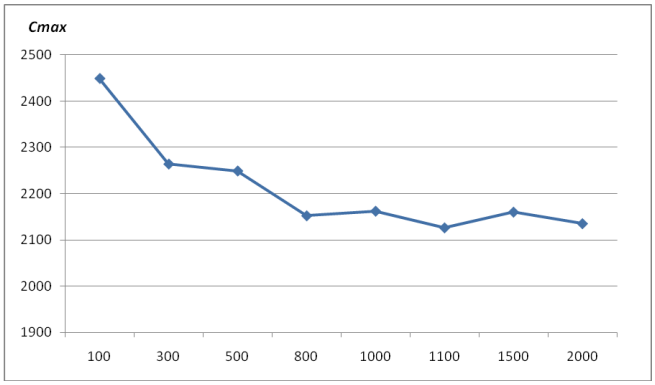


Figure 27. Taille de la population en fonction du *Cmax* pour l'AG

Comme deuxième test, nous allons voir maintenant l'effet du nombre d'itérations sur la valeur du *Cmax*. Cette fois-ci, nous faisons varier le nombre d'itérations effectuées par l'AG et nous sauvegardons les valeurs des *Cmax* retournés, comme montré dans le graphe de la Figure 28.

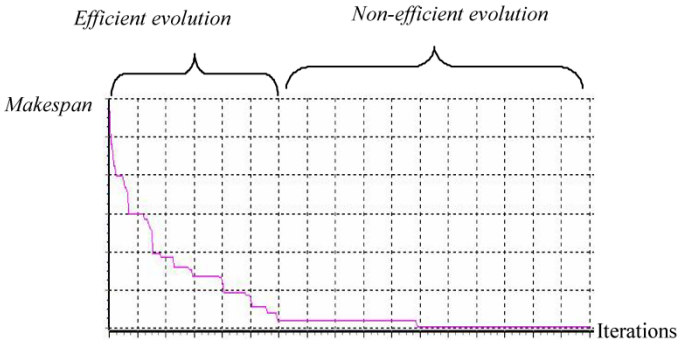


Figure 28. Le nombre d'itérations de notre AG en fonction du *Cmax*

Nous remarquons que la Figure 28 se caractérise par une décroissance rapide du *makespan,* ce qui signifie que la recherche est efficace. Dans un deuxième temps, la recherche commence à se stabiliser, et l'amélioration de la solution se fait très lentement jusqu'à stabilisation de la recherche.

Les mêmes tests ont été effectués cette fois-ci sur la deuxième méthode que nous avons développée à savoir PSO. Nous avons fait varier la taille de la population (qui représente le nombre de particules générées) en fonction des valeurs trouvées de *Cmax*. Ce test est réalisé sur un exemple assez difficile qui est : la_17. La taille de la population est variée de 100 à 10000 et le nombre d'itérations est fixé à 100. Les résultats obtenus sont résumés dans le Tableau 12.

| **Taille population** | **100** | **300** | **500** | **800** | **1 000** | **1 100** | **1 500** | **2 000** |
|---|---|---|---|---|---|---|---|---|
| Cmax | 1 195 | 1 172 | 1 140 | 1 135 | 1 138 | 1 120 | 1 087 | 1 048 |

Tableau 12. Résultats obtenus en variant la taille de la population pour PSO.

