REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOHAMED BOUDIAF - M'SILA
FACULTE DES MATHEMATIQUES ET DE L'INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

MEMOIRE de fin d'étude

Présenté pour l'obtention du diplôme de MASTER

Domaine : Mathématiques et Informatique

Filière : Informatique

Spécialité : Informatique Décisionnelle et Optimisation

Par : YAHIAOUI Khadidja
BOUSBA Achwak

SUJET

Un problème Flow Shop à deux machines avec des temps de latence

Soutenu publiquement le : 17/ 09/2020 devant le jury composé de :

Dr. AKHROUF Samir	**Université de M'sila**	**Président**
Dr. MOUHOUB Nasser Eddine	**Université de M'sila**	**Rapporteur**
Dr. BOUDAA Abd El Ghani	**Université de M'sila**	**Examinateur**

Promotion :2019/2020

Remerciements

On remercie Dieu le tout puissant et le Miséricordieux, qui nous a donné non seulement le courage mais aussi la force et la patience pour l'accomplissement de ce travail. Et a exaucé nos prières en nous donnant l'occasion de vous les présenter aujourd'hui.

Nous remercions en particulier ***Mr. MOUHOUB NASSER EDDINE,*** *pour l'honneur qu'il nous a fait de bien vouloir nous encadrer, et pour les précieux conseils donnés et prestation de documentations nécessaires lors de la réalisation de ce travail.*

Et Nous remercions aussi tous les professeurs qui nous a aidés des cinq années d'étude au l'université Mohamed BOUDIAF M'sila.

Nous également remercions notre amie et collègue KACIMI EL HASSANI Aicha qui nous a beaucoup aidés.

On adresse nos remerciements aux membres de jury pour avoir accepté de nous prêter de leur attention et évaluer notre travail.

Dédicaces

En signe de respect et de reconnaissance nous dédions ce modeste travail à :

Nos parents :

YAHIAOUI Mustapha, OUAHABI Hassiba

BOUSBA Fayçal, BOUCETTA Fatiha

La source de notre force et confiance, qui ne cessent de nous donner de l'amour et tout le nécessaire jusqu'à ce que nous soyons capables d'atteindre où nous en sommes aujourd'hui.

Que Dieu vous protège et que le succès soit toujours à notre portée afin que nous puissions vous combler de bonheur.

Nos frères et sœurs :

Mourad, Abdou, Amina, Randa, Hafsa BOUSBA

Et ma chère amie et my twin Racha BOUSBAA

Amina, Oumaima, Hhadjer YAHIAOUI

À chaque instant de l'examen, avec leur soutien moral. Nous vous souhaitons un avenir plein de joie, de bonheur, de réussite et de sérénité.

Nous vous exprimons à travers ce travail nos sentiments de fraternité et d'amour.

***À notre encadreur** : M. Mouhoub Nasser Eddine, qui nous a aidés à mener à bien ce travail, auquel nous devons beaucoup de remerciements, un grand respect, une pleine appréciation et une grande appréciation.*

Tables des matières (Sommaire)

Introduction générale..... 1

Chapitre I. La fonction ordonnancement

1. Introduction..... 4

2. Les éléments fondamentaux..... 4

2.1. Les tâches..... 4

2.2. Les ressources..... 5

2.3. Les contraintes..... 6

2.4. Les objectifs..... 6

3. Les caractéristiques générales des ordonnancements..... 7

3.1. Ordonnancements semi-actifs et actifs..... 7

3.2. Ordonnancements statiques et dynamiques..... 7

3.3. Ordonnancements admissibles..... 7

3.4. Ordonnancements sans retard..... 7

3.5. Ordonnancements préemptifs et non préemptifs..... 7

4. L'Ordonnancement dans les ateliers..... 7

4.1. Les problèmes à une machine..... 8

4.2. Les problèmes à machines parallèles..... 8

4.3. Les problèmes multi-machines..... 8

4.3.1. Flow Shop (atelier à cheminement unique) 8

4.3.2. Job shop (atelier à cheminements multiples) 9

4.3.3. Open shop (atelier à cheminements libres) 9

4.3.4. Notion de flexibilité 9

4.3.4.1. Flow Shop hybride 10

4.3.4.2. Job shop hybride 10

5. Formalisation des problèmes d'ordonnancements (Classification et notation) 10

5.1. Le champ α..... 11

5.2. Le champ β..... 11

5.3. Le champ γ..... 12

6. Modélisation..... 13
6.1. Modélisation graphique..... 13
6.2. Modélisation analytique..... 14
7. Représentation des solutions..... 14
7.1. Diagramme de Gantt..... 14
8. Conclusion..... 15
Chapitre II. Les méthodes de résolution
1. Introduction..... 17
2. Les méthodes exactes..... 17
2.1. Méthode de Séparation et Evaluation (Branch and Bound) 17
2.3. La Programmation linéaire..... 19
2.2. Programmation dynamique..... 20
3. Les Méthodes approchées..... 20
3.1. Les Heuristiques..... 21
3.2. Les Méta-Heuristiques..... 22
3.2.1. Les Méta-heuristiques à solution unique..... 22
3.2.1.1. Le Recuit Simulé..... 23
3.2.1.2. Recherche tabou..... 24
3.2.2. Les Méta-heuristiques à base de population..... 25
3.2.2.1. Les Algorithmes Génétiques..... 26
3.2.2.2. L'optimisation par colonie de fourmis(ACO) 26
3.3. Les algorithmes hybrides..... 27
3.3.1. Hybridation séquentielle..... 27
3.3.2. Hybridation parallèle synchrone..... 28
3.3.3. Hybridation parallèle asynchrone..... 28
4. Conclusion..... 29
Chapitre III. Résolution du problème Flow Shop à deux machines avec des temps de latence
1. Introduction..... 31
2. Le problème de flow shop..... 31

2.1. Les données d'un problème flow-shop..... 32

2.2. Propriétés du problème de flow shop..... 33

3. Un problème Flow Shop à deux machines avec des temps de latence..... 33

4. Approche heuristique pour le problème Flow Shop à deux machines avec des temps de latence..... 35

4.1. Algorithme de Johnson sans temps de latence..... 35

4.2. Algorithme de Johnson modifié avec des temps de latence..... 36

5. Approche exacte pour le problème Flow Shop à deux machines avec des temps de latence..... 37

5.1. Les bornes inférieures (Lowler Bounds) 37

5.1.1. La Première borne inférieure (LB1) 37

5.1.2. La Deuxième borne inférieure (LB2) 38

5.1.3. La Troisième borne inférieure (LB3) 39

5.1.4. La quatrième borne inférieure (LB4) 40

5.2. Les bornes supérieures (Upper Bounds) 42

5.2.1. La Première borne supérieure (UB1) 42

5.2.2. La Deuxième borne supérieure (UB2) 44

5.2.3. La Troisième borne supérieure (UB3) 45

5.2.4. La Quatrième borne supérieure (UB4) 46

5.3. Illustration de l'algorithme de Branch and Bound..... 49

6. Conclusion..... 50

Chapitre IV. Conception et implémentation du problème

1. Introduction..... 52

2. Langage de programmation et environnement de développement..... 52

2.1. Le langage Java..... 52

2.2. L'environnement NetBeans..... 52

3. Le Flow Shop à deux machines avec des temps de latence..... 53

4. Les méthodes utilisées pour la résolution de notre problème..... 53

5. L'Architecture générale de la réalisation..... 53

6. Illustration de l'application..... 55

7. Quelques exemples sur l'implémentation..... 56

8. Conclusion..... 64

Conclusion générale..... 65

Liste des Figures

Figure I.1. Typologie des problèmes d'ordonnancement par les ressources..... 5
Figure I.2. La représentation de modèle à machine unique..... 8
Figure I.3. La représentation de modèle à machines parallèles..... 9
Figure I.4. La représentation de modèle Flow shop..... 9
Figure I.5. La représentation de modèle job shop..... 10
Figure I.6. La représentation de modèle Open shop..... 10
Figure I.7. La représentation d'un système Flow Shop hybride à 'k' étages..... 11
Figure I.8. La représentation d'un système job shop hybride à trois étages..... 11
Figure I.9. Diagramme de Gantt d'une solution du problème F 3|perm|Cmax..... 15
Figure II.1. Les approches de résolution des problèmes d'ordonnancement..... 17
Figure II.2. Schéma de la programmation dynamique et son principe..... 19
Figure II.3. Exemple de variables de décision..... 20
Figure II.4. Exemple d'optima local et global..... 23
Figure II.5. Hybridation séquentielle..... 28
Figure II.6. Hybridation parallèle synchrone..... 28
Figure II.7. Hybridation parallèle asynchrone..... 29
Figure III.1. Modèle de flow-shop..... 33
Figure III.2. Ordonnancement sans temps de latence..... 34
Figure III.3. Ordonnancement avec temps de latence..... 34
Figure III.4. Solution optimale de l'Exemple III.1..... 36
Figure III.5. Application de LB4 avec des temps d'exécution quelconques..... 41
Figure III.6. L'application du UB1 sur la séquence des jobs de l'exemple III.6..... 44
Figure III.7. L'application du UB2 sur la séquence des jobs de l'exemple III.7..... 45
Figure III.8. L'application du UB3 sur la séquence des jobs de l'exemple III.8..... 46
Figure III.9. Ordonnancement d'une partie de la séquence en appliquant UB4..... 48
Figure III.10. Les différents ordonnancements induits par l'application de UB4..... 48
Figure III.11. Exécution de l'algorithme Branch and Bound sur une séquence de jobs dans le cas quelconque..... 49

Figure III.12. L'arbre de recherche dans le cas quelconque..... 50
Figure IV.1. Le schéma de réalisation de problème..... 54
Figure IV.2. La page d’entrée de l’application..... 55
Figure IV.3. La page qui permet de choisir le nombre des jobs..... 55
Figure IV.4. La page de l'interface..... 56
Figure IV.5. L’interface de l’implémentation de problème..... 56
Figure IV.6. Diagramme de Gantt de la machine 1..... 57
Figure IV.7. Diagramme de Gantt de la machine 2..... 57
Figure IV.8. L'application de l'algorithme de Johnson et l'obtention de temps de latence..... 57
Figure IV.9. Diagramme de Gantt de Johnson modifié de la machine 1..... 58
Figure IV.10. Diagramme de Gantt de Johnson modifié de la machine 2..... 58
Figure IV.11. Les résultats de Johnson modifié..... 59
Figure IV.12. Les résultats de LB (Lowler Bound) de Branch and Bound..... 59
Figure IV.13. Les résultats de UB (Upper Bound) de Branch and Bound..... 60
Figure IV.14. Les résultats finals et le Makespan optimal..... 60

Liste des Tables

Table I.1. Classification de Graham..... 12
Table I.2. Quelques valeurs du champ α1..... 12
Table I.3. Quelques valeurs du champ β..... 13
Table I.4. Quelques valeurs du champ γ..... 13
Table I.5. Temps d'exécution du problème m=3 et n=3..... 15
Tableau III.1. Temps d'exécution du problème n =6 et m =2..... 34
Tableau III.2. Temps de latence de jobs..... 34
Tableau III.3. Temps d'exécution du problème n =4 et m =2 de l'Exemple III.1..... 36
Tableau III.4. Temps d'exécution du problème n =4 et m =2 avec des temps de latences..... 36
Tableau III.5. Les nouveaux temps d'exécution..... 37
Tableau III.6. L'application de LB1 sur l'ExempleIII.2..... 38
Tableau III.7. L'application de LB2 sur l'ExempleIII.3..... 39
Tableau III.8. L'application de LB3 sur l'ExempleIII.4..... 40
Tableau III.9. Les temps d'exécution et de latence de l'Exemple III.5..... 41
Tableau III.10. Application de LB4 sur l'Exemple III.5..... 42
Tableau III.11. Les temps d'exécution et de latence de l'Exemple III.6..... 43
Tableau III.12. L'application de UB1 sur l'Exemple III.6..... 43
Tableau III.13. L'application de UB2 sur l'Exemple III.7..... 45
Tableau III.14. L'application de UB3 sur l'Exemple III.8..... 46
Tableau III.15. Les temps d'exécution et de latence de l'Exemple III.9..... 47
Tableau III.16. Application de UB4 sur l'Exemple III.9 dans le cas quelconque..... 48
Tableau III.17. Les temps d'exécution et de latence d'un problème de Branch and Bound…. 49
Tableau IV.1. Temps d'exécution du l'Exemple IV.1..... 56
Tableau IV.2. Temps d'exécution du l'Exemple IV.2..... 60
Tableau IV.3. Temps d'exécution du l'Exemple IV.3..... 61
Tableau IV.4. Temps d'exécution du l'Exemple IV.4..... 62
Tableau IV.5. Temps d'exécution du l'Exemple IV.5..... 62
Tableau IV.6. Temps d'exécution du l'Exemple IV.6..... 63
Tableau IV.7. Les makespans générés par les deux algorithmes et le Makespan optimal..... 63

Liste des Algorithmes

Algorithme 1 : Pseudo code de la méthode du Recuit Simulé..... 24

Algorithme 2 : Pseudo code de la recherche tabou..... 25

Algorithme 3 : Structure générale d'un algorithme génétique..... 26

Algorithme 4 : le schéma général de l'algorithme de colonies de fourmis pour le problème du voyageur de commerce (TSP) 27

Algorithme 5 : L'algorithme de Johnson pour le problème de flow-shop à deux machines..... 35

Algorithme 6 : Algorithme de la deuxième borne inférieure LB2 dans le cas quelconque..... 38

Algorithme 7 : Algorithme de la troisième borne inférieure LB3 dans le cas quelconque..... 40

Algorithme 8 : Algorithme de vérification des temps de latence..... 41

Algorithme 9 : Algorithme de la première borne supérieure UB1 pour le cas quelconque..... 43

Algorithme 10 : Algorithme de la deuxième borne supérieure UB2 pour le cas quelconque... 44

Algorithme 11 : Algorithme de la troisième borne supérieure UB3 pour le cas quelconque.... 46

Algorithme 12 : Algorithme d'adaptation NEH pour le calcul de UB4 pour le cas quelconque..... 47

Introduction générale

L'ordonnancement consiste à organiser, dans le temps, la réalisation des tâches (jobs, travaux) compte tenu des contraintes de temps et de ressources, afin d'optimiser un ou plusieurs objectifs. Ainsi, une tâche est une entité (opération ou ensemble d'opérations) élémentaire localisé dans le temps par une date de début et une date de fin, dont la réalisation est caractérisée par une durée positive qui consomme une ressource. Cette dernière est un moyen technique ou humain destiné à être utilisé pour la réalisation d'une tâche et est disponible en quantité limitée. L'évaluation d'une solution d'ordonnancement se fait par rapport à un ou plusieurs objectifs de performance.

Parmi les premiers problèmes d'ordonnancement étudiés, le problème des ateliers sériels *(flow-shop)*. Johnson, en 1954, a été le premier à avoir traité le problème de *flow-shop* à deux machines. Nous pouvons aussi trouver des problèmes de *job-shop, d'open-shop* ou encore des problèmes dits *flexibles*, dépendamment de la conception de l'atelier. Depuis, l'expérience a montré qu'il existe un gouffre entre la théorie de l'ordonnancement et ce qui se passe réellement dans les centres de productions ou les prestations de services.

Un problème Flow Shop stipule que tout travail visite chaque machine de l'atelier dans un ordre qui est le même pour tous les travaux (flux unidirectionnel). On vise à réduire au minimum le temps total d'exécution de tous les jobs, ainsi appelé Makespan C_{max}.

Parmi les contraintes que la théorie d'ordonnancement n'a pas considéré jusqu'à un passé récent, nous pouvons citer les temps de latence des travaux, correspondant aux différents temps nécessaires devant s'écouler entre la fin d'une opération et le début de la prochaine opération d'un même job. Les temps de latence peuvent correspondre par exemple aux temps de transport des jobs à travers les ressources. Ces temps peuvent dans certains cas prendre des proportions importantes qu'une entreprise ne doit en aucun cas ignorer.

Et c'est dans cette optique que nous nous intéressons, dans ce mémoire, plus particulièrement, au problème Flow-Shop à deux machines avec des temps de latence. Notre objectif est de trouver une séquence appropriée de tâches en fonction des temps de latence, de manière à minimiser le makespan. La prise en compte des temps de latence a rendu notre problème NP-difficile.

Toutefois, plusieurs méthodes peuvent être utilisées pour résoudre ce problème. En effet, nous pouvons trouver des méthodes exactes et des méthodes approchées. Et de ce principe on va essayer de mettre en œuvre l'algorithme de Johnson, tenter de le modifier, ainsi que l'algorithme de Séparation et Evaluation « Branch and Bound » pour résoudre ce type de problèmes.

Ce mémoire comporte quatre chapitres dont nous vous présentons une brève description comme suite :

Le premier chapitre présente les notions et les éléments fondamentaux, ainsi que les critères relatifs aux problèmes d'ordonnancement.

Dans le deuxième chapitre nous décrivons les méthodes de résolution d'un problème d'ordonnancement qui analysent les deux approches de résolution, à savoir les méthodes exactes et les méthodes approchées.

Dans le troisième chapitre, on abordera le problème d'ordonnancement flow shop, suivi d'une description de la résolution de problème à étudier. Par une approche heuristique, l'algorithme de Johnson et une approche exacte, l'algorithme de Branch and Bound.

Le quatrième et le dernier chapitre portera sur l'implémentation de notre application ainsi que l'élaboration des tests expérimentaux.

Nous finirons notre travail par une conclusion générale qui présente un résumé de ce qu'a été étudié dans ce mémoire, les résultats obtenus et le travail qui reste à faire pour l'accomplissement de cette étude.

Chapitre I

« La fonction Ordonnancement »

1. Introduction

L'ordonnancement consiste à organiser, dans le temps, la réalisation des tâches compte tenu des contraintes de temps et de ressources, afin d'optimiser un ou plusieurs objectifs.

La résolution d'un problème d'ordonnancement doit concilier deux objectifs. L'aspect statique consiste à générer un plan de réalisation des travaux sur la base de données prévisionnelles. L'aspect dynamique consiste à prendre des décisions en temps réel compte tenu de l'état des ressources et de l'avancement dans le temps des différentes tâches.

2. Les éléments fondamentaux

Dans un problème d'ordonnancement interviennent quatre éléments fondamentales : les tâches (travaux ou jobs), les ressources, les contraintes et les objectifs.

2.1. Les tâches

Une tâche est une entité (opération ou ensemble d'opérations) élémentaire localisé dans le temps par une date de début *ti* (Start time) et une date de fin *ci* (completion time), dont la réalisation est caractérisée par une durée positive *pi* (processing time) telle que $p_i = c_i - t_i$.

On distingue deux types de tâches :

- Les tâches préemptives (morcelable) dont l'exécution peut être divisée en plusieurs intervalles temporels (peut être exécutée par morceaux).
- Les tâches indivisibles qui sont exécutées en une seule fois et ne peuvent pas être interrompues avant qu'elles ne soient complétement achevées [1].

Une tâche est caractérisée par :

- La durée opératoire de la tâche i sur la machine j : (p_{ij}).

- La date de disponibilité de la tâche i : (r_i).

- La date de début d'exécution de la tâche i :(s_i).

- La date de fin d'exécution de la tâche i :(c_i).

- La date d'achèvement souhaitée de la tâche i :(d_i).

- Le facteur de priorité ou poids de la tâche i : (w_i).

- Le retard algébrique de la tâche i : ($L_i = c_i - d_i$).

- Le retard vrai de la tâche i : ($T_i = \max(c_i - d_i, 0)$).

- L'indicateur de retard de la tâche i : ($U_i = 1$, si $T_i > 0$, $U_i = 0$, sinon).

2.2. Les ressources

Une ressource est un moyen nécessaire humain ou technique utilisé pour exécuter une opération. Dans un atelier, on distingue plusieurs types de ressources.

1. Selon leurs disponibilités au cours du temps, on trouve :

- Les ressources renouvelables : Une ressource est renouvelable si, après avoir été allouée à une tâche, elle redevient disponible pour les autres. Les ressources renouvelables usuelles sont les machines, les processeurs, les fichiers, le personnel, etc.
- Les ressources consommables (non renouvelables) : une ressource est consommable si, après avoir été allouée à une tâche, elle n'est plus disponible pour les tâches restant à exécuter. C'est le cas pour l'argent, les matières premières, etc.
- Les ressources doublement contraintes, ces ressources combinent les contraintes liées aux deux catégories précédentes. Leur utilisation instantanée et leur consommation globale sont toutes les deux limitées. C'est le cas des ressources d'énergie (pétrole, électricité, etc.).

2. Selon leurs capacités, on trouve :

- Les ressources disjonctives (ou non-partageables), il s'agit des ressources qui ne peuvent exécuter qu'une seule opération à la fois c'est le cas par exemple de machine-outil ou robot manipulateur.
- Les ressources cumulatives (ou partageables), il s'agit des ressources qui peuvent être utilisé par plusieurs opérations simultanément (équipes d'ouvriers, poste de travail, … etc.). La Figure I.1 illustre une représentation schématique de la Typologie des problèmes d'ordonnancement par les ressources [2].

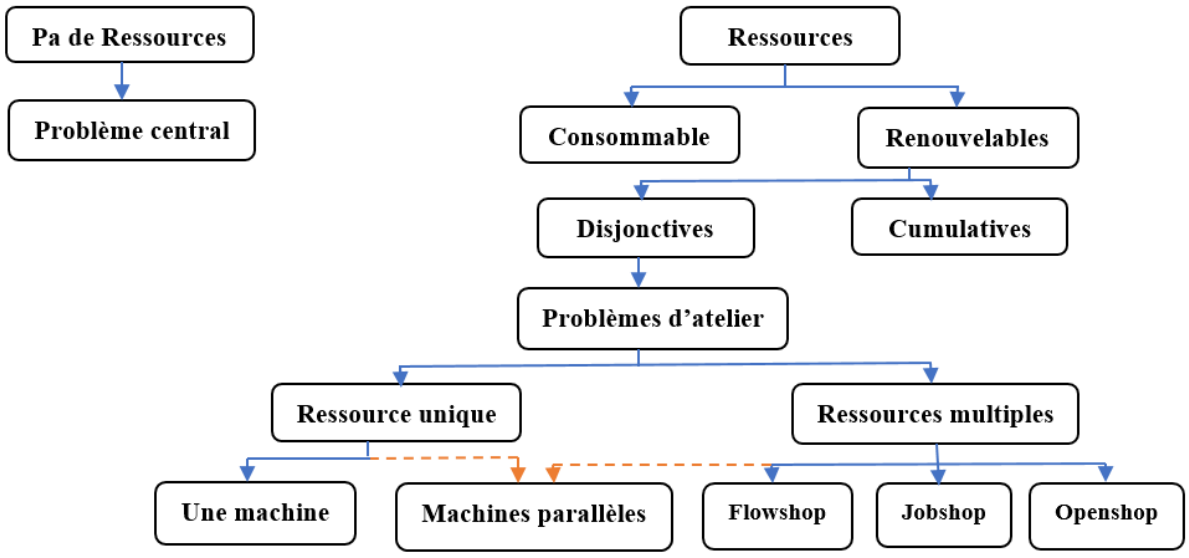

Figure I.1. Typologie des problèmes d'ordonnancement par les ressources

2.3. Les contraintes

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

- Des contraintes temporelles :
    - Les contraintes de temps alloué, issues généralement d'impératifs de gestion et relatives aux dates limites des tâches (délais de livraison, disponibilité des approvisionnements) ou à la durée totale d'un projet ;
    - Les contraintes de cohérence technologique, ou contraintes de gammes, qui décrivent des relations d'ordre entre les différentes tâches ;
- Des contraintes de ressources :
    - Les contraintes d'utilisation de ressources qui expriment la nature et la quantité des moyens utilisés par les tâches, ainsi que les caractéristiques d'utilisation de ces moyens ;
    - Les contraintes de disponibilité des ressources qui précisent la nature et la quantité des moyens disponibles au cours du temps. Toutes ces contraintes peuvent être formalisées sur la base des distances entre débuts de tâches ou potentiels [A].