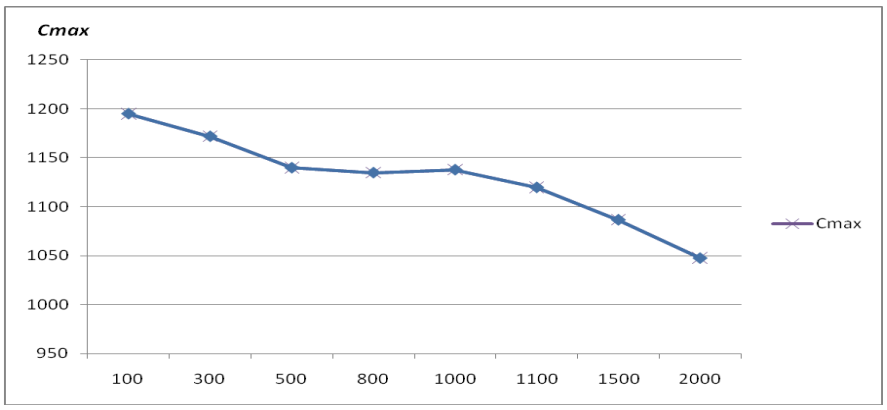


Figure 29. La taille de la population en fonction du *Cmax* pour PSO.

Nous constatons clairement pour PSO que l'augmentation du nombre de solutions générées à partir d'une particule améliore les chances de tomber sur de meilleures solutions.

## 4. Conclusion

Nous avons présenté dans ce chapitre, les résultats retournés par les différentes méthodes que nous avons développées pour la résolution du problème de job shop avec la contrainte de blocage. Les résultats obtenus sont très encourageants pour explorer plus de techniques de parallélisation qui paraissent très prometteuses. La modélisation par les graphes alternatifs a permis de diversifier le choix des codages des solutions et aussi pour développer plus d'une variante pour chaque méthode de résolution utilisée.

## Conclusion Générale

Dans cette thèse, nous avons étudié le problème d'ordonnancement avec blocage. Plusieurs modèles mathématiques ont été présentés. Parmi ces modèles, nous avons opté pour la modalisation basée sur la théorie des graphes qui utilise les graphes alternatifs.

Nous avons décrit en détail dans le chapitre 4 la méthode par séparation et évaluation que nous avons développée. Bien qu'elle atteigne les solutions optimales pour des instances de taille inférieure à 10×10, elle ne peut résoudre des instances de plus grande taille en un temps raisonnable.

Lors de la conception de cette méthode, nous avons exploité quelques techniques de base de la théorie des graphes comme : la mise à niveau, la recherche des composantes fortement connexes et la réduction des graphes, qui nous ont permis de résoudre le problème de swapping en prenant en considération les circuits de longueur nulle. Tout ceci a été fait dans la perspective d'accélérer la conquête de toutes les solutions réalisables du problème afin d'en extraire la meilleure.

Nous avons proposé aussi une nouvelle technique de parallélisation de cette méthode permettant d'effectuer un calcul parallèle au niveau des différentes branches de l'arborescence. Cette technique est implémentée sur un réseau d'ordinateurs où le nombre d'ordinateurs à intégrer n'est pas limité.

L'avantage de cette méthode est qu'elle utilise une nouvelle technique qui est l'anneau logique combiné avec la notion de jeton. Les résultats obtenus par cette technique de parallélisation sont très intéressants. En plus, nous constatons une réduction très importante du temps de calcul par rapport à la méthode séquentielle moyennant l'utilisation d'un réseau avec seulement trois (03) à quatre (04) postes.

Nous avons également adapté deux métaheuristiques, à savoir : les Algorithmes Génétiques et l'optimisation par Essaim de particules. Nous avons proposé trois types différents de codage : Le premier est un codage binaire qui utilise les arcs alternatifs, le deuxième utilise les règles de priorité pour lequel nous avons développé un algorithme de décodage spécifique qui calcule en même temps le coût de la solution trouvée et le troisième utilise l'ordre par rang. Les résultats obtenus montrent que le codage par les règles de priorité donne de meilleurs résultats en n'explorant que les solutions réalisables, à l'inverse des deux autres codes qui risquent de générer des solutions non réalisables.

Nous avons aussi utilisé trois méthodes différentes de génération des populations de solutions initiales. La première méthode utilise une génération aléatoire qui est apparue peu efficace en raison de sa génération des solutions non réalisables. La deuxième méthode exploite l'algorithme AMCC développé par Mascis & Pacciarelli (2002) [59] qui s'est avéré également peu efficace pour les grandes instances. En effet par moment, il prend des heures pour générer une seule solution. La troisième technique proposée est basée sur la méthode par séparation et évaluation que nous avons développée. Cette dernière est lancée pour un certain nombre d'itérations, ensuite elle est arrêtée pour récupérer les *n* solutions réalisables. La taille de la population initiale à prendre est égale aux *n/2* meilleures solutions obtenue. Cette méthode s'est avérée convaincante pour générer la population de départ.

L'optimisation par Essaim de particules est connue dans la littérature pour être une méthode dédiée aux problèmes d'optimisation continue. Nous avons proposé une nouvelle méthode permettant d'adapter PSO avec sa version continue à notre problème de job shop avec blocage qui est un problème discret. Ceci nous a permis d'améliorer beaucoup cette heuristique par rapport aux autres versions discrètes de PSO proposées dans la littérature.

Additivement aux autres travaux, nous avons procédé à la parallélisation de ces deux métaheuristiques et cela en utilisant cette fois ci les techniques maître/esclave, connues dans la littérature. Le principe utilisé consiste à dédier les postes clients pour le calcul de l'évaluation des solutions et le poste serveur, appelé le contrôleur, à la gestion des solutions et des autres calculs restants. Le comportement de ces deux méthodes reste moins intéressant. En effet, nous n'avons pu améliorer que quelques résultats obtenus sur des instances connues dans la littérature.

Tous ces travaux ont mis en exergue quelques axes de recherche intéressants à explorer toujours dans le cadre de la résolution du problème de Job shop avec blocage. En effet, il serait intéressant de voir l'impact de l'algorithme de parallélisation que nous avons proposé sur la méthode par séparation et évaluation de Carlier. Nous pensons quelle donnera de meilleurs résultats.

Aussi, il serait très intéressant d'affiner les bornes inférieures pour le problème du job shop avec blocage, car dans la littérature, la borne proposée par Carlier a souvent été reprise. Seulement, cette borne, proposée pour la résolution du problème de job shop simple, est basée sur la résolution des sous problèmes à une machines, Or nous avons remarqué lors des tests numériques que cette borne n'est pas du tout efficace pour le job shop avec blocage.

[1] Adams J, Balas E & Zawaku D (1988) The shifting bottleneck procedure for job shop scheduling. Management Science, vol.34, pp.391-401.

[2] AitZai A & Boudhar M (2013) Parallel Branch-and-Bound and Parallel PSO algorithms for job shop scheduling problem with blocking, Int. J. Operational Research, vol. 16, n°. 2, pp.xx–xx (Accepted, IJOR 2013 (forthcoming)).

[3] AitZai A, Benmedjdoub B & Boudhar M (2012) A branch and bound and parallel genetic algorithm for the job shop scheduling problem with blocking, Int. J. Operational Research, vol. 14, n°. 3, pp.343–365.

[4] AitZai A, Bentaher A, Bennoui H, & Boudhar M (2010) Séparation et Evaluation pour le Problème d'Ordonnancement avec Blocage, Colloque sur l'Optimisation et les Systèmes d'Information (COSI'2010) Ouargla, Algérie (du 18 au 20 Avril 2010).

[5] AitZai A & Boudhar M (2011) Parallel PSO for the Scheduling problem with Blocking, International Symposium on Operational Research (ISOR 2011) Algiers, Algeria (du 30 Mai au 02 Juin 2011).

[6] AitZai A, Dabah A, Amarni M & Boudhar M (2011) Etude Comparative entre AG et PSO appliquées au problème d'Ordonnancement avec Blocage, International Conference on Computer Science and Engineering (CSE-11) Istanbul, Turkey (du 19 au 21 Décembre 2011).

[7] Akers SB & Friedman J (1955) A non-numerical approach to production scheduling problems. Operations Research, vol.3, pp.429–442.