2.4. Les objectifs

Dans la résolution d'un problème d'ordonnancement, on peut choisir entre deux grands types de stratégies, visant respectivement à l'optimalité des solutions, ou plus simplement à leur admissibilité.

L'approche par optimisation suppose que les solutions candidates à un problème puissent être ordonnées de manière rationnelle selon un ou plusieurs critères d'évaluation numériques, construits sur la base d'indicateurs de performances. On cherchera donc à minimiser ou maximiser de tels critères. On note par exemple ceux.

- Liés au temps :
    - Le temps total d'exécution ou le temps moyen d'achèvement d'un ensemble de tâches
    - Le stock d'en-cours de traitement
    - Différents retards (maximum, moyen, somme, nombre, etc.) ou avances par rapport aux dates limites fixées ;
- Liés aux ressources :
    - La quantité totale ou pondérée de ressources nécessaires pour réaliser un ensemble de tâches ;

- La charge de chaque ressource ;
- Liés à une énergie ou un débit ;
- Liés aux coûts de lancement, de production, de transport, etc., mais aussi aux revenus, aux retours d'investissements.

3. Les caractéristiques générales des ordonnancements

3.1. Ordonnancements semi-actifs et actifs

Un ordonnancement est dit actif s'il est impossible d'avancer le début d'exécution d'une opération sans devoir retarder une autre tâche ou violer une contrainte (de précédence, date de début au plus tôt, ...). Et en dit qu'un ordonnancement est semi-actif si aucune tâche ne peut être exécutée plus tôt sans changer l'ordre d'exécution sur les ressources ou violer une contrainte (de précédence, date de début au plus tôt, ...).

3.2. Ordonnancements statiques et dynamiques

Si l'ensemble des informations nécessaires à la résolution d'un ordonnancement est fixé au départ (ensembles des tâches, des contraintes, des ressources, etc.), on est alors devant un problème d'ordonnancement *statique*. Il y a une nuance entre la solution proposée qui est, généralement accompagnée d'un plan prévisionnel d'exécution des tâches, et l'exécution réelle de ces tâches. Si le plan n'est pas respecté et les objectifs sont modifiés, on est devant un problème d'ordonnancement *dynamique* qui nécessite une résolution d'une série de problèmes statiques et chaque étape doit débuter par une prise d'informations permettant d'actualiser le modèle à résoudre.

3.3. Ordonnancements admissibles

Un ordonnancement est dit admissible s'il respecte toutes les contraintes du problème (dates de début, dates de fin, précédence, contraintes de ressources, etc.).

3.4. Ordonnancements sans retard

Dans un ordonnancement sans retard, on doit exécuter la tâche qui est en attente à condition que la ressource soit disponible.

3.5. Ordonnancements préemptifs et non préemptifs

Selon les problèmes, les tâches peuvent être exécutées sans interruption, c'est-à-dire si on commence l'exécution d'une tâche elle n'est pas interrompu jusqu'à son achèvement. On parle alors d'un ordonnancement *préemptif*. Par contre, si les tâches sont exécutées par morceaux, l'ordonnancement est appelé non *préemptif*.

4. L'Ordonnancement dans les ateliers

Une classification très répandue des ateliers, du point de vue ordonnancement, est basée sur les différentes configurations des machines (le nombre et la nature) ainsi que l'ordre d'enchaînement des opérations (gamme de fabrication). Les modèles les plus connus sont ceux d'une machine unique, de machines parallèles, d'un atelier à cheminement unique ou d'un atelier à cheminement multiple, Nous expliquons ces différents types comme suit :

4.1. Les problèmes à une machine

Dans les problèmes d'atelier à une machine (*single machine problem*), l'ensemble des tâches à réaliser est fait par une seule machine. Les tâches alors sont composées d'une seule opération qui nécessite la même machine [B] (*Figure I.2)*.

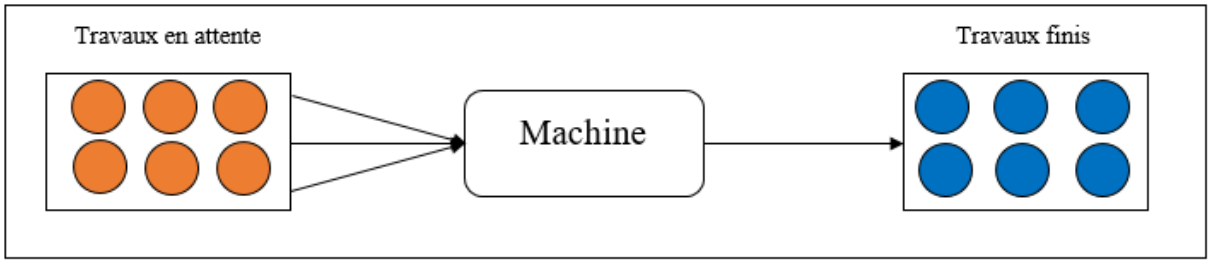

Figure I.2. La représentation de modèle à machine unique.

4.2. Les problèmes à machines parallèles

Les problèmes d'atelier à machines parallèles (*parallel machine problem*) sont une généralisation des problèmes à une machine. Ce type d'atelier se caractérise par le fait que chaque opération peut être réalisée par n'importe quelle machine, disposée en parallèle, mais n'en nécessite qu'une seul. Le problème d'ordonnancement consiste donc à déterminer l'affectation des opérations aux machines puis le séquencement de ces opérations sur chaque machine. Dans le dernier cas, il est possible de distinguer trois classes de machines :

- Machines parallèles identiques P (*identical parallel machines*) : la durée d'exécution des opérations est la même sur toutes les machines.
- Machines parallèles uniformes Q (*uniform parallel machines*) : la durée d'exécution des opérations varie uniformément en fonction de la performance de la machine choisie.
- Machines parallèles indépendantes ou dédiées R (*unrelated parallel machines*) : les durées opératoires dépendent complètement des machines utilisées [3] (*Figure I.3)*.

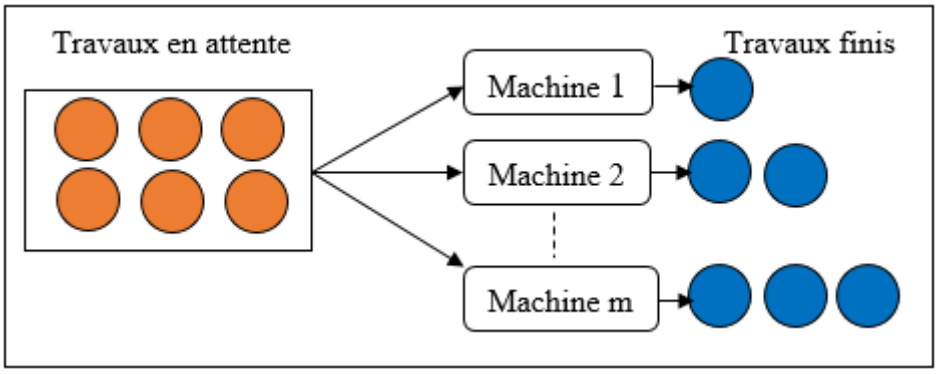

Figure I.3. La représentation de modèle à machines parallèles.

4.3. Les problèmes multi-machines

Les problèmes d'atelier sont dits Les problèmes multi-machines du fait de la nécessité du passage de chaque job sur deux ou plusieurs machines dédiées. Suivant le mode de passage des opérations sur les différentes machines, trois types d'ateliers sont distingués à savoir :

4.3.1. Flow Shop (atelier à cheminement unique)

Dans les ateliers de type Flow-Shop il s'agit d'un ensemble de m machines disposées en séries. Toutes les opérations de tous les jobs passent par les machines dans le même ordre (flot unidirectionnel) toutes les files d'attente des machines opèrent selon la règle FIFO (First In First Out). Un cas particulier important est celui d'un Flow-Shop de permutation, le Flow-Shop est dit de permutation s'il existe une contrainte selon laquelle la séquence des opérations des différents jobs est la même sur chaque machine *(Figure I.4).* Ce type de problème sera détaillé dans le chapitre 3.

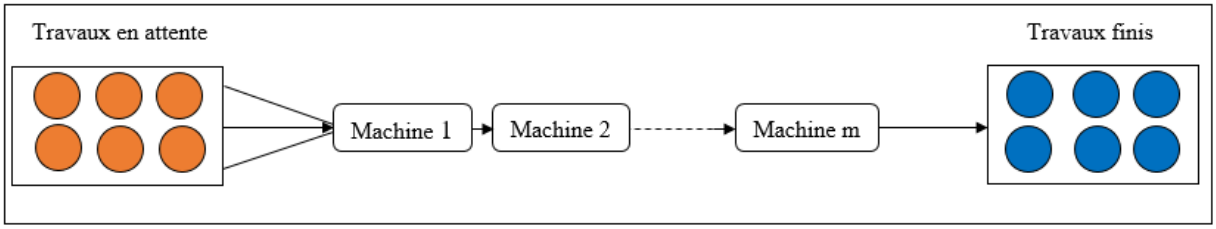

Figure I.4. La représentation de modèle Flow Shop.

4.3.2. Job shop (atelier à cheminements multiples)

Dans les ateliers de type Job Shop, chaque opération passe sur les machines dans un ordre fixé, mais à la différence du Flow Shop, cet ordre peut être différent pour chaque job (flot multidirectionnel). Il s'agit dans ce cas de déterminer les dates de passage sur différentes ressources d'ordres de fabrication ayant des trajets différents dans l'atelier. Ces ordres de fabrication partageant des ressources communes (*Figure I.5).*

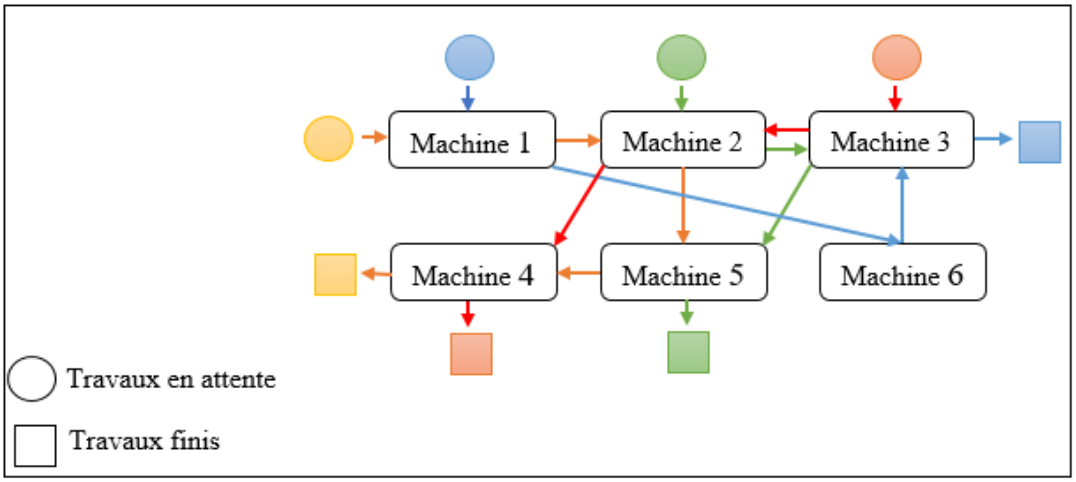

Figure I.5. La représentation de modèle job shop.

4.3.3. Open shop (atelier à cheminements libres)

Dans le modèle *d'open shop,* l'ordre de passage des jobs sur les machines n'est pas connu à l'avance (libre). Cet ordre est déterminé lors de la construction de la solution. Chaque job peut avoir son propre ordre de passage sur toutes les machines. Le fait qu'il n'y ait pas d'ordre prédéterminé rend la résolution du problème d'ordonnancement de ce type plus complexe, mais offre cependant des degrés de liberté intéressants. À la Figure I.6, nous avons un ensemble de trois jobs et un ensemble de trois machines. À droite de la figure nous pouvons remarquer que chaque job a suivi un ordre de passage différent sur les trois machines.

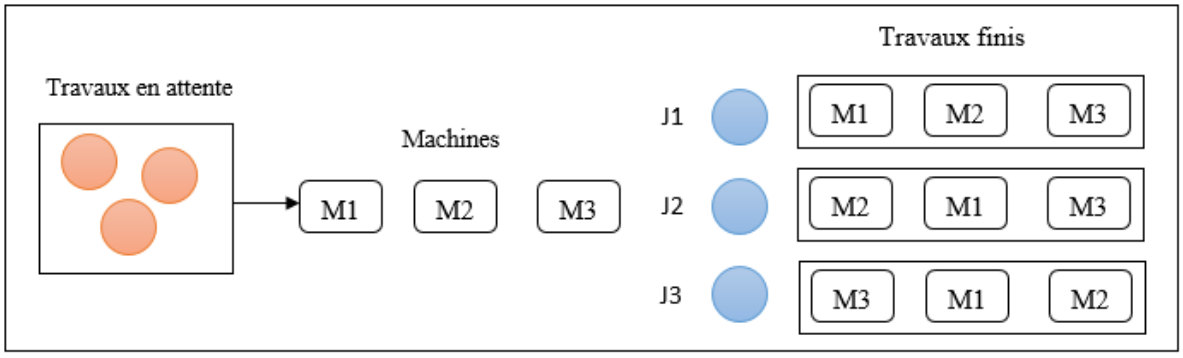

Figure I.6. La représentation de modèle Open shop.

4.4. Notion de flexibilité

Une façon d'augmenter la productivité d'un atelier est d'augmenter sa flexibilité. Pour cela, nous pouvons multiplier le nombre de machines qui réalisent la même tâche. Le modèle résultant est connu dans la littérature comme un Flow Shop hybride ou un job shop hybride.

4.4.1. Flow Shop hybride

Le Flow Shop hybride est une généralisation du Flow Shop classique et de machines parallèles. Dans ce cas l'atelier est constitué d'un certain nombre d'étages en série, chaque étage

étant constitué de plusieurs machines en parallèle. Ce type d'ateliers est également appelé « atelier à cheminement unique avec machines en exemplaires multiples » (Figure I.7).

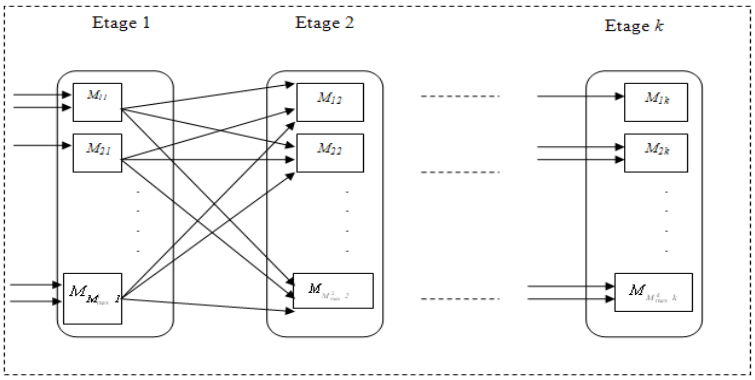

Figure I.7. La représentation d'un système Flow Shop hybride à 'k' étages.

4.4.2. Job shop hybride

Les problèmes de type job-shop hybride sont une extension de deux problèmes d'ordonnancement : le problème de job-shop et le problème d'ordonnancement des machines parallèles. Le job shop hybride peut être vu comme un problème de job-Shop qui possède une ou plusieurs machines parallèles par étage. La machine nécessaire pour exécuter une opération n'est pas connue a priori. Par conséquent, le problème se compose de deux parties dont la première est de trouver la séquence des jobs sur les étages, la deuxième partie est de trouver la machine nécessaire à l'exécution d'un job tout en respectant les contraintes du job-shop. Un exemple de ce problème à *m* étages et trois machines maximum par étage, est donné dans la Figure I.8 [4].

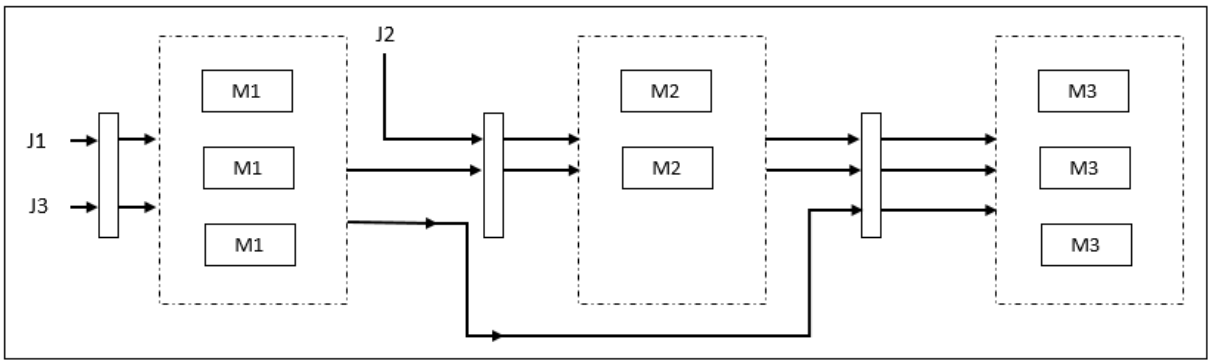

Figure I.8. La représentation d'un système job shop hybride à trois étages.

5. Formalisation des problèmes d'ordonnancements (Classification et notation)

Vu la très grande variété de problèmes d'ordonnancement, Graham Suggérez une méthode de classification qui permet en effet de caractériser un problème d'ordonnancement de manière précise. Cette méthode est basée une notation à trois champs distincts notée **α/β/γ**.

5.1. ***Le champ α*** **:** désigne l'environnement machine. Il se compose généralement de deux sous champs α1 et α2, le premier indiquant la nature du problème étudié (job shop, flow shop, etc.), et le second précisant le nombre de machines.

5.2. ***Le champ β*** **:** désigne les types de contraintes liées à l'exécution des tâches. Ce champ peut être vide Ø là où aucune contrainte n'est imposée et peut contenir une concaténation de 1 à *k* sous-champs comme indiqué dans le Tableau I.1.

5.3. ***Le champ γ*** **:** désigne-le (ou les) critère(s) à optimiser (la fonction), Il contient dans la plupart des cas un seul champ.

Les différentes notations utilisées pour les valeurs des champs *α*, *β* et figurent généralement sous forme d'abréviations (tableau I.1) ayant chacune d'elles un sens particulier. La description de ces abréviations est décrite dans les tableaux I.2, I.3 et I.4 respectivement.

Champ		Sous champs	Notation
α	(α_1)	Type de machine	{Ø,1, P, Q, R, F, J, O, FH, JF, OG}
	(α_2)	Nombre de machines	{Ø, m}
	(β_1)	Mode d'exécution des jobs	{Ø, pmtn}
	(β_2)	Ressources supplémentaires	{Ø, res}
	(β_3)	Relation de précédence	{Ø, prec, tree, cachins}
β	(β_4)	Date de disponibilité	{Ø, r_i}
	(β_5)	Durées opératoires	{Ø, p_i=p}
	(β_6)	Dates d'échéance	{Ø, d_i}
	(β_7)	Propriété d'attente	{Ø, nwt}
γ		///////////	{C_{max}, $\sum w_iC_i$, L_{max}, T_{max}, $\sum w_iT_i$, $\sum U_i$, $\sum w_iU_i$}

Table I.1. Classification de Graham.

Notation	**Description**
1	Problème à une seule machine
P	Problème à machines parallèles identiques
Q	Problème à machines parallèles uniformes
R	Problème à machines parallèles indépendantes
F	Flow-Shop
J	Job-Shop
O	Open-Shop
FH	Flow-Shop Hybrid
JH	Job-Shop Flexible
OG	Open-Shop Généralisé

Table I.2. Quelques valeurs du champ α1.

Exemple :

- $F2||C_{max}$ dénote un problème d'ordonnancement d'un atelier de type Flow Shop à deux machines avec minimisation de Makespan C_{max}. L'absence de valeurs dans le champ β.

Notaion	Description
pmtn	La préemption des opérations est autorisée
prec	Existence des contraintes de précédence entre les opérations
res	L'opération nécessite l'emploi d'une ou plusieurs ressources supplémentaires
nwt	Les opérations de chaque job doivent se succéder sans attente
$p_i = p$	Les temps d'exécution des tâches sont identiques et égaux à p
r_i	Une date de début au plus tôt est associée à chaque job *i*
d_i	Une date d'échéance est associée à chaque job *i*

Table I.3. Quelques valeurs du champ β.

Valeur	**Description (la fonction objectif à minimiser)**
C_{max}	La durée totale de l'ordonnancement (makespan)
L_{max}	Le plus grand retard algébrique
T_{max}	Le plus grand retard vrai
$\sum U_i$	Le nombre de travaux en retard
$\sum w_i C_i$	La somme [pondéré] des dates de fin des tâches
$\sum w_i T_i$	La somme [pondéré] des retards Σ
$\sum w_i U_i$	Le nombre [pondéré] des tâches en retard

Table I.4. Quelques valeurs du champ γ.

6. Modélisation

La modélisation est une étape très importante dans la résolution d'un problème. Elle est caractérisée par une écriture simplifiée de toutes les données de problème tout en utilisant un formalisme bien adapté. Principalement, il existe deux types de modélisation pour les problèmes d'ordonnancement. La modélisation graphique sous forme de graphe et la modélisation analytique sous forme de programme mathématique.

6.1. Modélisation graphique

Cette méthode de modélisation présente un caractère visuel facilitant la vérification de la cohérence du problème considéré et l'interprétation des solutions, ce qui lui a permettre d'être la méthode la plus utilisé dans la littérature. Dans cette modélisation, deux types de graphes sont utilisés, le graphe PERT et le graphe des potentiels.

6.1.1. La méthode PERT (Program Evaluation and Review Technique)

Est une méthode Américaine élaborée pour résoudre les problèmes d'ordonnancement. Elle consiste à associer à un problème d'ordonnancement un graphe constitué d'un ensemble de nœuds reliés entre eux par des arcs où :

- Chaque nœud correspond à un instant de déclenchement d'exécution d'une tâche représentée par l'arc sortant du nœud et également à un instant d'achèvement d'une tâche représentée par l'arc entrant au nœud.
- Sur les arcs, nous trouvons généralement les durées d'exécution associées aux tâches. Dans le cas où la valeur d'un arc est égale à 0 cela présente une contrainte de précédence.