[8] Ben Mabrouk B, Hasni H & Mahjoub Z (2009) On a parallel genetic–tabu search based algorithm for solving the graph colouring problem, European Journal of Operational Research, vol. 197, pp.1192–1201.

[9] Benmejdoub B (2010) Job shop avec contrainte de blocage et de sans attente, Mémoire de Magister, Faculté de Mathématiques, USTHB (Encadré par M. Boudhar USTHB).

[10] Bertolissi E (2000) Heuristic algorithm for scheduling in the no-wait fow-hop. Journal of Materials Processing Technology, vol. 107, pp.459-465.

[11] Blazewicz J, Ecker KH, Pesch E, Schmidt G & Weglarz J (1996) Scheduling computer and manufacturing processes. Springer Verlag, Berlin.

[12] Boudhar M, Notions de Complexité (seconde version corrigée), Première année de Post-Graduation : Recherche opérationnelle USTHB 2004-2005.

[13] Boukala MC (1991) Parallélisation des algorithmes branch-and-Bound, Thèse de Magister, Université des Sciences et de la Technologie Houari Boumediene.

[14] Bouquard JL, Billaut JC, Kubzin MA & Strusevich VA (2005) Two-machine flowshop scheduling problems with no-wait jobs. Operations Research Letters vol.33, pp.255-262.

[15] Brucker P (2007) Scheduling Algorithms, Fifth Edition, Springer.

[16] Campbell HG, Dudek RA & Smith ML (1970) A heuristic algorithm for the n job m machine sequencing problem. Management Science vol. 16, pp.630–637.

[17] Caraffa V, Ianes S, Bagchi TP & Sriskandarajah C (2001) Minimizing makespan in a blocking flowshop using genetic algorithms. International Journal of Production Economics, vol. 70, pp.101-115.

[18] Carlier J (1982) The one-machine sequencing problem. European Journal of Operational Research vol. 11, pp.42–47.

[19] Carlier J & Pinson E (1989) An algorithm for solving the job-shop problem. Management Science vol. 35, n°2, pp. 164–176.

[20] Carlier J & Pinson E (1990) A practical use of Jackson's preemptive schedule for solving the job shop problem, Annals of Operations Research, vol. 26, pp.269–287.

[21] Carlier J & Pinson E (1994) Adjustment of heads and tails for the job-shop problem, European Journal of Operational Research, vol. 78, pp.238–251.

[22] Chang J, Gong D & Xiao-ping MA (2007) A Heuristic Genetic Algorithm for No-Wait Flowshop Scheduling Problem. J China Univ Mining & Technol, vol.17, n° 4, pp. 582-586.

[23] Charon I, Germa A & Hudry O (1996) Méthodes d'optimisation combinatoire, Masson, Paris.

[24] Dannenbring DG (1977) An evaluation of flow-shop sequencing heuristics. Management Science, vol. 23, n°. 1, pp. 1174-1182

[25] Dorndorf U & Pesch E (1995) Evolution based learning in a job shop environment, Computers and Operations Research, Vol. 22, pp.25–40.

[26] Fahmy SA, ElMekkawy TY & Balakrishnan S (2010) Mathematical formulations for scheduling in manufacturing cells with limited capacity buffers, Int. J. Operational Research, Vol. 7, No.4, pp.463–486.

[27] Fisher H & Thompson G L (1963) Probabilistic learning combinations of local job-shop scheduling rules, in: J.F. Muth and G.L. Thompson (eds.), Industrial Scheduling, Prentice-Hall, Englewood Cliffs, pp.225–251.

[28] Flamini M & Pacciarelli D (2008) Real time management of a metro rail terminus, European Journal of Operational Research Vol. 189, n° 3, pp. 746-761

[29] French S (1982) Sequencing and scheduling: an introduction to the mathematics of the job- shop. Ellis Horwood Limited, New York.

[30] Garey MR, Johnson DS & Sethi R (1976) The complexity of flowshop and job shop scheduling, Mathematics of operational research, vol. 1, n° 2, pp.117- 129

[31] Garey MR & Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, ISBN 0-7167-1045-5.

[32] Ghedjati F (2010) Variety of meta-heuristics based on genetic algorithms to solve a generalized job-shop problem, IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC 2010), Istanbul, Turkey, October 2010.

[33] Gilmor P & Gomory R (1964) Sequencing a One State-Variable Machine: A Solvable Case of the Travelling Salesman Problem, Operation res.vol. 12, pp. 655-679.

[34] Goldberg D (1989) Genetic Algorithm in Search Optimization and machine Learning. Addison-Wesley, Reading Mass.

[35] Gondran M & Minoux M (1988) Graphes et Algorithmes, Editions Eyrolles, Paris.

[36] Grabowski J, Nowicki E & Zdrzalka S (1986) A block approach for single-machine scheduling with release dates and due dates, European Journal of Operational Research, vol. 26, pp. 278–285.

[37] Grabowski J & Pempera J (2007) The permutation flow shop problem with blocking: A tabu search approach. Omega, vol. 35, pp. 302-311.

[38] Graham R, Lawler E, Lenstra JK & RinnooyKan A (1979) Optimization and approximation in deterministic sequenecing and scheduling theory: a survey. Annals of discrete mathematics, vol. 5, pp. 287-326.

[39] Gröflin H & Klinkert A (2005) A Tabu Search for the Generalized Blocking Job Shop MIC2005: The Sixth Metaheuristics International Conference.

[40] Gröflin H & Klinkert A (2009) A new neighbourhood and tabu search for the blocking job shop, Discrete Applied Mathematics, vol. 157. n°. 17, pp.3643–3655.

[41] Guo Q (2007) Parallel genetic algorithms for the solution of inverse heat conduction problems, International Journal of Computer Mathematics vol. 84, n° 2, pp.241-249.

[42] Hall NG & Sriskandarajah C (1996) A survey of machine scheduling problems with blocking and no-wait process, Operations Research, Vol. 44, No. 3, pp.510-525

[43] Holland JH (1975) Adaptation in Natural and Artificial Systems. Ed. The University of Michigan Press.

[44] Hundal T S & Rajgopal J (1988) An extension of Palmer's heuristic for the flow shop scheduling problem, International journal of production research, vol. 26, n°. 6, pp. 1119-1124.

[45] Jackson JR (1956) An extension of Johnson's results on job lot scheduling. Naval Research Logistics Quaterly, vol. 3, pp.201-203.

[46] Johnson SM (1954) Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quaterly, vol. 1, pp.61-68.

[47] Jong KD (1980) Adaptive system design: A genetic approach, IEEE Transaction on Systems, Man and Cybernetics vol. 10, n° 9, pp.566 -574.

[48] Kirkpatrick S, Gelatt Jr & Vecchi CD (1983) Optimization by simulated annealing, Science, vol. 220, pp. 671–680.

[49] Krishnakumar K & Goldberg D (1992) Control system optimisation using genetic algorithm, Journal of Guidance, Control and Dynamics, vol. 15, n° 3, pp.735–740.

[50] Lawler EL (1978) Sequencing jobs to minimize total weighted completion time subject to precedence constraints, Annals of Discrete Mathematics, vol. 2, pp.75–90.

[51] Lawrence S (1984) Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement), Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania.

[52] Lee WC, Shiau YR, Chen SK & Wu CC (2010) A two-machine flow shop scheduling problem with deteriorating jobs and blocking, Int. J. Production Economics, vol. 124, pp.188–197.

[53] Lenstra JK (1977) Sequencing by enumerative methods, Mathematical Centre Tracts 69, Mathematisch Centrum, Amsterdam.

[54] Lenstra JK & RinnooyKan (1979) Computational complexity of discrete optimization problems, Annals of discrete mathematics, vol. 4, pp. 121-140.

[55] Ling-huey SU & Yuan-Yu L (2008) The two-machine flowshop no-wait scheduling problem with a single server to minimize the total completion time, Computers & Operations Research vol. 35, pp. 2952 – 2963.