6.1.2. La méthode des potentiels

A été développée vers la fin des années 50 parallèlement à la méthode PERT. Elle est appelée également la méthode MPM (méthode des potentiels Metra) ou encore méthode des potentiels – tâches. Elle se base aussi sur une modélisation par graphe constitué d'un ensemble de nœuds et d'arcs. Les nœuds du graphe représentent les tâches composant les travaux à réaliser, auxquels s'ajoutent deux sommets fictifs appelés « source » et « puits » correspondant respectivement au début et à la fin des travaux. Ainsi, les arcs peuvent être de deux types :

- Arcs conjonctifs connectent deux tâches consécutives d'un même travail (contraintes de précédence). Ces arcs sont pondérés par la durée opératoire de la tâche, exceptés les arcs de la source qui sont pondérés par 0.
- Arcs disjonctifs connectent deux tâches appartenant à des travaux différents qui utilisent la même machine (contraintes de ressources).

6.2. Modélisation analytique

Un problème d'ordonnancement peut également être modélisé sous une forme analytique. Cette modélisation, couramment utilisée, est souvent sous forme de programme mathématique dont les données, les contraintes et la fonction d'évaluation des critères sont écrites sous forme d'équations et d'inéquations mathématiques. Cette modélisation permet, non seulement de mettre en évidence l'objectif et les différentes contraintes du problème, mais également de le résoudre.

7. Représentation des solutions

Afin de visualiser une solution d'un problème d'ordonnancement, nous utilisons couramment une représentation graphique appelée diagramme de Gantt.

7.1. Diagramme de Gantt

Ce diagramme constitue un formalisme graphique qui a été mis au point par Henry Gantt en 1910. Dans le cas d'un problème d'atelier, le diagramme est formé de deux axes orthogonaux et peut avoir deux représentations :

- La représentation « Jobs » : l'axe horizontal représente le temps et l'axe vertical représente les jobs. Pour chaque job, nous dessinons des rectangles représentant l'ensemble des machines utilisées dans le temps pour la réalisation du job.
- La représentation « Machines » : l'axe horizontal représente le temps et l'axe vertical représente les machines. Pour chaque machine, nous représentons l'ensemble des opérations effectuées dans le temps par des barres ayant des longueurs proportionnelles à leurs durées opératoires.

Cette représentation permet de visualiser l'occupation des machines, l'enchaînement des opérations sur celles-ci, les dates de début et de fin de chaque opération ainsi que le temps d'inactivité des machines.

À titre d'illustration, une solution réalisable du problème $F3|perm|C_{max}$ avec les données fournies dans le tableau I.5 est donnée par la figure I.9. Nous utilisons la représentation (b), pour présenter notre solution dans ce qui suit.

J_i	P_{i1}	P_{i2}	P_{i3}
J_1	2	1	3
J_2	3	5	2
J_3	4	2	1

Table I.5. Temps d'exécution du problème m=3 et n=3.

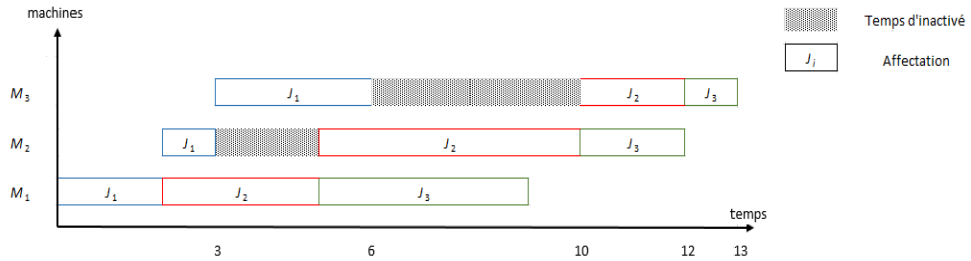

Figure I.9. Diagramme de Gantt d'une solution du problème F3|perm|C_{max}.

8. Conclusion

Dans ce chapitre nous avons résumé les différentes notions liées à l'ordonnancement on a mentionné ses différents éléments et ses caractéristiques générales, Et nous avons parlé brièvement sur l'ordonnancement dans les ateliers et nous expliqué ses différents types, Nous avons présenté les différentes notations, et classification des problèmes d'ordonnancement, ainsi que la modélisation et les représentations des ordonnancements. Nous avons exposé à la fin de ce chapitre la complexité des problèmes d'optimisation.

Chapitre II

« Les Méthodes de Résolution »

1. Introduction

Les méthodes exploitées dans la résolution des problèmes d'ordonnancement sont généralement celles de la discipline de la recherche opérationnelle. Cette dernière regroupe différentes méthodes et techniques pour la résolution des problèmes d'optimisation combinatoire. Dans ce chapitre, nous allons passer en revue les méthodes les plus connues en les répertoriant en méthodes exactes et méthodes approchées. Pour chaque méthode, l'idée générale est rapportée. Plus de détails ne sont donnés que pour la méthode que nous utilisons dans notre étude. Et la résolution des problèmes d'ordonnancement de type Flow-Shop avec ou sans temps de latence. Pour commencer nous allons présenter un schéma descriptif, Comme illustré par la Figure II.1.

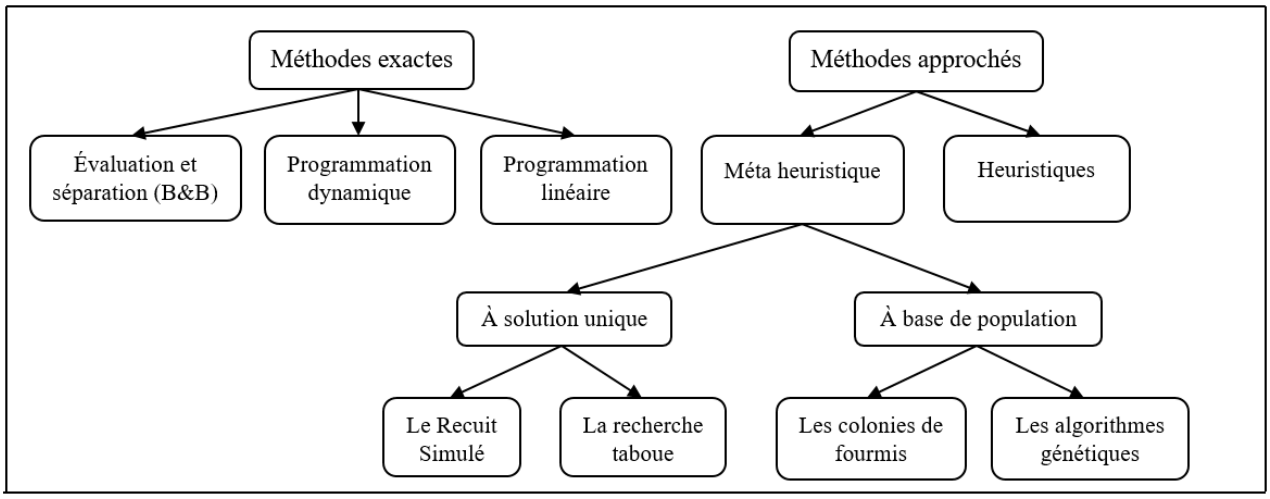

Figure II.1. Les approches de résolution des problèmes d'ordonnancement.

2. Les méthodes exactes

Les méthodes exactes sont celles qui nous permettent de fournir une solution optimale pour les problèmes d'optimisation combinatoire, grâce à une exploration intelligente de l'espace des solutions. Ces méthodes se caractérisent par un temps de calcul souvent exponentiel, ce qui explique qu'elles ne sont utilisables que pour des problèmes de petites tailles. Les méthodes les plus couramment utilisées sont : la méthode de Séparation et Evaluation, la programmation dynamique et la programmation linéaire. Ces méthodes sont décrites dans les sous-sections suivantes.

2.1. Méthode de Séparation et Evaluation (Branch and Bound)

Parmi les méthodes de résolution des problèmes d'ordonnancement, nous avons cité plus haut la méthode de Séparation et Evaluation, plus connue sous son appellation anglaise Branch and Bound.

La résolution optimale de problèmes d'optimisation combinatoire *NP-difficiles* nécessite une mise en œuvre de méthodes plus complexes. Il est possible d'énumérer toutes ces solutions, et

ensuite d'en prendre la meilleure. L'inconvénient majeur de cette approche est le nombre prohibitif des solutions que cela peut générer.

La procédure par évaluation et séparation est considérée parmi les méthodes exactes les plus reconnues dans la résolution optimale des problèmes d'optimisation combinatoires. Cette méthode est basée sur une énumération implicite et intelligente de l'ensemble des solutions, mais l'analyse des propriétés du problème évite l'énumération de larges classes ne contenant pas de solution optimale. En d'autres termes, seulement les solutions potentiellement bonnes seront énumérées. Le principe de cette méthode repose comme son nom l'indique sur deux notions clé : la séparation et l'évaluation.

- **La Séparation :** La phase de séparation consiste à diviser le problème en un certain nombre de sous problèmes qui sont décomposés à leur tour selon une démarche itérative. Ce processus peut se visualiser sous la forme d'un arbre d'énumération ou les nœuds de l'arbre correspondent aux sous-ensembles et les feuilles correspondent à des solutions réalisables. Cette arborescence est appelée *l'arbre de recherche.*

- **L'Evaluation :** Pour accélérer la recherche de la solution optimale en évitant l'exploration inutile de certains nœuds, on utilise une procédure d'évaluation. Cette dernière réduit l'espace de recherche en éliminant quelques sous-ensembles qui ne contiennent pas la solution optimale. Donc on peut distinguer pendant le déroulement de l'algorithme trois types de nœuds dans l'arbre de recherche : le *nœud courant* qui est le nœud en cours d'évaluation, des *nœuds actifs* qui sont dans la liste des nœuds qui doivent être traités, et des *nœuds inactifs* qui ont été élagués au cours du calcul. Et pour appliquer cette méthode de au moins un des points suivants doit être vérifié :

  - ***Le calcul d'une borne inférieure*** : la borne inférieure permet d'éliminer certaines solutions de l'arbre de recherche afin de faciliter l'exploration du reste de l'arbre et de faire converger la recherche vers la solution optimale.

  - ***Le calcul d'une borne supérieure*** : la borne supérieure peut être une heuristique qui permet d'élaguer certaines branches de l'arbre de recherche.

  - ***Application des règles de dominance*** : dans certains cas et pour certains problèmes, il est possible d'établir des règles qui permettent d'élaguer des branches de l'arbre de recherche.

  - ***Stratégie de recherche*** : Il existe plusieurs techniques pour l'exploration de l'arbre de recherche. On peut utiliser soit une exploration en profondeur d'abord (Depth First Search -DFS), soit une exploration en largeur d'abord (Breadth First Search-BFS), soit encore une recherche qui utilise une file de priorité. Et le choix du critère d'évaluation.

Cette méthode a été introduite pour la première fois par Dantzig, Fulkerson et Johnson [5] pour la résolution du problème du voyageur de commerce. Et a été utilisée aussi pour résoudre le problème du sac à dos *(Knapsack Problem)*. Depuis, des centaines, voire des milliers de

procédures par Séparation et Evaluation ont été proposées pour des problèmes d'ordonnancement.

2.2. La Programmation linéaire

La programmation linéaire (PL) est l'une des méthodes les plus puissantes en recherche opérationnelle. Elle se base sur la modélisation mathématique du problème. Son utilisation demande que le problème posé puisse se ramener à un programme linéaire, ce dernier se compose d'une fonction linéaire à optimiser (maximiser ou minimiser) désignant l'objectif du problème et d'un ensemble d'équations et/ou d'inéquations linéaires représentant les contraintes imposées par le problème. Généralement, dans le processus de modélisation d'un problème, nous commençons par la définition des variables de décision. En ordonnancement, ces variables peuvent être des variables indexées par le temps (variables temporelles) indiquant si un job est en train d'être exécuté à un instant t, des variables de précédence indiquant si deux jobs se précèdent dans une séquence et des variables de positions indiquant si un job est en position k dans une séquence [7]. La Figure II.3 montre un exemple de ces trois variables sur une séquence de cinq jobs. Un modèle linéaire peut s'exprimer sous la forme compacte suivante :

$$\begin{cases} Max \sum_{i=1}^{n} \text{ciXi} \\ \text{Sous contraintes} \\ \sum_{i=1}^{n} \text{aiXi} \geq \text{bi} \\ \text{Xi} \geq 0 \end{cases}$$

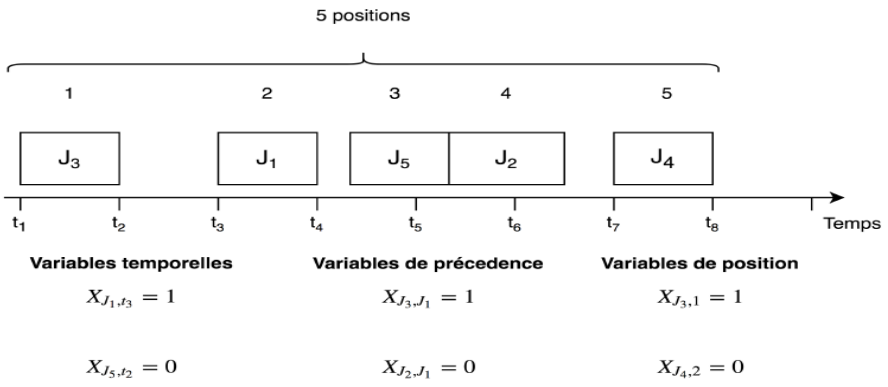

Figure II.3. Exemple de variables de décision.

Suivant l'ensemble de définition des variables, nous distinguons trois types pour cette méthode :

- Programme linéaire continu si les variables sont des nombres réels.
- Programme linéaire en nombres entiers si les variables sont des entiers.
- Programme linéaire en nombres mixtes si les variables sont des entiers et des réels.

La résolution exacte des problèmes linéaires à variables continues utilise des méthodes telles que la méthode de simplexe [8], ou la méthode des points intérieurs. Lorsqu'il y a des variables discrètes, on parle de programmes linéaires en nombre entiers (*PLNE* ou *IP*) ou encore des programmes linéaires à variables mixtes (*PLM* ou *MIP*).

2.3. La programmation dynamique

La programmation dynamique est considérée comme une approche permettant de concevoir des méthodologies robustes de résolution exacte des problèmes d'optimisation combinatoire dont la fonction objectif présente une propriété dite de décomposabilité. Cette méthode a été initiée par Richard Bellman dans les années cinquante dans le cadre d'une recherche du plus court chemin dans un graphe. Elle est conçue principalement pour la résolution de certains problèmes qui présentent certaines caractéristiques permettant leur résolution à l'aide d'une formule de récurrence. Son efficacité repose sur le principe d'optimalité, dit de Bellman « Toute politique optimale est composée de sous politiques optimales ». Alors, le concept de base de la programmation dynamique est simple, transformer la solution optimale par la somme de sous-problèmes résolus de façon optimale. Il faut donc diviser un problème donné en sous-problèmes liés par une relation de récurrence portant sur la valeur de la fonction objective.et les résoudre un par un. Ainsi, pour obtenir la solution optimale du problème, il suffit de revenir en arrière à travers l'ensemble des décisions prises et stockées à chaque résolution de sous-problème, comme le montre la Figure II.2. Cette méthode peut s'avérer couteuse en temps et en mémoire. Cependant, elle permet de fournir des algorithmes polynomiaux et pseudo-polynomiaux pour des cas particuliers [6].

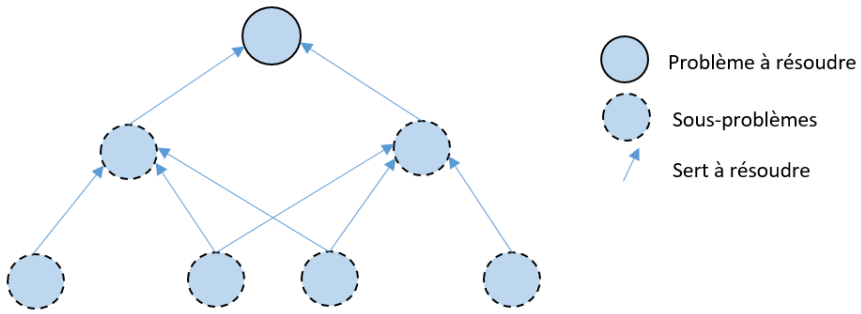

Figure II.2. Schéma de la programmation dynamique et son principe.

3. Les Méthodes approchées

Les difficultés rencontrées par les méthodes exactes pour la résolution des problèmes d'optimisation de grande taille en un temps raisonnable ont incité les chercheurs à trouver des alternatives connues sous l'appellation : méthodes approchées. Ces méthodes nous permettent

de trouver une solution acceptable en un temps de calcul raisonnable. La performance de telles méthodes est généralement donnée par l'estimation du pourcentage d'erreur entre la valeur de la solution fournie et la valeur de la solution optimale si elle est calculable. Dans les cas où la solution optimale est non calculable, il est également possible d'étudier et d'évaluer expérimentalement le comportement d'une méthode en comparant ses performances soit à celles d'autres méthodes, soit à des bornes inférieures. Lorsque ces méthodes sont conçues de manière simple, rapide et ciblée sur un problème particulier, on les appelle heuristiques. Cependant, lorsque celles-ci sont générales, adaptables et applicables à plusieurs catégories de problèmes d'optimisation combinatoire, elles portent le nom de méta-heuristiques [3].

3.1. Les Heuristiques

Les heuristiques sont des méthodes empiriques qui cherchent de bonnes solutions à un coût raisonnable sans être en mesure de garantir l'optimalité. Ces méthodes se basent sur des règles simplifiées pour optimiser un ou plusieurs critères. Leur principe général est d'intégrer des stratégies de décision pour construire une solution proche de l'optimum, tout en essayant de l'obtenir en un temps de calcul raisonnable. Parmi ces heuristiques, on peut citer :

- L'algorithme de Johnson : développée dans le cadre de la recherche d'une séquence de durée minimale dans un atelier de type Flow-Shop de permutation à deux machines.
- Les algorithmes de liste dont le principe consiste à trier la liste des tâches selon une stratégie de décision appelée « règles de priorité » telle que :
  - ***FIFO (First In First Out***) : la première tâche qui vient est la première tâche ordonnancée,
  - ***LIFO (Last In Last Out)*** : la dernière tâche qui vient est la première tâche ordonnancée,
  - ***SPT (Shortest Processing Time) :*** la tâche ayant le temps opératoire le plus court est traité en premier lieu,
  - ***LPT (Longest Processing Time)*** : la tâche ayant le temps opératoire le plus important est ordonnancé en premier lieu,
  - ***EDD (Earliest Due Date)*** : la tâche ayant la date due la plus petite est la plus prioritaire,
  - ***SRPT (Shortest Remaining Processing Time)*** : cette règle, servant à lancer la tâche ayant la plus courte durée de travail restant à exécuter, est très utilisée pour minimiser les encours et dans le cas des problèmes d'ordonnancement préemptifs,
  - ***ST (Slack Time)*** : à chaque point de décision, l'opération ayant la plus petite marge temporelle est prioritaire. Faute de disponibilité des ressources de production, cette marge peut devenir négative.
  - ***FAM (First Available Machine)*** : affecter les tâches à la première machine libre, etc.

Donc en peut résumer que les heuristiques sont des méthodes de résolution non fondées sur un modèle formel et qui fournissent rapidement (en temps polynomial) une solution réalisable, pas nécessairement optimale.

3.2. Les Méta-Heuristiques

Les méta-heuristiques ont connu un essor considérable depuis leur apparition dans les années 1970, par Osman et Laporte, et le mot méta-heuristique est dérivé de la composition de deux mots grecs Heuristique qui vient du verbe heuriskein et qui signifie ‘trouver’. Méta qui est un suffixe signifiant au-delà dans un niveau supérieur.

Les méta-heuristiques sont des techniques puissantes et généralement applicables à un grand nombre de problèmes. Formellement, une méta-heuristique fait référence à une stratégie de maître interactif qui guide et modifie les opérations heuristiques subordonnées en combinant intelligemment différents concepts pour explorer et exploiter tout l’espace de recherche. Des stratégies d’apprentissage sont utilisées pour structurer l’information afin de trouver efficacement des solutions optimales, ou proches de d’optimales [9]. Les performances de ce type de méthodes dépendent de la qualité de la solution obtenue et du temps de calcul nécessaire [3]. Les méta-heuristiques n’offrent généralement pas de garantie d’optimalité, mais leur succès est dû à la capacité de résoudre en pratiques certains problèmes difficiles. Ces méthodes ont en commun un certain nombre d’avantages [10,11] :

- Elles sont souvent inspirées par des analogies avec la physique (recuit simulé), avec la biologie (algorithmes génétiques, recherche tabou, etc.) ou avec l’éthologie (colonies de fourmis, etc.).

- Certaines de ces méthodes se caractérisent par un effet stochastique. Cet effet possède l’avantage de diversification des différentes solutions explorées au cours du processus de calcul.

- Elles sont souvent d’origine discrète à l’exception de certaines comme les essaims de particules.

- La combinaison de deux méthodes approchées ou plus est généralement permise.

Elles partagent aussi les mêmes inconvénients suivants :

- Difficulté de réglage des paramètres.

- Temps de calcul parfois élevé.

Les méta-heuristiques sont habituellement classées en fonction du nombre de solutions qu’elles manipulent : les méta-heuristiques à solution unique telles que le recuit simulé et la recherche tabou, et les méta-heuristiques à population de solutions telles que les algorithmes génétiques, les colonies de fourmis, etc. [3]

3.2.1. Les Méta-heuristiques à solution unique

Les méta-heuristiques à solution unique appelées aussi méthodes de recherche, se basent sur la notion de voisinage qui représente un ensemble de solutions obtenues à partir d’une solution

donnée en effectuant un certain nombre de transformations. Le but de ces transformations locales est d'explorer le voisinage de la solution courante afin d'améliorer progressivement sa qualité au cours des différentes itérations. Comme le montre la figure II.4, ces techniques tendent à trouver l'optimum global (G) sans être piégé par les optimums locaux (L).

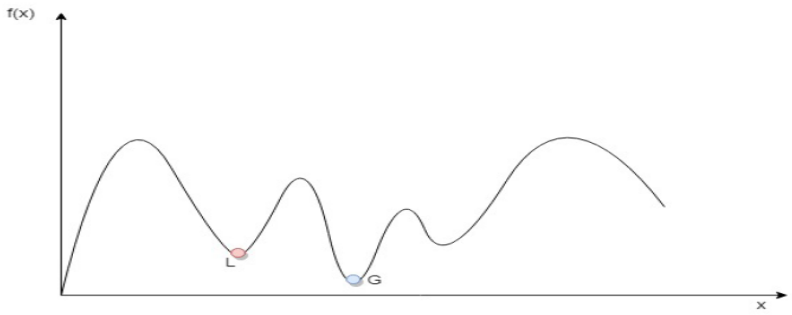

Figure II.4. Exemple d'optima local et global.

Il existe un grand nombre de méthodes à recherche locale. Les plus répondues sont présentées par la suite :

3.2.1.1. Le Recuit Simulé

Cette méthode est inspirée d'un processus utilisé en métallurgie dont le principe consiste à porter un métal à une température très élevée, puis à le refroidir très lentement. Cela permet la solidification du métal dans une structure d'énergie minimale.

En optimisation, le principe de cette méthode repose sur une procédure itérative qui cherche des solutions de coûts plus faibles tout en acceptant de manière contrôlée des solutions qui dégradent la fonction objective. À chaque itération, une nouvelle solution s ′ est choisie parmi l'ensemble des voisins N (s) de la solution courante s. Les nouvelles solutions sont toujours acceptées si le coût de la fonction objective diminue, c'est-à-dire si $f(s') \_ f(s)$ avec $f(s)$ le coût de la fonction objectif associée à la solution s. Dans le cas contraire, la solution s' est acceptée ou rejetée. Soit $\Delta f = f(s') - f(s)$ la différence entre les fonctions objectives de la nouvelle solution $f(s')$ et la solution courante $f(s)$.

L'acceptation d'une solution est déterminée de la façon suivante : on choisit un nombre aléatoire, noté q, de l'intervalle [0,1] que l'on compare avec la probabilité $prob(\Delta f, T) = \exp(-\Delta f = k * T)$ où T est un paramètre de contrôle appelé la température et k est la constante dite de Boltzmann.

Si $q \leq prob(\Delta f, T)$ alors la nouvelle solution est adoptée comme solution courante. Autrement, la nouvelle solution est rejetée.

La probabilité $prob(\Delta f, T)$ dépend essentiellement de deux facteurs :

- De l'importance de la dégradation Δf, les dégradations plus faibles sont plus facilement acceptées.

- D'un paramètre de contrôle *T* (la température), une température élevée correspond à une probabilité plus grande d'accepter des dégradations.

L'acceptation des solutions moins bonnes permet à l'algorithme de sortir d'optima locaux. Cette procédure est répétée pendant un nombre spécifié de cycles jusqu'à atteindre un quasi-équilibre.

La température est ensuite diminuée et une nouvelle itération est exécutée. En pratique, l'algorithme s'arrête lorsque soit la température est plus petite qu'une certaine valeur T_f appelée la température finale, soit lorsque aucune configuration voisine n'a été acceptée pendant un certain nombre d'itérations [12,3]. Ce principe peut se résumer par l'algorithme 1.

Algorithme 1 : Pseudo code de la méthode du Recuit Simulé.

```
Début
    s← s0                                   →s0 est la solution initiale
    T←T0                                    →T0 est la température
    initiale du système
    Tant que T > Tf
        Générer une solution s′ ∈ N (s ) aléatoirement
        Calculer Δf = f (s′) - f (s)
        Si Δf ≤ 0 alors
            s←s′
        Sinon
            Calculer p r o b (Δf, T) exp←(-Δf /T)
            Générer q uniformément dans l'intervalle [0,1]
            Si q < p r o b (Δf, T) alors
                s←s′
            Sinon
                s′ est rejetée
            Fin si
        Fin si
        T←T×Θ                               →0 < Θ < 1 coefficient de
        refroidissement
    Fin Tant que
Fin
```

Parmi les difficultés de cette méthode est la détermination de la valeur initiale de la température *T*0 et le coefficient de refroidissement de la température Θ. Le réglage de ces paramètres est assez délicat et repose sur des essais. L'avantage est la flexibilité concernant les évolutions du problème et la facilité de mise en œuvre.

3.2.1.2. Recherche tabou

La recherche tabou a été formalisée par Glover en 1986, son idée de base consiste à introduire la notion de mémoire dans la politique d'exploration de voisinage. Cette idée est inspirée de la mémoire humaine.

La recherche tabou est une procédure itérative qui, partant d'une solution initiale (cette solution peut être construite par une heuristique ou générée aléatoirement) tente de converger vers la meilleur solution en exécutant à chaque itération un mouvement dans l'espace de recherche. Chaque itération consiste d'abord à engendrer un ensemble de solutions voisines de la solution courante pour ensuite en choisir la meilleure.

La recherche taboue utilise une mémoire afin de conserver pendant un moment les informations sur les solutions déjà visitées. Ces informations sont déclarées taboues et elles sont stockées dans une liste de longueur donnée, appelée la « liste des tabous » ou la « mémoire tabou » (qui a donné le nom à la méthode). Les informations données par la liste taboue sont utilisées pour établir une restriction appelée « restriction tabou », qui permet de classer certains mouvements comme étant interdits et permet ainsi d'éviter de retourner à des solutions déjà visitées dans un passé récent (Phénomène de cyclage). La procédure s'arrête lorsqu'une condition d'arrêt est satisfaite, généralement après un nombre fixé d'itérations ou encore après un nombre d'itérations sans amélioration de la solution. Pour certains problèmes d'optimisation, la recherche tabou donne de bons résultats. De plus, dans sa forme de base, la méthode contient moins de paramètres que le recuit simulé, ce qui la rend simple à utiliser [13], Ce principe peut se résumer par l'algorithme 2.

Algorithme 2 : Pseudo code de la recherche tabou.

Début
  $s \leftarrow s_0$ → s_0 est la solution initiale.
  L ={} →la liste des tabous est vide initialement.
  Tant que la condition d'arrêt n'est pas vérifiée
    Générer un ensemble $N' \subseteq N(s)$ de solution voisines de s
    Choisir la meilleur solution $s' \in N'$ telle que $s' \notin L$
    Mettre à jour la liste L des solutions taboues
    $s \leftarrow s'$
  Fin Tant que
Fin

3.2.2. Les Méta-heuristiques à base de population

Contrairement, Dans cette classification, La méta-heuristique manipule un ensemble de solutions en parallèle, à chaque itération. L'idée centrale consiste à utiliser régulièrement les propriétés collectives d'un ensemble de solutions distinguables (la population), dans le but de guider efficacement la recherche vers de bonnes solutions dans l'espace de recherche. Une grande variété de méta-heuristiques basées sur une population de solutions ont été proposées dans la littérature pour résoudre les problèmes d'optimisation, tel que : les algorithmes génétiques, l'optimisation par essaims particulaires, les algorithmes de colonies de fourmis, etc.

Dans ce qui suit, nous présentons, un peu plus des détails sur les algorithmes génétiques, et les algorithmes de colonies de fourmis.

3.2.2.1. Les Algorithmes Génétiques

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes inspirés par l'évolution biologique des espèces. Ils ont été introduits par John Holland et de ses collègues et élèves de l'Université du Michigan qui ont, dès 1960, travaillé sur ce sujet. Le premier aboutissement de ces recherches a été la publication en 1975 de ''*Adaptation in Natural and Artificial System*''. Leur but est d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode exacte (ou que la solution est inconnue) pour le résoudre en un temps raisonnable. Les algorithmes génétiques utilisent la notion de sélection naturelle et l'appliquent à une population de solutions potentielles au problème donné.

Par analogie à la génétique, les algorithmes génétiques se basent sur une population initiale d'individus, de taille bien déterminée, de solutions admissibles. Les solutions sont aussi appelées individus ou chromosomes. Par la suite, la structure de la population change en appliquant des évolutions progressives sur plusieurs générations. Pendant chaque génération, des nouveaux individus sont générés par les opérateurs de sélection, de croisement et de mutation. Nous présentons dans cette section les différentes étapes d'un algorithme génétique standard qui sont illustrées par l'algorithme 3 [3].

Algorithme 3 : Structure générale d'un algorithme génétique.

Debut
  Initialiser les paramètres de l'algorithme génétique.
  Générer la population initiale.
  Tant que critère d'arrêt non satisfait
    Évaluer les individus.
    Choisir les parents en se basant sur une stratégie de sélection.
    Appliquer l'opérateur de croisement avec un taux de croisement associé.
    Appliquer l'opérateur de mutation avec un taux de mutation associé.
    Remplacement de la population
  Fin Tant que
  Retourner le meilleur individu.
Fin

3.2.2.2. L'optimisation par colonie de fourmis(ACO)

Les algorithmes de colonies de fourmis (en anglais, ant colony optimization, ou ACO) sont des algorithmes inspirés du comportement des fourmis. Initialement proposé par Marco Dorigo et al. Dans les années 19901 pour la recherche de chemins optimaux dans un graphe (la résolution d'un problème du voyageur du commerce), le premier algorithme s'inspire du comportement des fourmis recherchant un chemin entre leur colonie et une source de nourriture. L'idée originale s'est depuis diversifiée pour résoudre une classe plus large de problèmes [C].

Nous présentons dans la suite de cette section les différentes étapes d'un algorithme de colonies de fourmis pour un problème de voyageur de commerce (TSP) qui sont illustrées par l'algorithme 4.

Algorithme 4 : le schéma général de l'algorithme de colonies de fourmis pour le problème du voyageur de commerce (TSP).

```
Debut
    Initialiser une population de m fourmis.
    Évaluer les m fourmis.
    Tant que le critère d'arrêt n'est pas atteint faire
        Pour i=1 à m faire
            Construire le trajet de la fourmi i ;
            Déposer des phéromones sur le trajet de la fourmi i ;
        Fin pour
        Évaluer les m fourmis ;
        Évaporer les pistes de phéromones ;
    Fin Tant que
    Retourner la ou les meilleures solutions ;
Fin
```

3.3. Les algorithmes hybrides

Les différentes méthodes vues précédemment possèdent bien entendu leurs propres avantages et inconvénients, en termes de qualité de solutions fournies, de la complexité temporelle et la simplicité d'implémentation. Une tendance actuelle en optimisation combinatoire consiste à coopérer/ hybrider plusieurs méthodes parmi celles que nous venons de citer. La motivation derrière une telle hybridation est d'avoir des méthodes plus sophistiquées et plus efficaces.

Le développement et l'application des méta-heuristiques hybrides est de plus en plus susciter l'intérêt académique, ces les méthodes hybrides combiner différents concepts ou des composants de divers méta-heuristiques et à cette fin, ils tentent de fusionner les points forts et éliminent les faiblesses des différents concepts de méta-heuristiques. Par conséquent, l'efficacité de l'espace de solution rechercher peut-être encore améliorée et de nouvelles opportunités émergé ce qui peut conduire à des méthodes de recherche encore plus puissante et plus flexible [14]. Talbi [15] a proposé une taxonomie pour les méta-heuristiques hybrides, [16], propose une classification des techniques d'hybridation en les classifiant en trois catégories selon leur architecture :

3.3.1. Hybridation séquentielle

C'est l'hybridation la plus simple et populaire. Elle consiste à exécuter séquentiellement différentes méthodes de recherche de telle manière que le (ou les) résultat(s) d'une méthode serve(nt) de solution(s) initiale(s) à la suivante comme le montre la figure II.4.

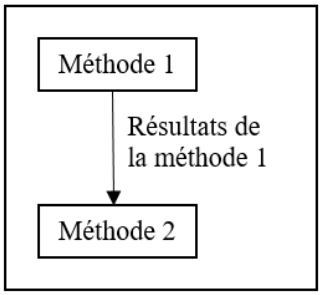

Figure II.5. Hybridation séquentielle.

3.3.2. Hybridation parallèle synchrone

Cette hybridation est réalisée par incorporation d'une méthode de recherche particulière dans un opérateur comme le montre la figure II.5. Elle est la plus fine et plus complexe à mettre en œuvre que la précédente.

L'objectif est de combiner une recherche locale avec une recherche globale dans le but d'améliorer la convergence. Un exemple de ce type d'hybridation est de remplacer l'opérateur de mutation d'un algorithme génétique par une recherche taboue.

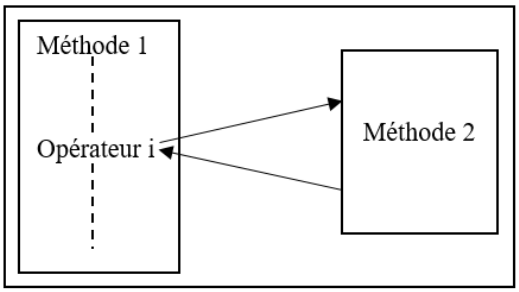

Figure II.6. Hybridation parallèle synchrone.

3.3.3. Hybridation parallèle asynchrone

Hybridation parallèle asynchrone (coopérative) : les méthodes hybrides appartenant à cette classe sont caractérisées par une architecture telle que deux algorithmes A et B sont impliqués simultanément et chacun ajuste l'autre. Ils aussi partagent et échangent de l'information tout au long du processus de recherche.

Cette coévolution permet une bonne coopération des méthodes de recherche au travers d'un coordinateur qui est chargé d'assurer les échanges d'informations entre les méthodes de recherche constituant l'hybride (cf. Figure II.6.).

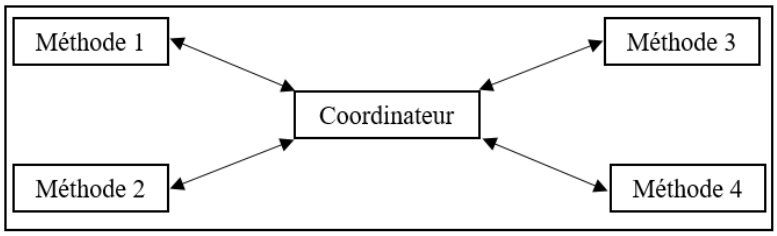

Figure II.7. Hybridation parallèle asynchrone.

4. Conclusion

Dans ce chapitre, nous avons présenté les techniques utilisées pour résoudre les problèmes d'optimisation, et en particulier, les problèmes d'ordonnancement. Pour les deux classes de méthodes : exactes et approximatives. Et nous avons présenté le concept, le principe, les techniques et les étapes de chaque méthode.

Chapitre III

« Résolution du problème Flow Shop à deux machines avec des temps de latence »

1. Introduction

Nous nous concentrons dans ce chapitre au problème de flow-shop. Dans un premier lieu nous étudions ces différentes propriétés et restrictions. Ensuite nous explorons l'algorithme de Johnson avec et sans temps de latence, Dans un deuxième temps nous étudions le problème de flow-shop à deux machines avec des temps de latence et d'exécution quelconques. Et pour terminer ce chapitre nous décrivons les différentes bornes inférieures et supérieures utilisées pour la réalisation de l'algorithme de Branch and Bound.

2. Le problème de flow-shop

Dans le cas de flow shop, tout travail visite chaque machine de l'atelier et l'ordre de passage d'un travail sur les différentes machines est le même pour tous les travaux (flux unidirectionnel). Cet ordre, ou gamme, unique est une donnée de problème. Cette particularité se rencontre très fréquemment en pratique, elle correspond par exemple à une chaine de traitement ou de montage. Dans les ateliers de type Flow Shop hybride, une machine peut exister en plusieurs exemplaires identiques et parallèles.

La résolution du problème de Flow Shop consiste à déterminer l'ordre de passage des jobs sur l'ensemble des machines ainsi que les instants de début et de fin des opérations des jobs afin de réduire au minimum le temps total d'exécution de tous les jobs, ainsi appelé Makespan C_{max}.

Ce type de production se distingue de celui de la chaine de production par trois caractéristiques :

- Le fait qu'une tâche peut ne pas faire appel à tous les centres de production (un nombre quelconque de centres pouvant être « sautés ») ;

- La dispersion importante des temps opératoires des opérations exécutées sur un même poste de travail (en raison d'une absence de spécialisation étroite du poste dans l'exécution d'une même opération) ;

- L'existence de files d'attente, de longueur variable au cours de temps, en amont des différents postes de travail. On distingue quatre types de Flow Shop :

- ***Flow Shop pur*** **:** tous les temps opératoires sont positifs.

- ***Flow Shop généralisé*** **:** les temps opératoires peuvent être nuls si une tâche ne doit pas subir un traitement sur une machine particulière.

- ***Flow Shop de permutation*** **:** c'est-à-dire une chaîne de fabrication où les tâches sont disponibles à l'instant 0, et ne se doublent pas entre les postes (ou les machines), l'ordre d'exécution des tâches est donc le même sur toutes les machines.

- ***Flow Shop hybride*** **:** c'est un atelier Flow Shop dans lequel chaque machine est remplacée par un étage composé de plusieurs machines qui ne sont pas forcément identique disposées en parallèle [17].

2.1. Les données d'un problème Flow Shop

Le flow-shop définit un ensemble de *n* jobs (tâches), *notés* $j_1, j_2, ..., j_n$, et *m* machines notées $M_1, M_2, \ldots, M_m$, Chaque job doit être exécuté, au plus une seule fois, sur M_1 puis M_2 et ainsi de suite, jusqu'à ce qu'il soit exécuté sur la dernière machine M_m dans cet ordre. Chaque job est donc composé de *m* opérations élémentaires O_{j1}, O_{j2}, …, O_{jm}. Pour chaque opération O_{ij}, on désigne P_{ij} son temps d'exécution. Et leurs fonctions objectives sont généralement :

- Le Makespan C_{max} : date de fin de la dernière tâche sur la machine M, soit le temps total passé à exécuter tous les travaux.

- La somme des dates de fin des tâches sur la machine M $\sum_{i=1}^{n} Ci$.

Sur cette définition du problème de flow-shop peuvent venir se greffer des nombreuses notions pour rendre compte des contraintes réelles d'un atelier. Nous ne présentons ici que les plus usuelles, même si cette liste non exhaustive peut être très largement complétée :

- ***Les stocks inter-machines :*** les jobs transitent par un stock limité pour aller d'une machine à l'autre. La politique de gestion de ce stock, ainsi que le nombre de jobs qui peuvent y être entreposées modifient la structure du problème.

- ***Les temps de montage :*** lorsqu'une machine finit d'exécuter un job pour en commencer une autre, elle peut avoir besoin de subir un changement quelconque de son mode opératoire. La durée de ces changements peut alors être prise en compte dans les solutions.

- ***Les moyens et les temps de transports ou de latence :*** pour qu'un job puisse passer d'un centre à un autre, ce dernier doit emprunter un moyen de transport.

- ***La distance entre les centres :*** le temps de transport des opérations d'un centre à l'autre, est une contrainte à prendre en compte lors de la construction d'une solution.

- ***La disponibilité des machines :*** les machines peuvent être soumises à des périodes d'inactivité qui peuvent être des périodes de maintenance.

- ***La nature des machines :*** certaines machines d'un même centre peuvent par exemple être plus rapides que d'autres.

Pour les besoins de notre étude, nous supposons également ce qui suit :

- ***La non préemption des opérations :*** une fois que l'exécution d'un job a débuté sur une machine, celle-ci ne peut pas être interrompue. Aucune opération ne peut commencer sur cette machine avant la fin de l'opération en cours.

- ***Les durées des opérations sont entières :*** cette hypothèse n'est pas réductrice en général. Elle permet simplement de traiter le problème en évitant la manipulation de nombres réels.

- ***Les machines ne peuvent exécuter qu'une opération à la fois :*** ces machines sont disponibles sans restriction du début à la fin de l'ordonnancement.

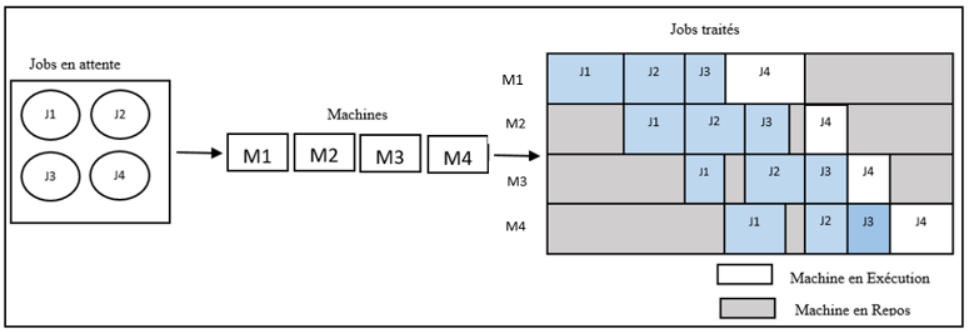

Figure III.1. Modèle de flow-shop.

2.2. Propriétés du problème Flow Shop

Les premières recherches effectuées sur le problème de Flow Shop se sont essentiellement focalisées sur les problèmes de permutations. Sous classe du problème de flow shop, un problème de Flow Shop de permutation est un problème de Flow Shop ou l'ordre de passage des jobs sur chacune des machines est identique.

Résoudre ces problèmes revient à trouver une permutation π= (π (1), π (2), ……, π (n)) minimisant le *makespan*, Il est symbolisé par *Fm |perm |*C_{max}, (m le nombre de machines).

La restriction aux Flow Shop de permutation permet de simplifier la structure des solutions et des preuves. En effet, d'un point de vue pratique, l'ordonnancement obtenu à une structure simple pouvant être facilement implémentée.

Pour des temps de latence nuls, le problème de Flow Shop est dominé par la permutation pour un nombre de machines m≤3. Johnson en 1954 a prouvé que le problème à deux machines est résoluble en complexité $O(n \log n)$. Le problème devient NP-difficile au sens fort pour m≥3. Concernant le problème de permutation de Flow Shop avec des temps de latence.

Mitten en1959 a montré que ce problème à deux machines peut être résolu en une complexité de $O(n \log n)$, pour le cas unitaire, il est résolu en $O(n)$, pour m=3, le problème peut être résolu $O(n \log n)$ et en $O(n^2)$ pour m=4. La complexité de ce problème pour m≥5 reste encore inconnue. Rebaine [18], en 2005, a montré que, dans le cas des temps d'exécutions unitaires et *m* machines, la meilleure permutation est pire d'un facteur m que le meilleur ordonnancement d'un flow shop. Cependant, il existe des cas ou la permutation est encore dominante.

3. Un problème Flow Shop à deux machines avec des temps de latence

Jusqu'à un passé récent, il a été souvent supposé dans la littérature que les temps de latence τ_j induits par exemple par le déplacement d'un job j d'une machine i à une autre, sont négligeables.

Soit l'exemple ci-dessous de flow-shop à deux machines $F2|perm|C_{max}$ et six jobs. Le Tableau III.1 présente les temps d'exécution de ces jobs sur les deux machines:

Machines \ Jobs	J1	J2	J3	J4	J5	J6
M1	1	3	8	5	10	4
M2	4	2	5	1	9	6

Tableau III.1. Temps d'exécution du problème n =6 et m =2.

Une solution optimale pour ce problème peut être obtenue comme illustrée par la Figure III.2.

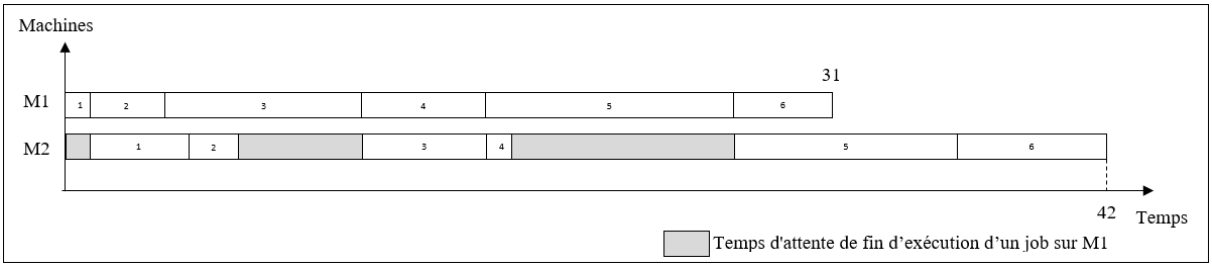

Figure III.2. Ordonnancement sans temps de latence.