[56] Liu B, Wang L & Yi-Hui J (2008) An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. Computers & Operations Research, vol. 35, pp. 2791-2806.

[57] Liu SQ & Kozan E (2009) Scheduling a flow shop with combined buffer conditions, Int. J. Production Economics, vol. 117, pp.371–380.

[58] Martinez S, Dauzère-Pérès S, Guéret C, Mati Y & Sauer N (2006) Complexity of flowshop scheduling problems with a new blocking constraint, European Journal of Operational Research, vol. 169, pp. 855-864.

[59] Mascis A & Pacciarelli D (2002) Job-shop scheduling with blocking and no-wait constraints, European Journal of Operational Research, vol. 143, pp. 498–517.

[60] Mati Y, Rezg N & Xie X (2001) Scheduling Problem of Job-Shop with Blocking: A Taboo Search Approach, MIC'2001 - 4th Metaheuristics International Conference.

[61] Maurice Clerc M & Kennedy J (2002) The Particle Swarm : Explosion, Stability and Convergence in a Multi-Dimensional Complex Space. In Proceedings of the IEEE Transactions on Evolutionary Computation.

[62] McCormick ST, Pinedo ML, Shenker S & Wolf B (1989) Sequencing in an assembly line with blocking to minimize cycle time, Operations Research, vol. 37, n° 6, pp. 925–936.

[63] Moyson F & Manderick B (1988) The collective behaviour of Ants: an Example of self Organization in Massive Parallelisme. Spring Symposium on Parallel Models of Intelegence, Standford, Californie.

[64] Muth JF & Thompson GL (1963) Industrial Scheduling, Wiley, New York.

[65] Nawaz M, Enscore E & Ham I (1983) A heuristic algorithm for the m-machine, n-job flow shop sequencing problem, OMEGA, International Journal of Management Science vol. 11, pp. 91–95.

[66] Ng WC & Mak KL (2005) Yard crane scheduling in port container terminals, Applied Mathematical Modelling vol. 29, pp. 263–276

[67] Ng CT, Allahverdi A, Al-Anzi FS & Edwin Cheng TC (2007) The three-machine flowshop scheduling problem to minimise maximum lateness with separate setup times, Int. J. Operational Research, vol. 2, No.2, pp.135–155.

[68] Omkumar M, Shahabudeen P, Gughan S & Azad A (2009) GA based static scheduling of multilevel assembly job shops, Int. J. Operational Research, vol. 4, No.2, pp.197–213.

[69] Palmer DS (1965) Sequencing jobs through a multi-stage process in the minimum total time- a quick method of obtaining a near optimum, Operational Research Society, vol. 16, n° 1, pp.101-107.

[70] Perregaard M & Clausen J (1998) Parallel branch-and-bound methods for the job-shop scheduling problem, Annals of Operations Research, vol. 83, pp.137–160.

[71] Pham D & Klinkert A (2008) Surgical case scheduling as a generalized job shop scheduling problem, European Journal of Operational Research, vol. 185, pp.1011–1025.

[72] Pinedo M (1995) Scheduling: Theory, Algorithms and Systems. Prentice Hall, Englewood Cliffs N.J.

[73] Pinedo M (2002) Scheduling: theory, algorithms and systems. 2nd ed. New York: Prentice Hall, 586 p.

[74] Ribas I, Companys R & Tort-Martorell X (2011) An iterated greedy algorithm for the flow shop scheduling problem with blocking, Omega, vol. 39, pp.293–301.

[75] Rinnooy Kan AHG (1976) Machine Scheduling Problems: Classification, Complexity and Computations. The Hague, Martinus Nijhoff, ISBN 9024718481.

[76] Rogers RV & White KP (1991) Algebraic, mathematical programming and network models of the deterministic job-shop scheduling problem, IEEE Transactions on Systems, Man, and Cybernetics, vol. 21, n° 3, pp. 693–697.