Comme on peut le remarquer sur le diagramme de la Figure III.2, le temps pour transférer le job 2 de Ml à M2 est considéré comme nul. Cependant, en pratique, cette supposition est souvent non justifiée. En effet, la distance pouvant séparer les machines est en général importante. Autrement dit, le temps mis pour atteindre une machine à partir d'une autre peut être parfois plus important que les temps d'exécution eux-mêmes. Dans ce cas de figure, il est nécessaire de prendre en compte ces temps de latence lors de la construction de la solution. En effet, considérons l'exemple d'un problème Flow Shop ci-dessus avec en plus les temps de latence associés aux jobs. Notons que pour ce problème à deux machines [18]. Le Tableau III.2 présente les temps de latence de jobs.

Jobs	J1	J2	J3	J4	J5	J6
temps de latence τ_j	1	0	5	0	9	0

Tableau III.2. Temps de latence de jobs.

Comme il fallait s'y attendre, le Makespan a augmenté en considérant les temps de latence comme illustré par la Figure III.3.

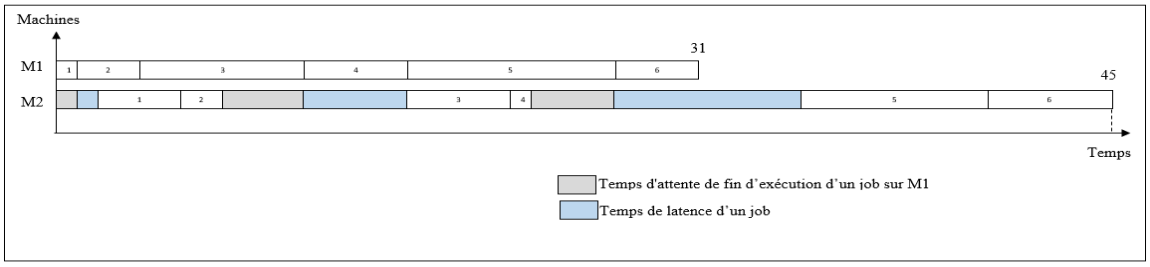

Figure III.3. Ordonnancement avec temps de latence.

Soit S un ordonnancement quelconque et C_{ij}, S_{hj} respectivement, la date de fin de l'exécution du job j sur la machine i, la date de début d'exécution d'un job j sur la machine h telle que $i \neq h$. Pour que cet ordonnancement soit valide, il est nécessaire que la relation $S_{hj} - C_{ij} \geq \tau_{ihj}$ soit vérifiée.

4. Approche heuristique pour le problème Flow Shop à deux machines avec des temps de latence

L'approche heuristique qui nous intéresse dans la résolution de notre problème est l'algorithme de Johnson pour des problèmes de Flow Shop sans temps de latence ainsi que l'algorithme de Johnson modifié que nous avons perfectionné selon la théorie de Mitten qui est utilisée dans le cas d'un problème de permutation avec des temps de latence $F2|perm|C_{max}$.

4.1. Algorithme de Johnson sans temps de latence

Johnson (1954) considère un problème de Flow Shop à deux machines. L'objectif est la minimisation du makespan $F2|perm|C_{max}$. Il proposa un algorithme et démontra que la même permutation des jobs pouvait être utilisée sur les deux machines. Il démontra aussi que s'il y a M machines, un ordonnancement optimal consiste à exécuter des séquences identiques de jobs sur les deux premières et deux dernières machines.

Cet algorithme consiste à diviser l'ensemble des jobs à traiter en deux sous-ensembles U et V : U contient les jobs i tels que $P_{1i} \leq P_{2i}$, et V contient les jobs i tels que $P_{1i} > P_{2i}$ Ces deux ensembles sont triés : U suivant l'ordre croissant des P_{1i}, et V suivant l'ordre décroissant des P_{2i}, U et V sont ensuite fusionnés de manière à ajouter les jobs de V à la fin de l'ensemble U.

Ces jobs seront ensuite exécutés dans cet ordre sur la première machine M1 puis sur la deuxième machine M2. En d'autres termes, les jobs ayant les plus petits temps d'exécution sur la première machine seront exécutés en premier, ceux qui ont les plus petits temps sur la deuxième machine seront exécutés à la fin. L'algorithme de Johnson est le suivant :

Algorithme 5 : L'algorithme de Johnson pour le problème de flow-shop à deux machines.

Debut
  Tant que la liste des jobs non vide **faire**
    Placer dans l'ensemble U les jobs pour lesquels $P_{1i} \leq P_{2i}$;
    Placer dans l'ensemble V les jobs pour lesquels $P_{1i} > P_{2i}$;
  Fin Tant que
  Ordonnancer les jobs de l'ensemble U dans l'ordre croissant des P_{1i} ;
  Ordonnancer les jobs de l'ensemble V dans l'ordre décroissant des P_{2i} ;
  On fusionne les deux ensembles U et V ;
  On exécute la permutation obtenue sur les deux machines tout en respectant le même ordre.
Fin

Exemple III.1

L'exemple ci-dessous illustre l'application de l'algorithme de Johnson sur 4 jobs J1, J2, J3 etJ4 et leurs temps d'exécution respectifs sur les deux machines M1 et M2.

Machines \ Jobs	J1	J2	J3	J4
M1	1	3	8	5
M2	4	2	7	6

Tableau III.3. Temps d'exécution du problème n =4 et m =2 de l'Exemple III.1.

On affecte les différents jobs à la liste U et V tout en respectant l'Algorithme 5. On aura :

- La liste U va contenir les jobs : J1, J4.
- La liste V va contenir les jobs : J2, J3.

L'ordre final sera alors J1, J4, J3, J2. Le diagramme de Gantt associé à cet exemple est illustré par la Figure III.2.

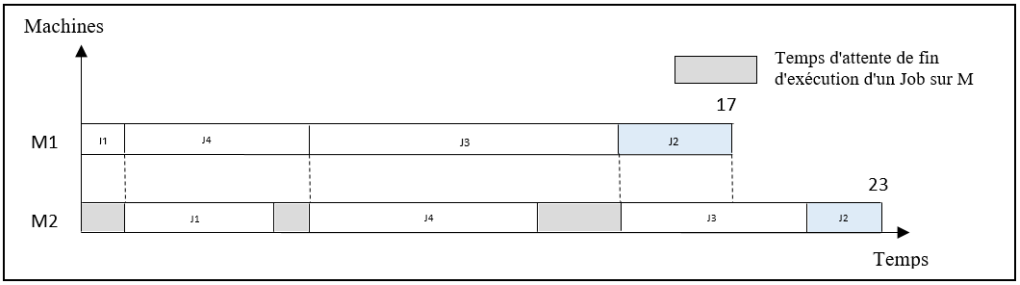

Figure III.4. Solution optimale de l'Exemple III.1.

4.2. Algorithme de Johnson modifié avec des temps de latence

Dans le cas d'un problème de permutation avec des temps de latence, l'algorithme de Johnson peut être utilisé pour résoudre ce problème. En effet, Mitten (1958) a obtenu le résultat suivant :

Selon la théorie de Mitten utilisée, la permutation optimale est obtenue en appliquant l'ordre de Johnson aux temps d'exécution pour "$P_{ij}=P_{ij}+\tau_j$".

On applique l'algorithme de Johnson modifié sur l'exemple III.1, avec les temps de latence que nous avons obtenu à partir de diagramme de Gantt, leurs temps d'exécution respectifs sur les deux machines M1 et M2 et le temps de latence τ_j pour (j = 1, 2, 3 et 4).

M / J	**J1**	**J2**	**J3**	**J4**
M1	1	3	8	5
M2	4	2	7	6
$\boldsymbol{\tau_j}$	1	0	2	1

Tableau III.4. Temps d'exécution du problème n =4 et m =2 avec des temps de latences.

Après avoir appliqué le Théorème de Mitten (1958) sur la séquence de jobs de notre exemple, on aura des nouveaux temps d'exécutions p_{1j} et p_{2j} comme suit :

M / J	**J1**	**J2**	**J3**	**J4**
M1	2	3	10	6
M2	5	2	9	7

Tableau III.5. Les nouveaux temps d'exécution.

La solution optimale est la permutation suivante : J1, J4, J3 et J2.

5. Approche exacte pour le problème Flow Shop à deux machines avec des temps de latence

Rappelons que le problème de Flow Shop à deux machines avec des temps de latence quelconques et d'exécution unitaires ou quelconques est NP-difficile. L'approche exacte qui nous intéresse dans la résolution de notre problème est la méthode énumérative de Branch and Bound dans le cas de temps d'exécution quelconques, le développement de cette méthode est basé sur l'utilisation des bornes inférieures et supérieures et une stratégie de branchement. Notons que cette partie est partiellement basée sur le travail de YU, Wenci (1996) [19].

5.1. Les bornes inférieures (Lowler Bounds)

Rappelons que le temps d'exécution d'un job *j (j* = 1, 2, ..., *n)* sur la machine i (i = 1, 2) est désigné par P_{ij} et son temps de latence par τ_j.

5.1.1. La Première borne inférieure (LB1)

La première borne inférieure LB1, que nous présentons, est donnée par la Théorème 3 de Yu Wenci (1996).

Théorème 3 :

Si Opt désigne la valeur optimale du makespan, alors

$$Opt \geq LB1 = \max\left(\sum_{j=1}^{n} p1j + min(p2j + \tau j), \sum_{j=1}^{n} p2j + \min\,(p1j + \tau j)\right)$$

Exemple III.2

Soit l'instance ci-dessous avec n = 5 jobs.

Jobs	**1**	**2**	**3**	**4**	**5**
Temps d'exécution p_{1j}	13	12	2	1	1
Temps d'exécution p_{2j}	8	8	7	6	1
Temps de latence τ_j	9	7	2	1	0
$p_{1j} + \tau_j$	22	19	4	2	1
$P_{2j} + \tau_j$	17	15	9	7	1

Tableau III.6. L'application de LB1 sur l'ExempleIII.2.

Les cases grises représentent respectivement le minimum de $P_{1j} + \tau_j$ et celui de $P_{2j} + \tau_j$. Le calcul de la première borne inférieure LB1 est le suivant :
LB1 =max {(13+12+2+1+1) + 1, (8+8+7+6+1) +1} =31.
Étant donnée une sous séquence a des jobs déjà ordonnancés, nous présentons, dans ce qui suit, trois bornes inférieures LB2, LB3, LB4.

5.1.2. La Deuxième borne inférieure (LB2)

La deuxième borne inférieure LB2 est calculée à chaque nœud de l'arbre de recherche. Elle est donnée par la Théorème 4 de Yu Wenci.

Théorème 4

$$Opt \geq LB2 = \max (p_{1j}+\tau_j + p_{2j})$$

Preuve

Quel que soit le job, il aura à s'exécuter sur M1, ensuite un temps de latence s'en suit avant de s'exécuter sur M2. Il est clair que le Makespan ne peut être inférieur à la somme de ces trois valeurs.

Étant donné α jobs déjà ordonnancés, la date au plus tôt d'un job A sur la machine M2 est $\sum_{j=1}^{A} p1j + \tau A$ pour tout jobs A ∈ α. Il est clair que les jobs restants appartenant à la séquence n - α vont commencer leur exécution après que les jobs de la séquence α soient exécutés. De plus, la borne inférieure correspondant aux jobs appartenant à la séquence n - α, est clairement LB2 (n- α), c'est-à-dire LB2 appliquée aux jobs de l'ensemble n - α. Par conséquent, le Makespan de l'ensemble de jobs, étant donnée la séquence de jobs α, est $\sum_{j\in\alpha} p1j + LB2(n - \alpha)$. Cette borne peut être obtenue en appliquant l'Algorithme 6.

Algorithme 6 : Algorithme de la deuxième borne inférieure LB2 dans le cas quelconque.

Début

|α| : représente le cardinal de la sous-séquence α.
n – α : étant le nombre de jobs non ordonnancés et qui n'appartiennent pas à α.
On calcule la borne LB2 sur le reste de jobs n'appartenant pas à a on aura alors :

$$LB2 = \max_{j\in n-\alpha} (p_{1j}+\tau_j + p_{2j}).$$

La valeur de la borne inférieure finale est la suivante : $LB2 = \sum_{j\in\alpha} p1j + LB2(n - \alpha)$.

Fin

Exemple III.3

Soit l'instance ci-dessous avec n = 5 jobs, et α = {1,2}.

	j ϵ α		**j ϵ n - α**		
Jobs	**1**	**2**	**3**	**4**	**5**
Temps d'exécution p_{1j}	13	12	2	1	1
Temps d'exécution p_{2j}	8	8	7	6	1
Temps de latence τ_j	9	7	2	1	0
$p_{1j} + \tau_j + p_{2j}$	/////////	/////////	11	8	2

Tableau III.7. L'application de LB2 sur l'ExempleIII.3.

Pour calculer LB2, on doit procède comme suit : On détermine: $\sum_{j=1}^{2} p1j = 12 + 13 = 25$ avec j ϵ α. Ensuite, on applique la formule de LB2 sur les n - α jobs restants. La case grise représente le maximum de ($p_{1j} + \tau_j + p_{2j}$) La valeur finale de LB2 est comme suivie :

$$LB2 = \sum_{j=1}^{2} p1j + LB2(n - \alpha) = 25 + \max\{11,8,2\} = 36.$$

5.1.3. La Troisième borne inférieure (LB3)

Notons que les deux bornes inférieures LBI et LB2 contiennent seulement un seul terme représentant le temps de latence. Il est clair qu'il est préférable d'impliquer l'ensemble de temps de latence pour espérer avoir une borne inférieure efficace. La troisième borne inférieure LB3, que nous présentons, est donnée par la Théorème 7 toujours de Yu Wenci.

Théorème 7 : Soient q_j =min (p_{1j}, p_{2j}), r_j =max (p_{1j}, p_{2j}) avec (j = 1, 2, ..., n). Si Opt désigne le Makespan de la solution optimale, alors :

$$Opt \geq LB3 = \left[\left(\sum_{j=1}^{n} qj\,(\tau j + rj - 1) \right) \Big/ \sum_{j=1}^{n} qj \right] + 1 + \sum_{j=1}^{n} qj$$

Cette borne peut être obtenue en appliquant l'Algorithme 7 sur notre arborescence.

Algorithme 7 : Algorithme de la troisième borne inférieure LB3 dans le cas quelconque.

Début

|α| : représente le cardinal de la sous-séquence α.

n – α : représente le nombre de jobs qui n'appartiennent pas à α.

$q_j = \min_{j \epsilon n-\alpha} (p_{1j}, p_{2j})$,

$r_j = \max_{j \epsilon n-\alpha} (p_{1j}, p_{2j})$,

On calcule la borne LB3 sur le reste de jobs n'appartenant pas à a on aura alors :

$$LB3 = \left[\left(\sum_{j=1}^{n} qj\,(\tau j + rj - 1)\right) \Big/ \sum_{j=1}^{n} qj\right] + 1 + \sum_{j=1}^{n} qj$$

La valeur de la borne inférieure finale est la suivante : $LB3 = \sum_{j\in\alpha} \text{p1j} + \text{LB3}(\text{n} - \alpha)$.

Fin

Exemple III.4

L'exemple ci-dessous illustre le calcul de la borne inférieure LB3. On suppose qu'on a une sous séquence a de taille 2 comme dans l'exemple précédent.

	j ϵ α		**j ϵ n - α**		
Jobs	**1**	**2**	**3**	**4**	**5**
Temps d'exécution p_{1j}	13	12	2	1	1
Temps d'exécution p_{2j}	8	8	7	6	1
Temps de latence τ_j	9	7	2	1	0
$r_j = \max_{j \epsilon n-\alpha} (p_{1j}, p_{2j})$	/////////	/////////	7	6	1
$q_j = \min_{j \epsilon n-\alpha} (p_{1j}, p_{2j})$	/////////	/////////	2	1	1
$\tau_j + r_j - 1$	/////////	/////////	8	6	0
$q_j (\tau_j + r_j - 1)$	/////////	/////////	16	6	0

Tableau III.8. L'application de LB3 sur l'ExempleIII.4.

On peut procéder comme suit pour le calcul de LB3 :

1. On calcule LB3 pour les jobs n'appartenant pas à α. On aura :

   LB3 (n - α) = [(16+ 6+0) /4] + 1+ 4 = [(22) /4] + 5 =11.

2. Enfin, la valeur finale de la borne LB3 est : $LB3 = \sum_{j\in\alpha} \text{p1j} + \text{LB3}(\text{n} - \alpha)$= 25+11=36.

5.1.4. La quatrième borne inférieure (LB4)

Elle est donnée par la Théorème 8 de Yu Wenci.

Théorème 8 : Soit p'_{ij} la somme de j plus petites valeurs de $\{p_{1j}, p_{2j}, ..., p_{in}\}$ avec i=1, 2 et j =l, 2, ..., n, alors on aura :

$$Opt \geq LB4 = \left[\left(\sum_{j=1}^{n} \tau j + \sum_{j=1}^{n} p'1j + \sum_{j=1}^{n} p'2j\right)/n\right].$$

Et pour appliquer la quatrième borne inférieure LB4, on doit d'abord vérifier les temps de latence de chaque job appartenant à α car ils peuvent augmenter en plaçant, par exemple, deux jobs sur le même emplacement. Pour cela, on applique l'Algorithme 8.

Après avoir obtenu les nouveaux temps de latence, on applique le Théorème 8 sur l'ensemble des jobs.

Algorithme 8 : Algorithme de vérification des temps de latence.

Début
q : fin de temps d'exécution d'un job sur M2.
Pour p = 1 to | α | faire
$\tau j = q - \sum_{j=1}^{p} p1j - p2j$
Fin pour
Fin

Exemple III.5

Soit l'instance suivante avec n = 5 jobs. On suppose qu'on a une sous séquence α = 3 de jobs.

	j ϵ α			**j ϵ n - α**	
Jobs	**4**	**5**	**3**	**1**	**2**
Temps d'exécution p_{1j}	1	1	2	13	12
Temps d'exécution p_{2j}	6	1	7	8	8
Temps de latence τ_j	1	0	2	9	7

Tableau III.9. Les temps d'exécution et de latence de l'Exemple III.5.

Pour calculer la borne LB4, on doit tout d'abord vérifier s'il a un changement dans les temps de latence des jobs appartenant à α. Donc, on doit tout d'abord ordonnancer les jobs appartenant à α comme illustré dans la Figure III.2:

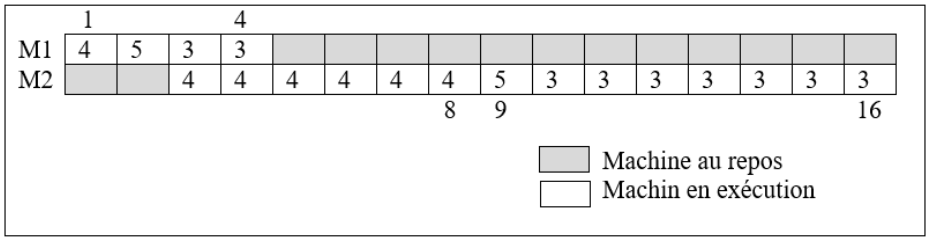

Figure III.5. Application de LB4 avec des temps d'exécution quelconque.

De la Figure III.2, on peut remarquer qu'il y'a eu des changements dans les temps de latence des jobs appartenant à α. On a donc :

- Pour p= 1 : j= 4, τ_4= 1. Donc, le temps modifié de τ_4 =8-1-6=1(dans ce cas, on n'a pas de changement). - Pour p= 2 : j= 5, τ_5= 0. Donc, le temps modifié de τ_5 =9-2-1=6. Notons que, dans ce cas, on a fait un changement.

- Pour p= 3 : j= 3, τ_3= 2. Donc, le temps modifié de τ_3 =16-4-7=5. Notons que, dans ce cas, on a fait également un changement.

Après avoir modifié les temps de latence, notre tableau sera comme suit :

	j ϵ α			**j ϵ n - α**	
Jobs	**4**	**5**	**3**	**1**	**2**
Temps d'exécution p_{1j}	1	1	2	13	12
Temps d'exécution p_{2j}	6	1	7	8	8
Temps de latence τ_j	1	6	5	9	7
p'_{1j}	1	2	4	17	29
p'_{2j}	6	7	14	22	30

Tableau III.10. Application de LB4 sur l'Exemple III.5.

On a: $LB4 = \left[\left(\sum_{j=1}^{n} \tau j + \sum_{j=1}^{n} p'1j + \sum_{j=1}^{n} p'2j\right)/n\right]$= [(1+6+5+9+7) + (1+2+4+17+29) + (6+7+14+22 +30) /5] = 32.

5.2. Les bornes supérieures (Upper Bounds)

Nous utilisons dans notre algorithme de Branch and Bound quatre bornes supérieures basées sur des heuristiques ci-dessous. Notons que ces heuristiques sont conçues pour être utilisées pour les nœuds internes de l'arbre de recherche, c'est-à-dire qu'il est supposé qu'une sous-séquence de jobs α est déjà fixée.

5.2.1. La Première borne supérieure (UB1)

Pour calculer la première borne supérieure, on place tout d'abord les jobs appartenant à α sur la machine M1. Ensuite, on passe à traiter les n - α jobs restants. Nous devrons appliquer l'algorithme de Johnson modifié, présenté à la section 3.2. Puisqu'il est à l'origine de plusieurs travaux sur deux machines, nous avons jugé utile de l'utiliser pour calculer UB1. Et enfin, on place les n - α jobs à la suite des α jobs déjà placés sur M1. L'heuristique UB1 est obtenue en appliquant l'Algorithme 9.

Algorithme 9 : Algorithme de la première borne supérieure UB1 pour le cas quelconque.

Début
|α| : représente le cardinal de la sous-séquence α.
n – α : représente le nombre de jobs qui n'appartiennent pas à α.
Pour i= 1 jusqu'à 2 **faire**
Pour j ϵ n - α **faire**
Appliquer l'algorithme de Johnson modifié
Fin pour
Fin pour
Après la fin d'exécution de chaque job j sur M1, on essaie de le placer le plus tôt possible sur M2.
Fin

Exemple III.6

L'exemple ci-dessous illustre le calcul de la borne supérieure UB1 : Soit l'instance suivante avec n = 5 jobs, Tableau III.9.

Jobs	**1**	**2**	**3**	**4**	**5**
Temps d'exécution p_{1j}	13	12	2	1	1
Temps d'exécution p_{2j}	8	8	7	6	1
Temps de latence τ_j	9	7	2	1	0

Tableau III.11. Les temps d'exécution et de latence de l'Exemple III.6.

Soit la sous séquence α= (1,2), Les jobs restants sont : 3, 4 et 5. On applique l'algorithme de Johnson modifié sur Les jobs restants (n - α). On aura alors deux sous-ensembles U et V qui contiendront les jobs suivant : U= {4, 3,5} et V={ø}. On obtient alors le Tableau III.10.

	j ϵ α		**j ϵ n - α**		
Jobs	**1**	**2**	**3**	**4**	**5**
Temps d'exécution p_{1j}	13	12	2	1	1
Temps d'exécution p_{2j}	8	8	7	6	1
Temps de latence τ_j	9	7	2	1	0
$p'_{1j} = p_{1j} + \tau_j$	/////////	/////////	4	2	1
$p'_{2j} = p_{2j} + \tau_j$	/////////	/////////	9	7	1

Tableau III.12. L'application de UB1 sur l'Exemple III.6.

La séquence finale des jobs sera ordonnancée comme suit : Après avoir placé les α jobs, on met les n-α jobs restants. On obtient la permutation : 1, 2, 4, 5 et 3. L'ordonnancement obtenu est présenté par la Figure III.3. La valeur du Makespan est donc UB1 = 53.

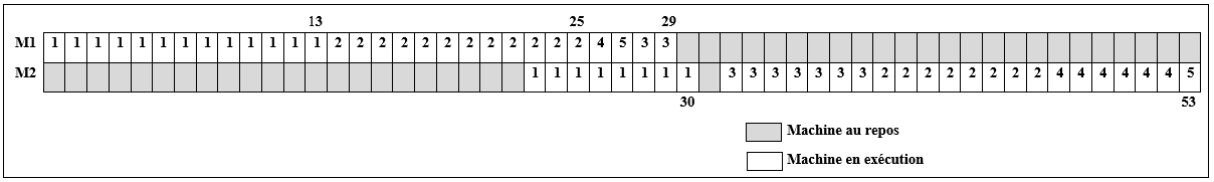

Figure III.6. L'application du UB1 sur la séquence des jobs de l'exemple III.6.

5.2.2. La Deuxième borne supérieure (UB2)

La deuxième borne supérieure UB2 est obtenue comme suit : on commence par placer les α jobs sur M1. Puis, on va modifier les temps d'exécution des jobs restants (n - α). Autrement dit, les nouveaux temps d'exécution sont : $p'_{ij} = p_{ij} + \tau_j$ (i = 1, 2, j = 1, 2, ... n).

Ensuite, on ordonnance les jobs suivant ces nouveaux temps d'exécution obtenus dans l'ordre décroissant (l'algorithme LPT) et on place la séquence obtenue sur la première machine M1, ensuite on place les jobs le plus tôt possible sur M2. Quoique très simple, cette heuristique s'est avérée efficace. On calcule enfin le Makespan pour obtenir alors la deuxième borne supérieure UB2. Le calcul de la deuxième borne est illustré à l'Algorithme 10.

Algorithme 10 : Algorithme de la deuxième borne supérieure UB2 pour le cas quelconque.

Début
- |α| : représente le cardinal de la sous-séquence α.
- n – α : représente le nombre de jobs qui n'appartiennent pas à α.
- **Pour** i= 1 jusqu'à 2 **faire**
  - **Pour** j ϵ n - α **faire**
    - $p'_{ij} = p_{ij} + \tau_j$
  - **Fin pour**
- **Fin pour**
- Ordonnancer les jobs suivants l'ordre décroissant de p'_{1j} et on les place sur M1.
- Après la fin d'exécution de chaque job j sur M1, on essaie de le placer le plus tôt possible sur M2.

Fin

Exemple III.7

Soit l'instance suivante avec n = 5 jobs, et la sous séquence α= (1,2). On doit tout d'abord modifier les temps d'exécution des jobs restants (n - α) On applique la formule suivante : $p'_{ij} = p_{ij} + \tau_j$. On aura alors des nouveaux temps d'exécution présentés dans le Tableau III.11 ci-dessous :

	j ϵ α		**j ϵ n - α**		
Jobs	**1**	**2**	**3**	**4**	**5**
Temps d'exécution p_{1j}	13	12	2	1	1
Temps d'exécution p_{2j}	8	8	7	6	1
Temps de latence τ_j	9	7	2	1	0
$p'_{1j} = p_{1j} + \tau_j$	/////////	/////////	4	2	1
$p'_{2j} = p_{2j} + \tau_j$	/////////	/////////	9	7	1

Tableau III.13. L'application de UB2 sur l'Exemple III.7.

Ensuite, on ordonnance les jobs suivant l'ordre décroissant de p'_{1j}, on obtient alors la séquence de jobs : 1, 2, 3, 4 et 5, illustrée par la Figure III.4. La valeur de la borne UB2 est donnée par le Makespan qui est égal à 53.

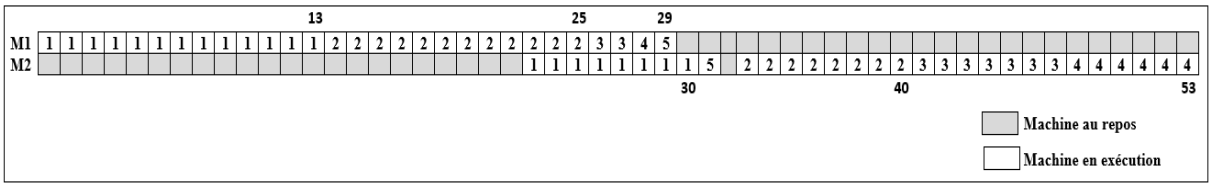

Figure III.7. L'application du UB2 sur la séquence des jobs de l'Exemple III.7.

5.2.3. La Troisième borne supérieure (UB3)

L'heuristique pour obtenir la troisième borne supérieure UB3 consiste à affecter des priorités aux différents jobs. On applique cette priorité au n - α jobs. Le job qui a la priorité la plus élevée sera exécuté en premier et ainsi de suite.

Pour affecter une priorité S_j à un job j on calcule cette formule :

$Sj = (\sum_{i=1}^{m}(m - 2i + 1)Pij) + \tau j$, avec i= (1,2) et j = (1, 2, …n).

Cette formule pour le calcul des priorités est une modification de celle utilisée par Palmer (1965). Ensuite, on ordonnance les S_j suivant l'ordre décroissant. Puis on place les n - α jobs obtenus sur la machine M1 à la suite des a jobs déjà ordonnancés et au plus tôt possibles sur M2. On calcule enfin le makespan. Le calcul de la troisième borne est illustré à l'Algorithme 11.

Algorithme 11 : Algorithme de la troisième borne supérieure UB3 pour le cas quelconque.

Début

|α| : représente le cardinal de la sous-séquence α.

n – α : représente le nombre de jobs qui n'appartiennent pas à α.

Pour i= 1 jusqu'à 2 **faire**

Pour j ϵ n - α **faire**

$S_j = (\sum_{i=1}^{m}(m-2i+1)Pij) + \tau_j$

Fin pour

Fin pour

On ordonnance les S_j suivant l'ordre décroissant. Puis on place les n - α jobs obtenus sur la machine M1 à la suite des a jobs déjà ordonnancés et au plus tôt possibles sur M2.

Fin

Exemple III.8

Pour mieux comprendre cette borne, considérons l'exemple suivant. Soit α = (3,4). Le tableau III.12 présente la priorité des jobs restants (l, 2 et 5).

	j ϵ α		**j ϵ n - α**		
Jobs	**3**	**4**	**1**	**2**	**5**
Temps d'exécution p_{1j}	2	1	13	12	1
Temps d'exécution p_{2j}	7	6	8	8	1
Temps de latence τ_j	2	1	9	7	0
$S_j = (\sum_{i=1}^{m}(m-2i+1)Pij) + \tau_j$	/////////	/////////	14	11	0

Tableau III.14. L'application de UB3 sur l'Exemple III.8.

Le nouvel ordre de la séquence des jobs sera le suivant : 3, 4, l, 2 et 5. La Figure III.5 représente l'ordonnancement de la séquence sur les deux machines. La borne UB3 est égale à 43.

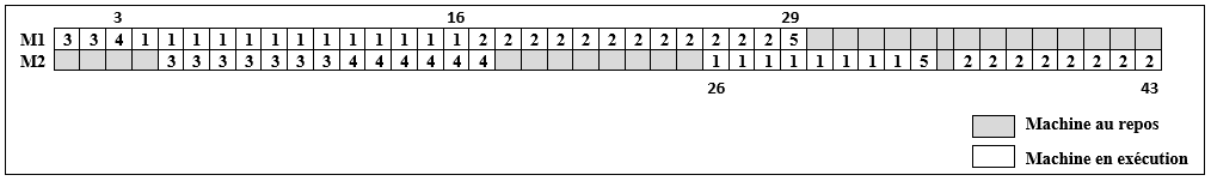

Figure III.8. L'application du UB3 sur la séquence des jobs de l'exemple III.8.

5.2.4. La Quatrième borne supérieure (UB4)

Pour le calcul de la quatrième borne supérieure UB4, nous appliquons l'heuristique NEH [20]. Cette heuristique est basée sur l'hypothèse qu'un lot de jobs ayant un temps total d'exécution élevé est prioritaire (le lot est positionné en priorité dans un ordonnancement partiel) par rapport à un job de faible priorité. On classe les jobs dans l'ordre croissant de leur durée de traitement totale p_{ij} : temps d'exécution sur les deux machines et celui de latence ($p_{ij} = p_{1j} + \tau_j + p_{2j}$). Puis, on construit progressivement la séquence correspondant à la solution en insérant, à chaque

étape, le job suivant dans la liste à la meilleure place dans la séquence, de manière à minimiser le makespan. L'adaptation de l'algorithme NEH pour le calcul de la borne UB4 est illustrée à l'Algorithme 12.

Algorithme 12 : Algorithme d'adaptation NEH pour le calcul de UB4 pour le cas quelconque.

Début

|α| : représente le cardinal de la sous-séquence α.

n – α : représente le nombre de jobs qui n'appartiennent pas à α.

Pour j ϵ n - α **faire**

$p_{ij} = p_{1j} + \tau_j + p_{2j}$

Fin pour

Adaptation de NEH :

- Ordonner les tâches selon l'ordre croissant de temps d'exécution des jobs p_{ij} sur les machines.
- Créer une séquence vide S.
- On ordonnance les deux premiers jobs à la suite de la séquence α.
- Soit une séquence de jobs vide T

Tant que (T non vide) **faire**

T= T - premier élément j de T ;

Tester l'élément j à tous les emplacements dans S ;

Insérer j dans S à l'emplacement qui minimise le makespan.

Fin tan que

Fin NEH

Fin

Exemple III.9

Cet exemple illustre le calcul de la borne supérieure UB4. Considérons l'instance suivante pour n = 5.

Jobs	**1**	**2**	**3**	**4**	**5**
Temps d'exécution p_{1j}	13	12	2	1	1
Temps d'exécution p_{2j}	8	8	7	6	1
Temps de latence τ_j	9	7	2	1	0

Tableau III.15. Les temps d'exécution et de latence de l'Exemple III.9.

Soit une sous séquence α = (3,4). Les jobs restants n'appartenant pas à α sont : l, 2 et 5. On doit tout d'abord calculer le temps d'exécution de chaque job sur les deux machines : $p_{ij} = p_{1j} + \tau_j + p_{2j}$. On aura alors le tableau suivant :

	$j \in \alpha$		$j \in n - \alpha$		
Jobs	**3**	**4**	**1**	**2**	**5**
Temps d'exécution p_{1j}	2	1	13	12	1
Temps d'exécution p_{2j}	7	6	8	8	1
Temps de latence τ_j	2	1	9	7	0
$p_{ij} = p_{1j} + \tau_j + p_{2j}$	/////////	/////////	30	27	2

Tableau III.16. Application de UB4 sur l'Exemple III.9 dans le cas quelconque.

On ordonnance les jobs n'appartenant pas à α suivant l'ordre croissant du temps p_{ij}. La séquence sera la suivante : 5, 2 et 1. Après avoir ordonnancé les jobs, on doit placer les jobs appartenant à α ainsi que les deux premiers jobs des n - α jobs restants. C'est-à-dire, on place les jobs 3 et 4, ensuite les jobs 5 et 2, tout en respectant l'ordre. La Figure III.6 illustre cet ordonnancement.

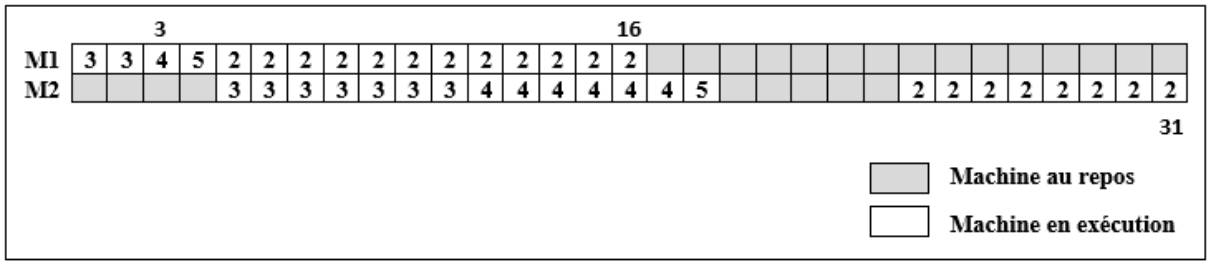

Figure III.9. Ordonnancement d'une partie de la séquence en appliquant UB4.

On applique notre algorithme sur le reste des jobs non encore ordonnancées. Le job restant est le job 1. Ce dernier va être placé dans trois différentes positions, comme illustré dans la Figure III.7. À partir de là, on doit choisir l'ordonnancement qui génère le plus petit makespan.

On a trois différents ordonnancements, (a), (b) et (c). L'ordonnancement qui a le Makespan minimal est l'ordonnancement (b) ou (c), puisqu'ils ont le même makespan. Pour conclure, nous allons choisir par défaut le premier ordonnancement, (b). La valeur de la borne UB4 est égale à 44.

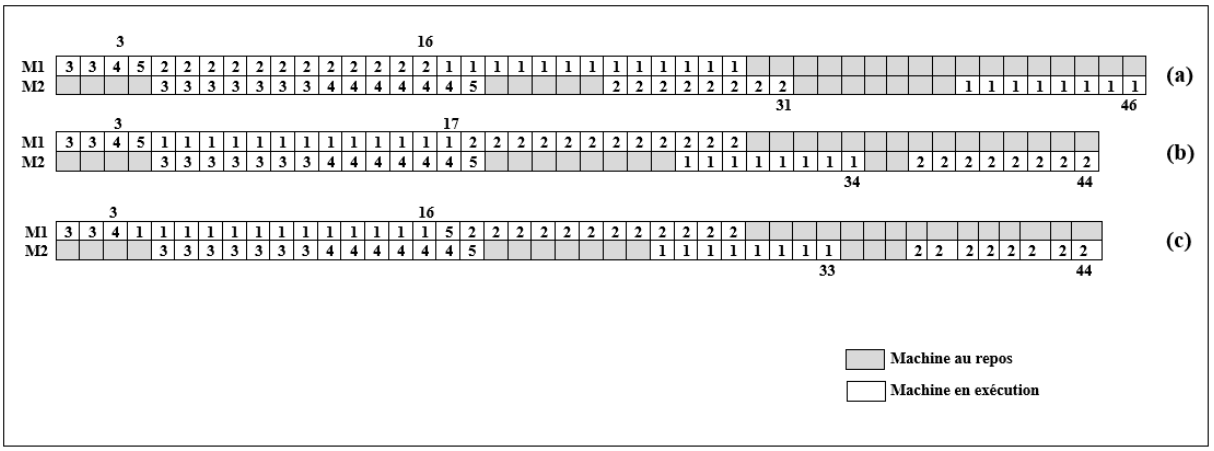

Figure III.10. Les différents ordonnancements induits par l'application de UB4.

5.3. Illustration de l'algorithme de Branch and Bound

Appliquons l'algorithme de Branch and Bound sur l'exemple ci-dessous pour n = 5, où les temps de latence et d'exécution sur *M1* et *M2* de jobs sont résumés comme suit :

Jobs	**1**	**2**	**3**	**4**	**5**
Temps d'exécution p_{1j}	13	12	2	1	1
Temps d'exécution p_{2j}	8	8	7	6	1
Temps de latence τ_j	9	7	2	1	0

Tableau III.17. Les temps d'exécution et de latence d'un problème de Branch and Bound.

- On calcule d'abord la valeur de LB 1 = 31.

- Les différentes bornes supérieures initiales sont : UB1 = 46, UB2 = 46, UB3 = 49, UB4 = 44.

- On fait appel au Branch and Bound. On obtient alors une séquence minimale : 3, 1,4, 2 et 5et un Makespan = 43, (Figure III.8).

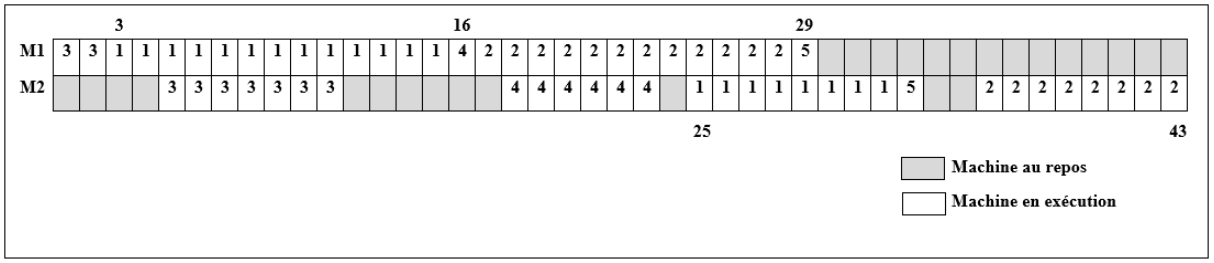

Figure III.11. Exécution de l'algorithme Branch and Bound sur une séquence de jobs dans le cas quelconque.

En appliquant l'algorithme général de Branch and Bound sur notre problème, on obtient l'Arbre de la Figure III.9. Dans cet arbre, LB représente la valeur de la borne choisie entre LB2, LB3 et LB4. Les cases grisées de cet arbre représentent la séquence optimale.

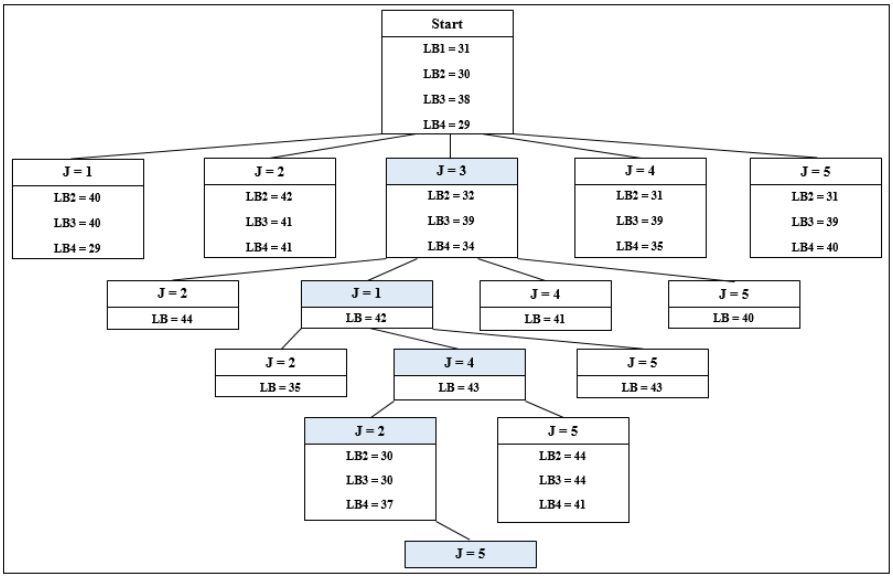

Figure III.12. L'arbre de recherche dans le cas quelconque.

6. Conclusion

Dans ce chapitre, nous avons présenté les techniques utilisées pour résoudre le problème de flow-shop à deux machines avec des temps de latence, après avoir les données et les différentes propriétés d'un problème de flow shop.

L'objectif était de faire une comparaison statistique très poussée sur les résultats des deux méthodes pour parler de la supériorité d'une méthode sur l'autre mais des contraintes de temps ne nous ont pas permis de le faire.

Chapitre IV

« Conception et implémentation du problème »

1. Introduction

Dans ce chapitre, nous allons passer à l'implémentation du problème par les deux algorithmes étudiés dans le chapitre précédant. Dans un premier temps, nous présentons langage et l'environnement de développement que nous avons utilisé. Ensuite, on a présenté l'application. Après Nous avons procédé quelques exemples pour tester.

2. Langage de programmation et environnement de développement

Nous avons implémenté notre modèle en utilisant le langage JAVA car, il est considéré parmi les meilleurs langages qui prennent en charge la véritable programmation orientée objet (OOP), il utilise des processus qui augmentent les performances des entrées/sorties, facilitent l'internationalisation. En plus, il examine le programme au fil de l'exécution et libère automatiquement la mémoire. Cette fonctionnalité diminue les risques de panne du programme et évite la possibilité de mal utiliser la mémoire.

Notre travail est écrit dans L'environnement de développement NetBeans version 8.2 sous Windows 10 Professionnel. Et nous avons utilisé comme élément matériel un ordinateurs HP qui possède les propriétés suivant :

- Un processeur Intel (R) Core (TM) i3-6006U CPU@ 2.00GHz 2.00 GHz

- Une mémoire vive de 4Go.

- Système d'exploitation 64 bits.

2.1. Le langage Java

Le langage Java est un langage de programmation orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy, cofondateur de Sun Microsystems.

Java a été officiellement présenté le 23 mai 1995 au SunWorld. La société Oracle racheta alors la société Sun en 2009, ce qui explique pourquoi ce langage appartient désormais à Oracle. La particularité et l'intérêt de Java réside dans sa portabilité entre les différents systèmes d'exploitations tels que Unix, Windows, ou MacOS.

Un programme développé en langage Java, peut ainsi s'exécuter sur toutes les plateformes, grâce à ses frameworks associés visant à garantir cette portabilité [D].

2.2. L'environnement NetBeans

NetBeans est l'environnement de Développement Intégré (EDI) supporté par SUN. Il est particulièrement bien adapté pour le développement d'applications WEB. Il remplace l'IDE Java Studio Creator. C'est un IDE moderne offrant un éditeur avec des codes couleurs et un ensemble de signes, des modèles de projets multi-langage et de différents types (application indépendante,

distribuée, plugin, mobiles, ...), le refactoring, l'éditeur graphique d'interfaces et de pages web pour supporter le programmeur dans son travail.

Il permet d'accéder rapidement à la documentation détaillée, de naviguer dans les sources et de faire des recherches d'usage des classes, méthodes et propriétés. NetBeans indique à l'utilisateur les erreurs et fait des propositions pour y remédier.

Un débuggeur permet l'exécution pas à pas. Un suivi des ressources utilisées (CPU, mémoire) par le logiciel développé peut être fait via un profiler. Un framework de test unitaire tel que Junit Fiche Junit peut être utilisé [E].

3. Le Flow Shop à deux machines avec des temps de latence

Dans le cas de Flow Shop à deux machines avec des temps de latence, toute tâche visite chaque machine de l'atelier et l'ordre de passage d'une tâche sur les deux machines est le même pour tous les tâches (travaux). Cet ordre, ou gamme, unique est une donnée de problème.