[77] Ronconi DP and Henriques L (2009) Some heuristic algorithms for total tardiness minimization in a flowshop with blocking, Omega, vol. 37, pp.272–281.

[78] Ronconi DP (2004) A note on constructive heuristics for the flowshop problem with blocking. International Journal of Production Economics, vol. 87, pp.39-48.

[79] Roy B & Sussman B (1964) Les problèmes d'ordonnancement avec contraintes disjonctives. Note DS No. 9 bis, SEMA, Paris.

[80] Sotskov YN & Shakhlevich NV (1995) NP-hardness of shop scheduling problems with three jobs, Discrete Applied mathematics, vol. 59, n° 3, pp. 237–266.

[81] Soukhal A, Oulamara A & Martineau P (2005) Complexity of flow shop scheduling problems with transportation constraints, European Journal of Operational Research, vol. 161, pp.32-41.

[82] Soukhal A & Martineau P (2005) Resolution of a scheduling problem in a flowshop robotic cell, European Journal of Operational Research, vol. 161, pp. 62-72.

[83] Kravchenko S (1998) A polynomial algorithm for a two-machine no-wait job-shop scheduling problem, European Journal of Operational Research vol. 106, pp. 101-107.

[84] Tanaev VS, Gordon VS & Shafransky YM (1994) Scheduling theory, Single-stage systems, vol. 1, Kluwer Academic Publishers, Dordrecht.

[85] Tanaev VS, Sotskov YN & Strusevich VA (1994) Scheduling theory, Multi-stage systems, vol. 285 of Mathematics and its Applications, Kluwer Academic Publishers Group, Dordrecht. Translated and revised from the 1989 Russian original by the authors.

[86] Thornton HW & Hunsucker JL (2004) A new heuristic for minimal makespan in flow shops with multiple processors and no intermediate storage, European Journal of Operational Research, vol. 152, pp. 96-114

[87] Tseng LY & Lin YT (2010) A genetic local search algorithm for minimizing total flow time in the permutation flow shop scheduling problem, Int. J. Production Economics, vol. 127, pp.121–128.

[88] Wang X & Tang L (2009) A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers, Computers & Operations Research, vol. 36, n° 3, pp. 907–918.

[89] Wang L, Zhang L & Zheng D (2006) An effective hybrid genetic algorithm for flow shop scheduling with limited buffers, Computers & Operations Research, vol. 33, pp.2960-2971.

[90] Wang L & Zheng DZ (2003) An effective hybrid heuristic for flow shop scheduling, International Journal of Advanced Manufacturing Technology, vol. 21, pp.38–44.

[91] Wang L, Pan QK, Suganthan PN, Wang WH & Wang YM (2010) A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems, Computers and Operations Research, vol. 37, pp.509–520.

[92] Yagouni M (1999) L'approche neuro-mimétique et le recuit simulé pour la résolution des problèmes de l'optimisation combinatoire. Thèse de magister USTHB, Algérie.

[93] Yang RL (2000) Convergence of the Simulated Annealing Algorithm for Continuous Global Optimization. Journal of Optimization Theory and Applications, vol. 104, n° 3, pp. 691–716.

[94] Yeh J, Wu T & Chang J (2007) Parallel genetic algorithms for product configuration management on PC c1uster systems, International Journal of Advanced Manufacturing Technology vol. 31, n° 11, pp. 1233-1242.

[95] Zhong W, Zhang J & Che W (2007) A novel discrete particle swarm optimization to solve traveling salesman problem, In: Proceedings of the 2007 IEEE Congress on Evolutionary Computation, Singapore, September 2007, pp. 3283-3287, SBN: 978-1-4244-1340-9, IEEE

[96] Zhu J, Li X, Wang Q (2008) Complete local search with limited memory algorithm for job shops to minimize makespan, Key laboratory of Computer Network and Information Integration (Southeast University) Ministry of Education, 210096 Nanjing, PR China.