Donc, la résolution du problème de Flow Shop consiste à déterminer l'ordre de passage des jobs sur l'ensemble des machines ainsi que les instants de début et de fin des opérations des jobs afin de réduire au minimum le temps total d'exécution de tous les jobs Cmax (makespan).

Rappelons que dans le cas d'un problème de permutation avec des temps de latence avec des temps d'exécution unitaires ou quelconques est NP-difficile.

4. Les méthodes utilisées pour la résolution de notre problème

Pour résoudre notre problème nous avons utilisé deux méthodes, une méthode heuristique par une version modifiée dans le cas d'un problème de permutation avec des temps de latence de l'algorithme de Johnson. Et une méthode exacte par l'algorithme de Branch and Bound dans le cas de temps d'exécution quelconques.

5. L'Architecture générale de la réalisation

Nous avons présenté dans le schéma suivant (Figure IV.1) tous les procédures que nous avons créées pour réaliser notre résolution.

Ci-dessous, nous fournirons une explication détaillée de chaque procédure.

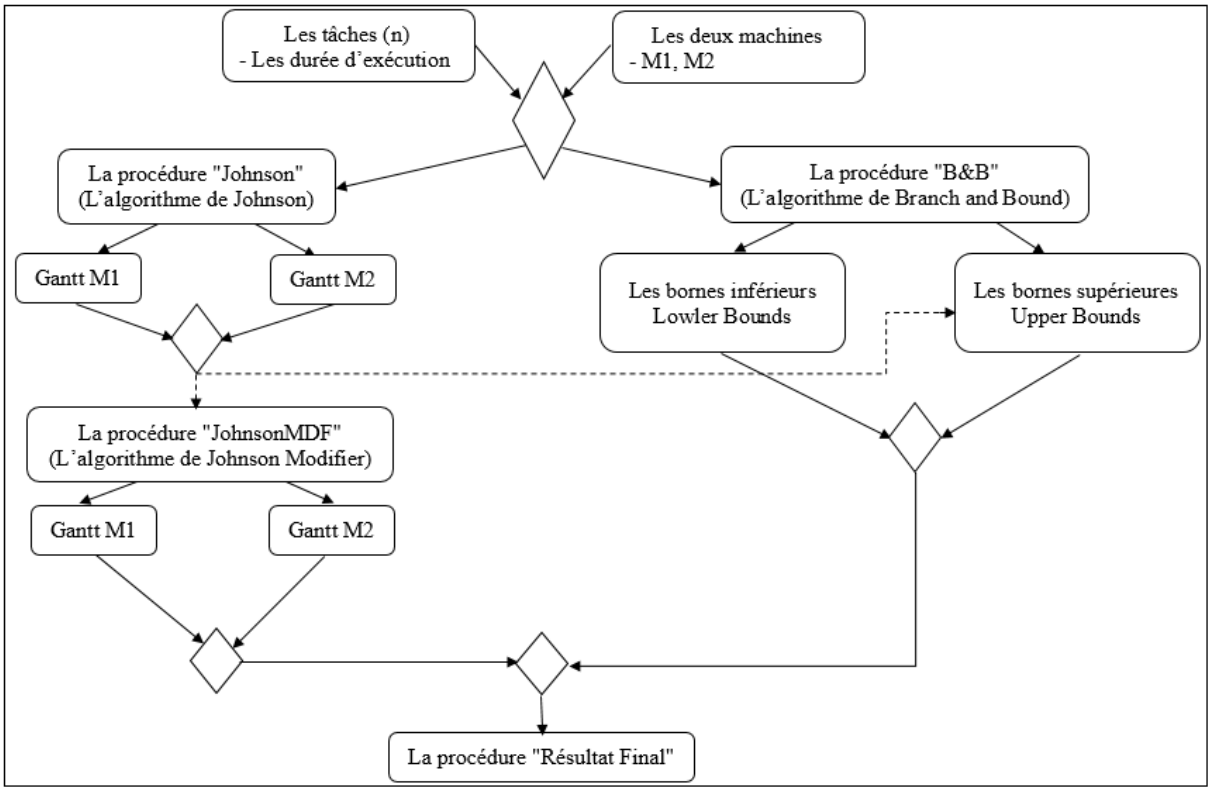

Figure IV.1. Le schéma de réalisation de problème.

Nous avons deux machines, Nous entrons le nombre de tâches à remplir dans le tableau :

1. La procédure "Johnson" : appliquons l'algorithme de Johnson au tableau pour nous donner l'ordre final des tâches et à partir de celui-ci produit le diagramme de Gantt pour la Machine 1 et le diagramme de Gantt pour la Machine 2 afin d'obtenir le temps de latence pour chaque tâche.
2. La procédure "JohnsonMDF" : Nous avons besoin du temps de latence résultant de la procédure "Johnson" pour remplir le nouveau tableau et lui appliquer un algorithme JohnsonMDF pour nous donner l'ordre final des tâches et à partir de celui-ci produit le diagramme de Gantt pour la Machine 1 et le diagramme de Gantt pour la Machine 2 afin d'obtenir Makespan (C_{max}).
3. La procédure "B&B" : Il a deux bornes :
   - Les bornes inférieures (Lowler Bounds) : Nous avons besoin des deux tableaux précédents (le premier tableau et le tableau résultant de la procédure "JohnsonMDF") et appliquons certaines opérations pour obtenir Makespan LB.
   - Les bornes supérieures (Upper Bounds) : Dans la première, nous entrons dans la sous-séquence α, et nous avons également besoin du temps de latence résultant de la procédure "Johnson", puis nous appliquons quelques opérations pour obtenir le Makespan UB. En fin de compte, nous choisissons la minimum valeur de Makespan LB et Makespan UB, et nous obtiendrons Makespan de B&B.
4. La procédure "Résultat Final" : À ce stade, nous comparons le Makespan résultant de la procédure "JohnsonMDF" et le Makespan résultant de la procédure "B&B" en prenant

le minimum entre eux pour obtenir le Makespan optimal et l'exprimer dans la conclusion.

6. Illustration de l'application

Notre application se compose de trois fenêtres d'affichage, la première représente la page d'accueil qui permet de choisir le nombre des jobs (Figure IV.1, Figure IV.2), et la deuxième représente l'interface qui permet de saisir les jobs et leur temps d'exécutions dans les deux machines (Figure IV.2), et la troisième représente l'interface de l'implémentation de problème.

Figure IV.2. La page d'entrée de l'application.

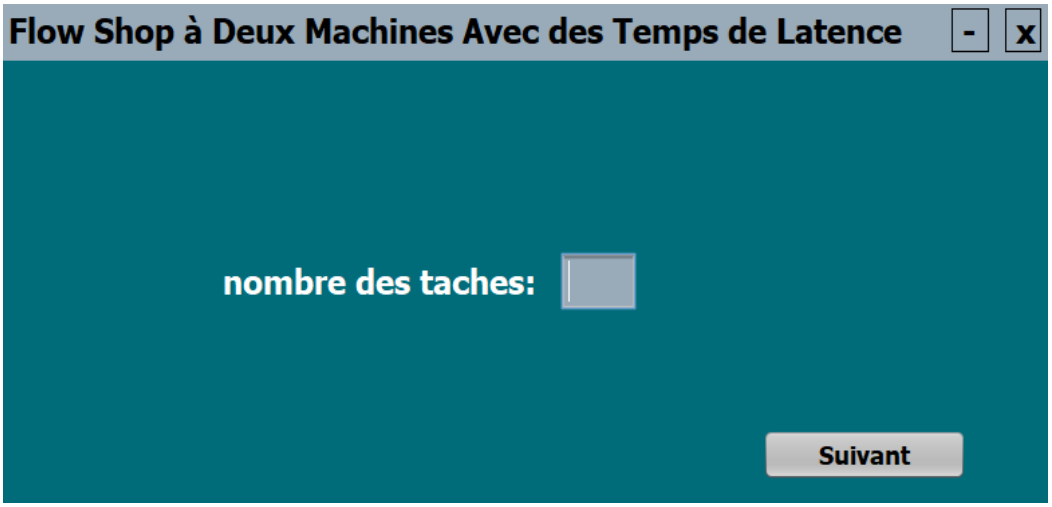

Figure IV.3. La page qui permet de choisir le nombre des jobs.

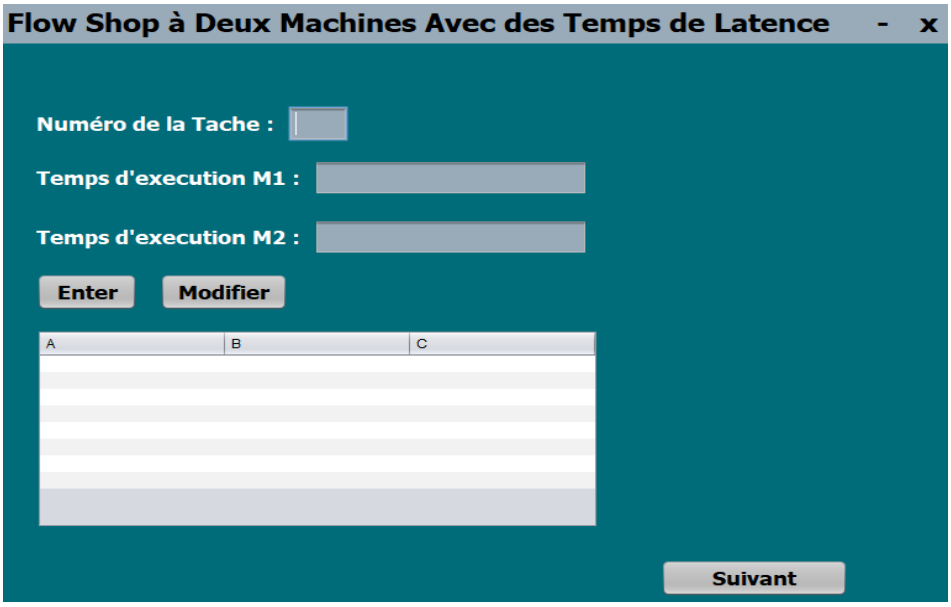

Figure IV.4. La page de l'interface.

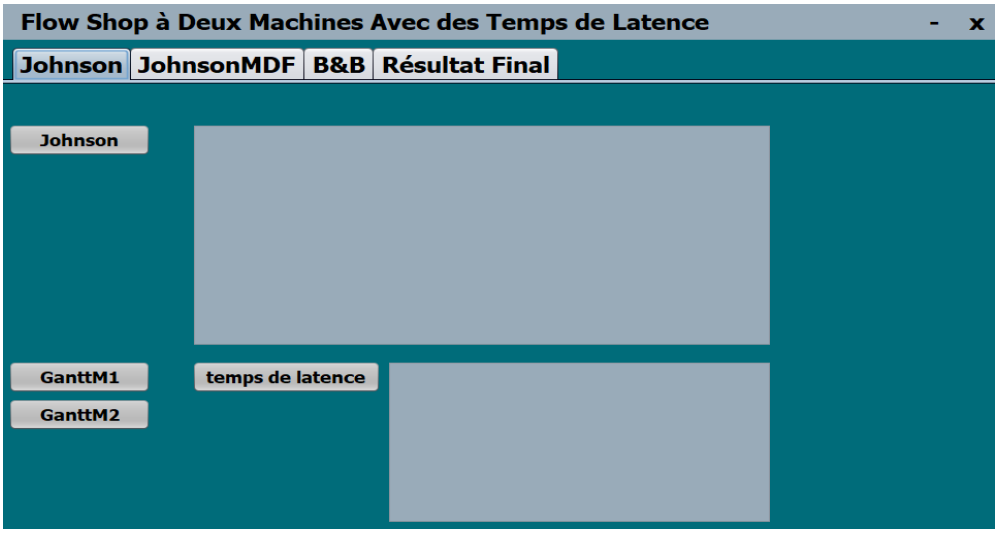

Figure IV.5. L'interface de l'implémentation de problème.

7. Quelque exemple sur l'implémentation

Exemple IV.1 : Dans un problème de flow-shop à deux machines et 4 jobs tels que les données :

Machines \ Jobs	J0	J1	J2	J3
M1	1	3	8	5
M2	4	2	7	6

Tableau IV.1. Temps d'exécution du l'Exemple IV.1.

Nous appliquons d'abord l'algorithme de Johnson pour obtenir les temps de latence (Figure IV.6) à partir des deux diagrammes de Gantt (Figure IV.4, et Figure IV.5).

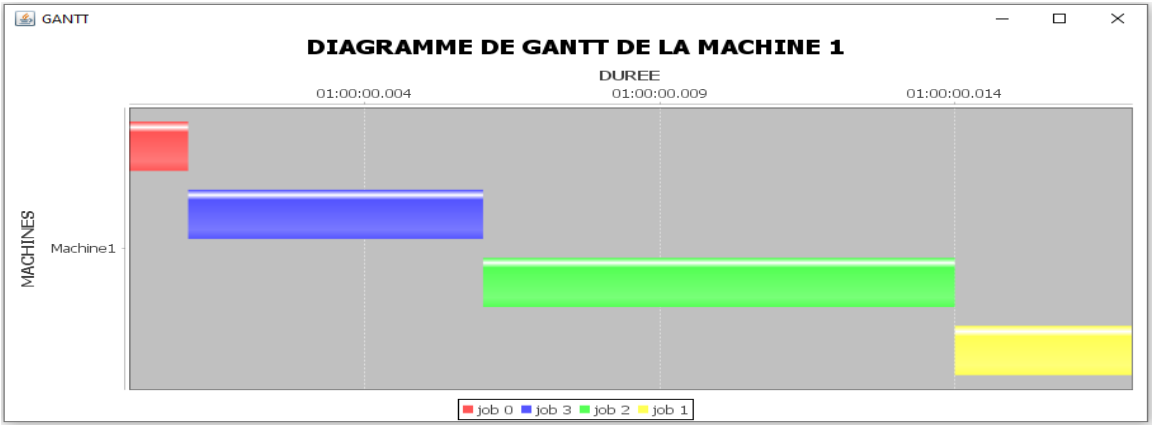

Figure IV.6. Diagramme de Gantt de la machine 1.

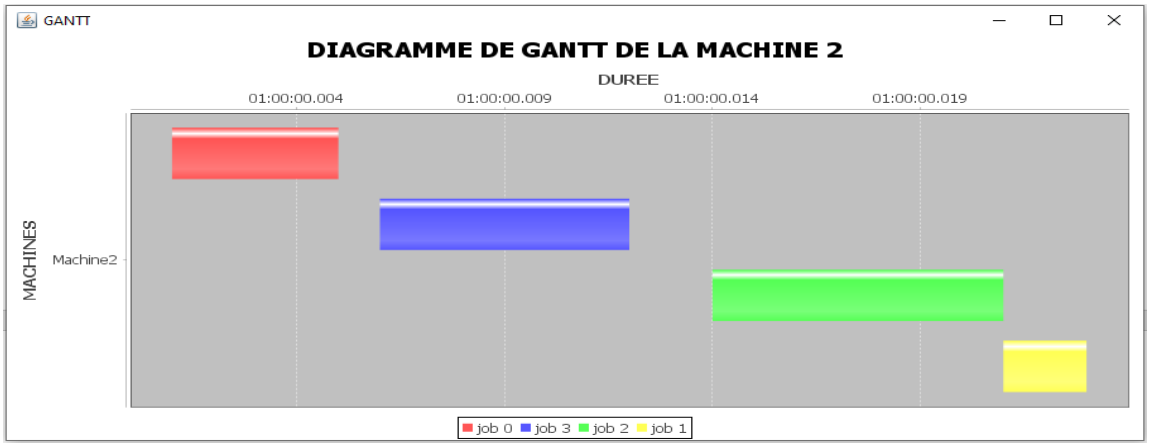

Figure IV.7. Diagramme de Gantt de la machine 2.

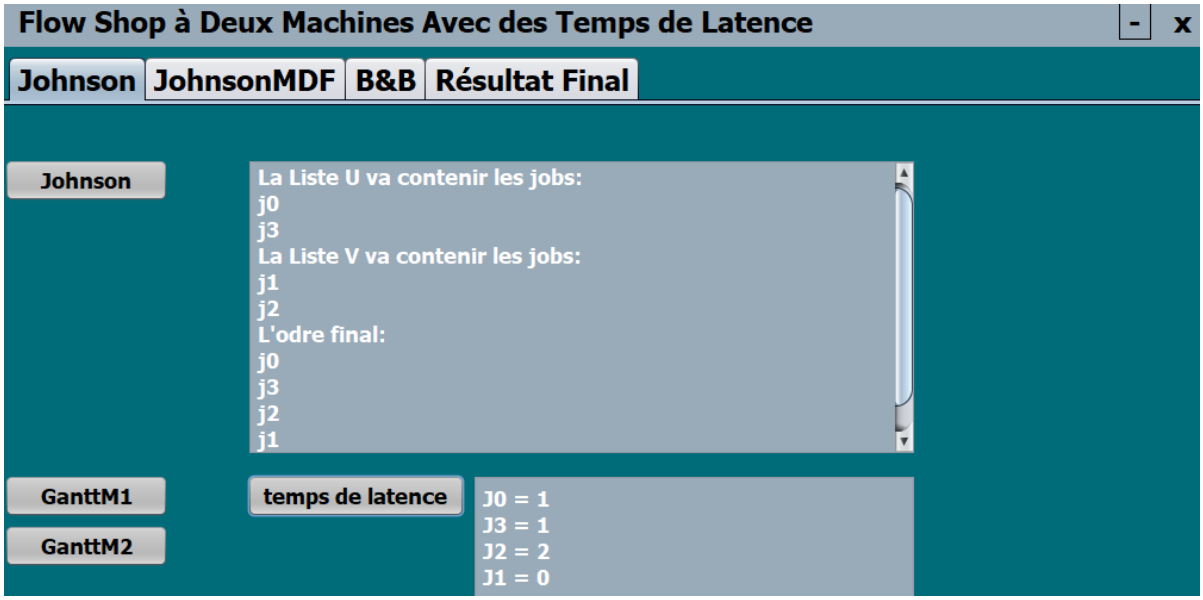

Figure IV.8. L'application de l'algorithme de Johnson et l'obtention de temps de latence.

Ensuite, nous appliquons l'algorithme de Johnson modifié (Figure IV.7, Figure IV.8 et Figure IV.9) et l'algorithme de Branch and Bound (Figure IV.10, Figure IV.11), et affichons leurs résultats, et enfin nous affichons le Makespan optimal dans une conclusion (Figure IV.12).

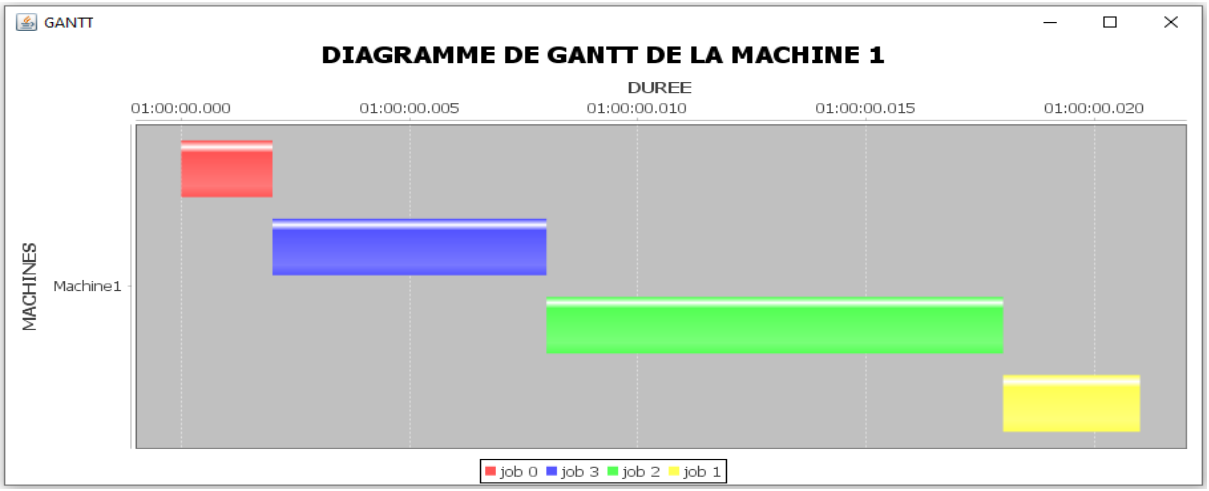

Figure IV.9. Diagramme de Gantt de Johnson modifié de la machine 1.

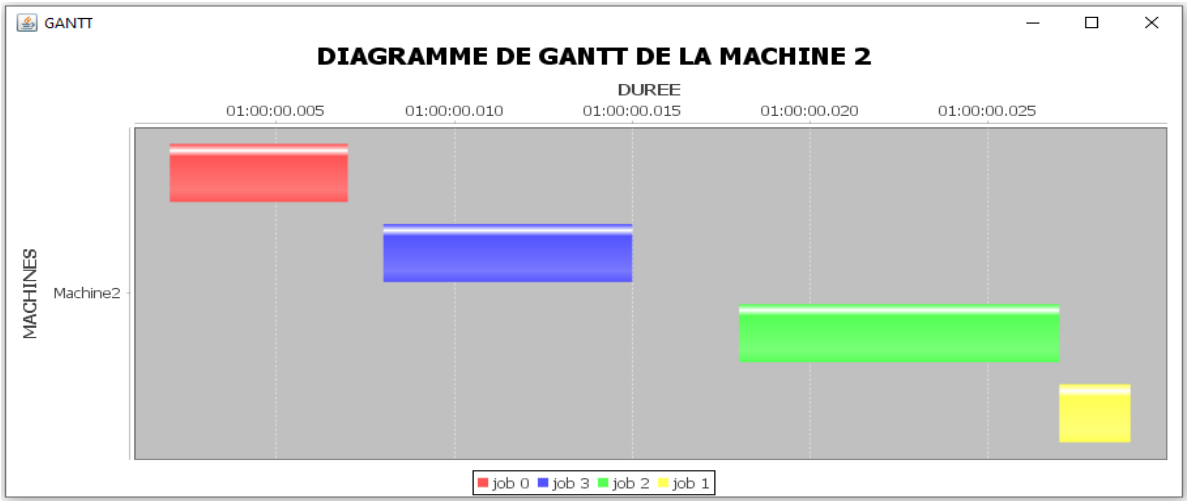

Figure IV.10. Diagramme de Gantt de Johnson modifié de la machine 2.

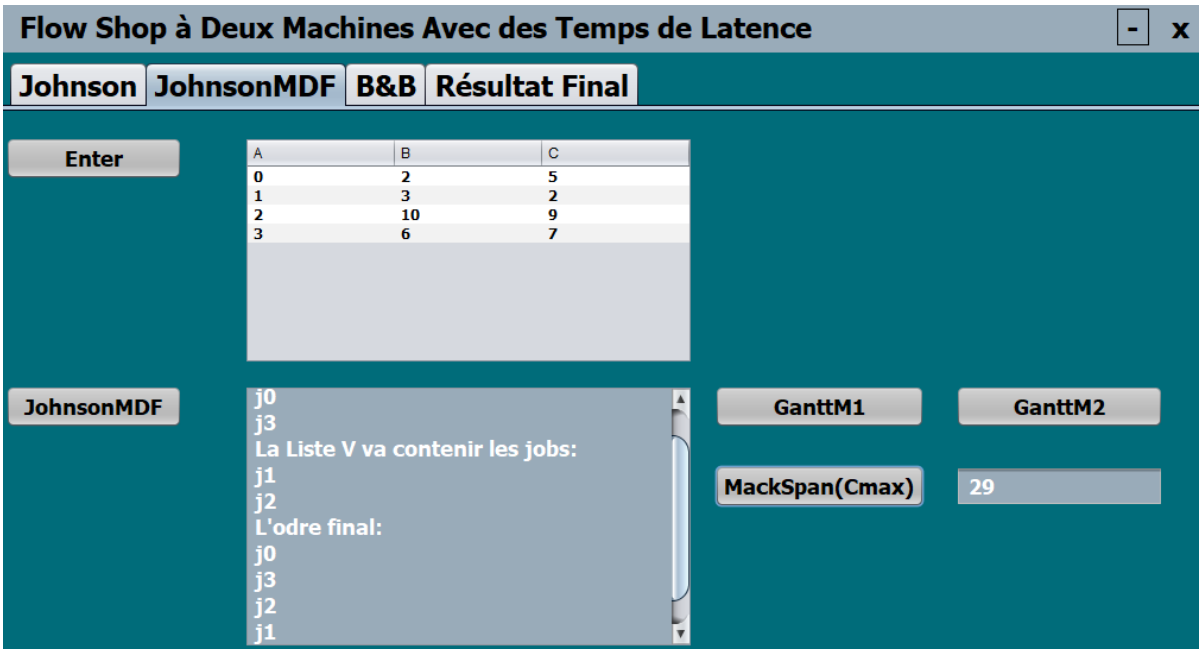

Figure IV.11. Les résultats de Johnson modifié.

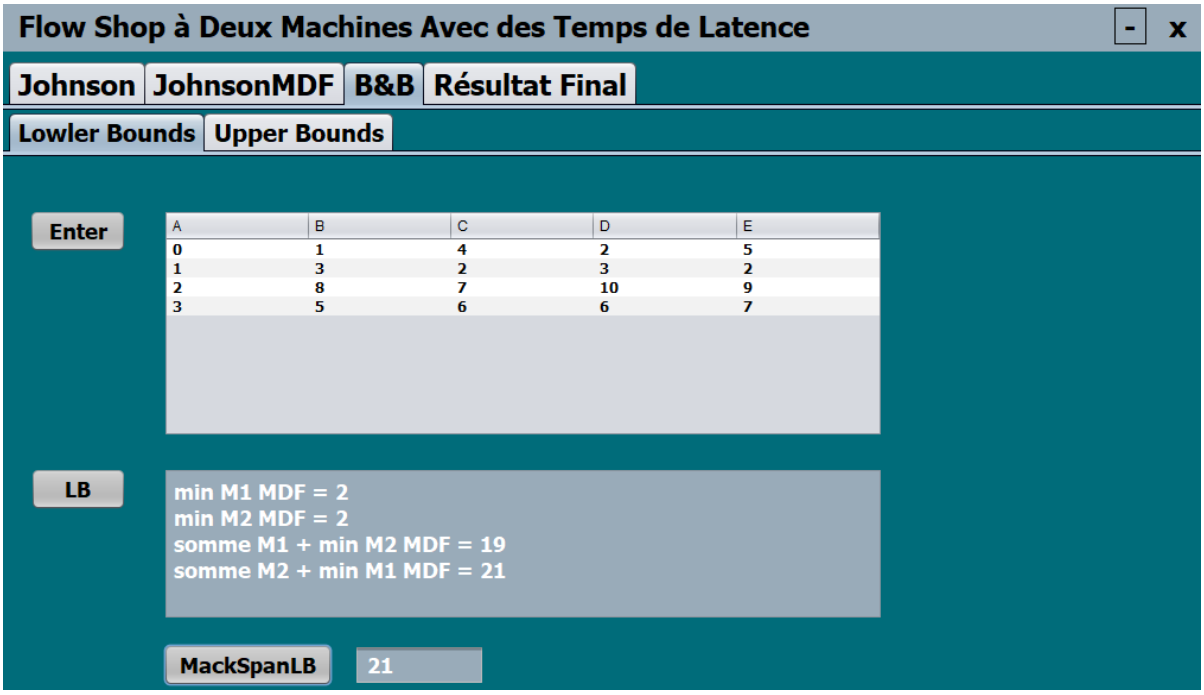

Figure IV.12. Les résultats de LB (Lowler Bound) de Branch and Bound.

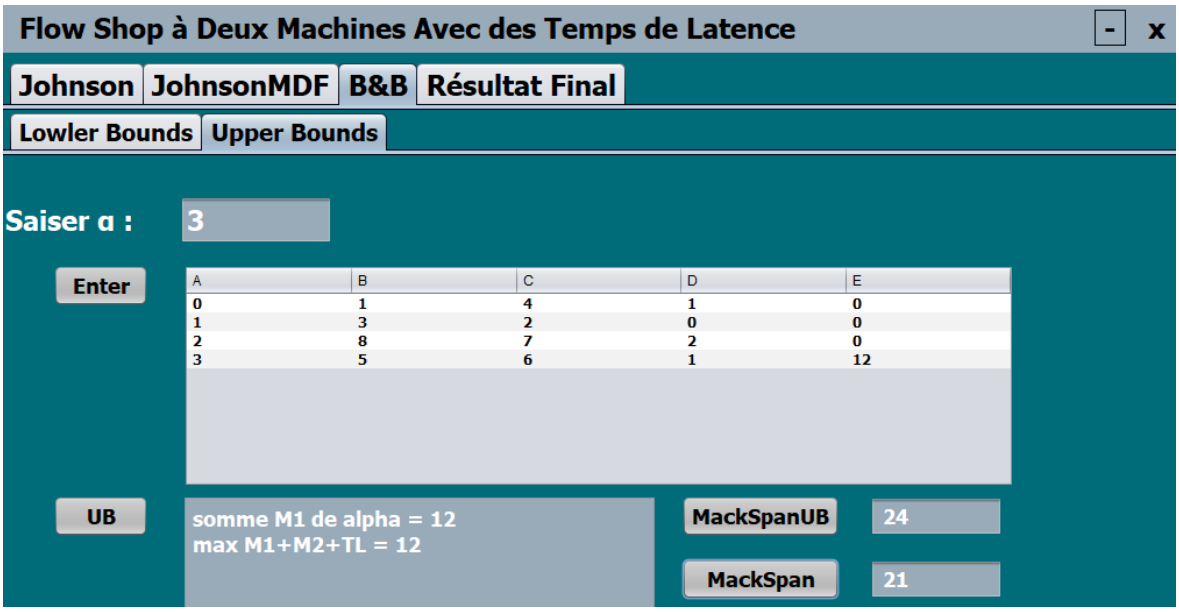

Figure IV.13. Les résultats de UB (Upper Bound) de Branch and Bound.

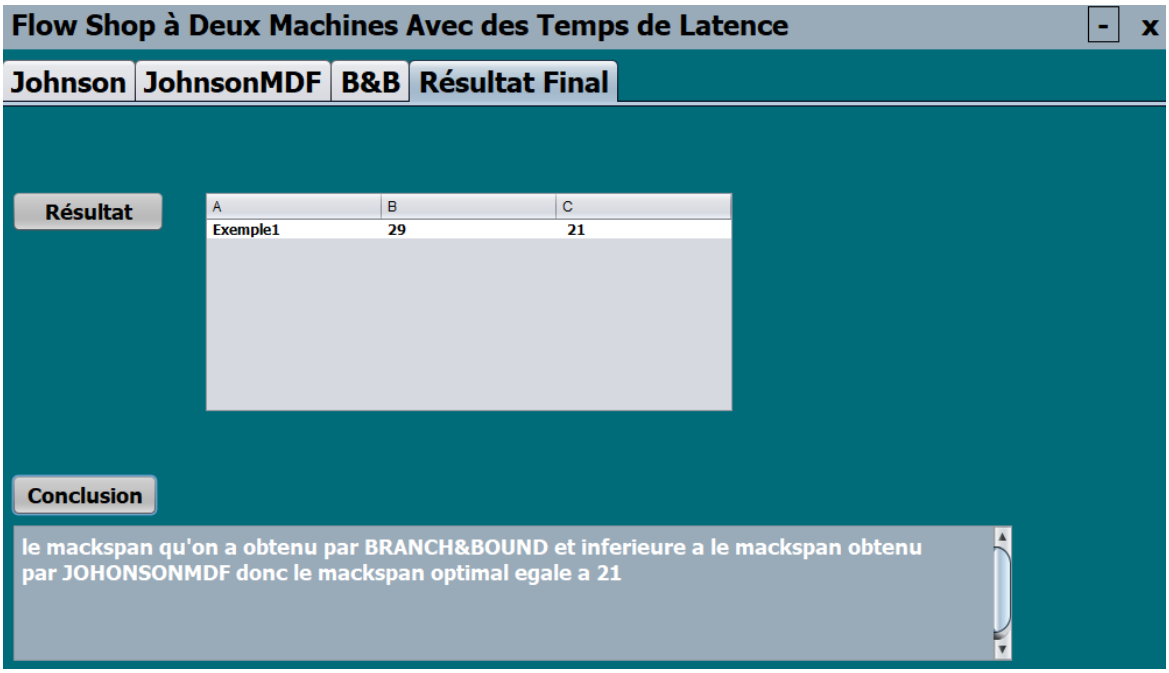

Figure IV.14. Les résultats finals et le Makespan optimal.

Exemple IV.2

Soit l'instance ci-dessous avec n = 5 jobs.

Machines \ Jobs	J0	J1	J2	J3	J4
M1	24	61	22	21	13
M2	4	19	18	16	23

Tableau IV.2. Temps d'exécution du l'Exemple IV.2.

Avec notre implémentation nous avons la solution suivante :

- L'ordre final de Johnson : J4, J1, J0, J2, J3.

- Les temps de latence : J0=15, J1=38, J2=15, J3=13, J4=13.

- L'ordre final de Johnson modifié : J4, J1, J0, J2, J3.

- Le Makespan de Johnson modifié : C_{max} = 243.

- Les bornes inférieures (LB) et supérieurs (UB avec α=3) de Branch and Bound : LB=160, UB=156.

- Le Makespan de Branch and Bound: C_{max} = 156.

Exemple IV.3

Soit l'instance ci-dessous avec n = 6 jobs.

Machines \ Jobs	J0	J1	J2	J3	J4	J5
M1	4	15	14	10	19	16
M2	9	14	16	17	21	23

Tableau IV.3. Temps d'exécution du l'Exemple IV.3.

Avec notre implémentation nous avons la solution suivante :

- L'ordre final de Johnson : J0, J3, J2, J5 J4, J1.

- Les temps de latence : J0=4, J1=0, J2=0, J3=1, J4=0, J5=0.

- L'ordre final de Johnson modifié : J0, J3, J2, J5 J4, J1.

- Le Makespan de Johnson modifié : C_{max} = 113.

- Les bornes inférieures (LB) et supérieurs (UB avec α=4) de Branch and Bound :LB=108, UB=83.

- Le Makespan de Branch and Bound: C_{max} = 83.

Exemple IV.4

Soit l'instance ci-dessous avec n = 7 jobs.

Machines \ Jobs	J0	J1	J2	J3	J4	J5	J6
M1	9	7	6	10	13	11	4
M2	7	8	4	5	12	14	15

Tableau IV.4. Temps d'exécution du l'Exemple IV.4.

Avec notre implémentation nous avons la solution suivante :

- L'ordre final de Johnson : J6, J1, J5 J4, J3, J0, J2.

- Les temps de latence : J0=4, J1=0, J2=0, J3=0, J4=0, J5=0, J6=0.

- L'ordre final de Johnson modifié : J1, J6, J5 J4, J3, J0, J2.

- Le Makespan de Johnson modifié : C_{max} = 76.

- Les bornes inférieures (LB) et supérieurs (UB avec α=5) de Branch and Bound : LB=71, UB=70.

- Le Makespan de Branch and Bound: C_{max} = 70.

Exemple IV.5

Soit l'instance ci-dessous avec n = 8 jobs.

Machines \ Jobs	J0	J1	J2	J3	J4	J5	J6	J7
M1	8	7	19	11	9	12	18	20
M2	16	10	22	5	13	17	6	15

Tableau IV.5. Temps d'exécution du l'Exemple IV.5.

Avec notre implémentation nous avons la solution suivante :

- L'ordre final de Johnson : J1, J0, J4, J5, J2, J7, J6, J3.

- Les temps de latence : J0=0, J1=7, J2=0, J3=0, J4=0, J5=0, J6=0.

- L'ordre final de Johnson modifié : J0, J4, J5, J1, J2, J7, J6, J3.

- Le Makespan de Johnson modifié : C_{max} = 119.

- Les bornes inférieures (LB) et supérieurs (UB avec α=6) de Branch and Bound :LB=112, UB=101.

- Le Makespan de Branch and Bound: C_{max} = 101.

Exemple IV.6

Soit l'instance ci-dessous avec n = 5 jobs.

Machines \ Jobs	J0	J1	J2	J3	J4
M1	11	13	12	18	10
M2	12	12	17	19	15

Tableau IV.6. Temps d'exécution du l'Exemple IV.6.

Avec notre implémentation nous avons la solution suivante :

- L'ordre final de Johnson : J4, J0, J2, J3, J1.

- Les temps de latence : J0=0, J1=0, J2=0, J3=0, J4=4.

- L'ordre final de Johnson modifié : J0, J2, J4, J3, J1.

- Le Makespan de Johnson modifié : C_{max} = 98.

- Les bornes inférieures (LB) et supérieurs (UB avec α=3) de Branch and Bound :LB=99, UB=109.

- Le Makespan de Branch and Bound: C_{max} = 99.

Après avoir obtenu les résultats de l'exécution de l'algorithme de Johnson modifié et de Branch and Bound sur les exemples précédents, nous les organisons dans le tableau suivant (Tableau IV.7) :

Les Exemples	Le Makespan de Johnson modifié	Le Makespan de Branch and Bound	le Makespan optimal
Exemple IV.1	29	21	21
Exemple IV.2	243	165	156
Exemple IV.3	113	83	83
Exemple IV.4	76	70	70
Exemple IV.5	119	101	101
Exemple IV.6	98	99	98

Tableau IV.7. Les makespans générés par les deux algorithmes et le Makespan optimal.

On le remarque la plupart du temps le Makespan optimal c'est le Makespan obtenu par l'algorithme de Branch and Bound.

8. Conclusion

Nous avons présenté dans ce chapitre l'application développée, le langage et ses outils utilisés pour résoudre le problème que nous avons couvert dans ce mémoire.

Ensuite, on a présenté l'implémentation de notre application et nous avons considéré des exemples avec les résultats obtenus. Nous mettons une simple comparaison entre les deux algorithmes utilisés, par rapport au Makespan (temps total d'exécution Cmax).

Conclusion générale

Les problèmes d'ordonnancement Flow Shop (les ateliers sériels) sont des ateliers à cheminement où le processus d'élaboration de produits est dit « linéaire », c'est-à-dire lorsque les étapes de transformation sont identiques pour tous les produits fabriqués.

Dans le cas de Flow Shop à deux machines avec des temps de latence, toute tâche visite chaque machine de l'atelier et l'ordre de passage d'une tâche sur les deux machines est le même pour tous les tâches (travaux). Cet ordre, ou gamme, unique est une donnée de problème. Donc, la résolution du problème de Flow Shop $F2|perm|C_{max}$, consiste à déterminer l'ordre de passage des jobs sur les deux machines ainsi que les instants de début et de fin des opérations des jobs afin de réduire au minimum le temps total d'exécution de tous les jobs C_{max} appelé le makespan. Rappelons que dans le cas d'un problème de permutation avec des temps de latence avec des temps d'exécution unitaires ou quelconques est NP-difficile.

Pour résoudre notre problème nous avons utilisé deux méthodes, une méthode heuristique par l'algorithme de Johnson que nous avons modifié selon la théorie de Mitten qui est utilisée dans le cas d'un problème de permutation avec des temps de latence. Une deuxième méthode est exacte, c'est l'algorithme de Branch and Bound dans le cas des temps d'exécution quelconques. Dans ce cas, nous nous sommes servis des bornes inférieures de YU, Wenci, et nous avons présenté des bornes supérieures, qui sont des versions modifiées de l'algorithme de Johnson, l'algorithme LPT (Longest Processing time), règles de priorité de Palmer et NEH.

Ensuite, nous avons réalisé une application informatique des deux techniques : l'algorithme Johnson modifié et la méthode de Branch and Bound et que nous avons testé sur un nombre d'exemples présentés dans ce mémoire. Les résultats obtenus sont très satisfaisants dans la comparaison entre eux, par rapport au Makespan C_{max} (temps total d'exécution). L'objectif espéré étant atteint.

Notre étude malgré sa simplicité, elle nous a permis de dire que les méthodes approchées ne sont pas toujours mauvaises. Elles présentent, dans certains cas de meilleures solutions. Par contre, les méthodes exactes, donnent des solutions qui ne sont pas entachées d'erreurs mais elles sont difficiles à appliquer et demandent beaucoup plus de temps d'exécution.

Enfin, notre travail nécessite des améliorations dans la méthode Branch and Bound, surtout par l'utilisation d'autres bornes supérieures et inférieures. Ces améliorations futures peuvent améliorer le Makespan de la méthode.

Références bibliographiques

[1] CARLIER, Jacques, CHRÉTIENNE, Philippe, VILLON, P., *et al.* Les problèmes d'ordonnancement. *RAIRO. Recherche opérationnelle*, 1993, vol. 27, no 1, p. 77-150.

[2] MOUHOUB, Nasser Eddine. *Algorithmes de construction de graphes dans les problèmes d'ordonnancement de projet*, Sétif, Thèse de Doctorat, 2011.

[3] LARIBI, Imane. Résolution de problèmes d'ordonnancement de type Flow-Shop de permutation en présence de contraintes de ressources non-renouvelables. 2018. Thèse de doctorat.

[4] GORINE, Ali. Ordonnancement des systèmes flexibles avec contrainte de blocage. 2011. Thèse de doctorat.

[5] DANTZIG, George, FULKERSON, Ray, et JOHNSON, Selmer. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 1954, vol. 2, no 4, p. 393-410.

[6] AGGOUNE, Riad. Ordonnancement d'ateliers sous contraintes de disponibilité des machines. 2002. Thèse de doctorat.

[7] CARRERA, Susana. Planification et ordonnancement des plateformes logistiques. 2010. Thèse de doctorat.

[8] BENTOUBACHE, Mohand. Nouvelle méthodes pour la résolution des problèmes de programmation linéaire sous forme canonique et à variables bornées. 2005. Thèse de doctorat. Université de Béjaia-Abderrahmane Mira.

[9] OSMAN, Ibrahim et LAPORTE, Gilbert. Metaheuristics : À bibliography. 1996.

[10] DRÉO, Johann, PÉTROWSKI, Alain, SIARRY, Patrick, *et al. Méta-heuristiques pour l'optimisation difficile*. 2003.

[11] BLUM, Christian et ROLI, Andrea. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 2003, vol. 35, no 3, p. 268-308.

[12] WOLOSEWICZ, Cathy. *Approche intégrée en planification et ordonnancement de la production*. 2008. Thèse de doctorat.

[13] GLOVER, Fred et LAGUNA, Manuel. Tabu Search. In: *Handbook of combinatorial optimization*. Springer, Boston, MA, 1998. p. 2093-2229.

[14] JEBARI, Hakim, EL AZZOUZI, Saida Rahali, et SAMADI, Hassan. Hybridation des méta-heuristiques pour la résolution de problème d'ordonnancement multi-objectif dans un atelier flow shop.

[15] TALBI, E.-G. A taxonomy of hybrid metaheuristics. *Journal of heuristics*, 2002, vol. 8, no 5, p. 541-564.

[16] DUVIVIER, David. *Étude de l'hybridation des méta-heuristiques, application à un problème d'ordonnancement de type job shop*. 2000. Thèse de doctorat.

[17] AMOUBOUDI, Ferroudja, BELAID, A., *et al. Étude comparative de quelques heuristiques pour la résolution du problème d'ordonnancement de type Flow shop*. 2017. Thèse de doctorat. Université Abderrahmane Mira.

[18] EL BAHLOUL, Sana. Flow-shop à deux machines avec des temps de latence : approche exacte et heuristique. 2008.

[19] YU, Wenci. The two-machine Flow Shop problem with delays and the one-machine total tardiness problem. 1996.

[20] NAWAZ, Muhammad, ENSCORE JR, E. Emory, et HAM, Inyong. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 1983, vol. 11, no 1, p. 91-95.

Les sites web

[A]Wikipédia,fr.wikipedia.org/wiki/Th%C3%A9orie_de_l%27ordonnancement#Les_contraintes, consulté le 10 mars 2020.

[B] Wikipédia, fr.wikipedia.org/wiki/Ordonnancement_d%27atelier, consulté le 13 mars 2020.

[C] Wikipédia, fr.wikipedia.org/wiki/Algorithme_de_colonies_de_fourmis, consulté le 7 juin 2020.

[D] SilkHhom, www.silkhom.com/langage-java-histoire-caracteristiques-popularite/, consulté le 25 juillet 2020.

[E] projet-plume, www.projet-plume.org/fiche/netbeans, consulté le 25 juillet 2020.

الملخص:

يهتم العمل المقدم في هذه المذكرة بمشكلة الجدولة في ورشة التدفق المكونة من جهازين مع أوقات الكمون.
الهدف هو العثور على تسلسل مناسب للمهام بناءً على أوقات الكمون، وذلك هو الـ makespan (وقت التنفيذ الأقصى) يُعتبر الأقل
يمكن استخدام عدة طرق لحل هذه المشكلة، حيث يمكننا إيجاد طرق دقيقة وطرق تقريبية. ومن هذا المنطلق، تهدف هذه المذكرة إلى تنفيذ خوارزمية جونسون المُعدَّلة وخوارزمية الفصل والتقييم لحل هذا النوع من المشكلات.
الكلمات المفتاحية: الجدولة، ورشة التدفق، جهازان، أوقات الكمون، الـ Makespan ، خوارزمية جونسون، خوارزمية الفصل والتقييم.

Résumé :

Le travail exposé dans ce mémoire s'intéresse au problème d'ordonnancement d'un Flow Shop à deux machines avec des temps de latence.
L'objectif est de trouver une séquence appropriée de tâches en fonction des temps de latence, de manière à minimiser le Makespan (temps d'exécution maximal).
Plusieurs méthodes peuvent être utilisées pour résoudre ce problème. En effet, nous pouvons trouver des méthodes exactes et des méthodes approchées. Et c'est dans cette optique que ce mémoire a pour but de mettre en œuvre l'algorithme de Johnson modifié et l'algorithme de Branch and Bound pour résoudre ce type de problèmes.
Mots clés : Ordonnancement, Flow Shop, deux machines, temps de latence, Makespan, algorithme de Johnson, algorithme de Branch and Bound.

Abstract:

The work presented in this thesis is interested in the scheduling problem of a two machines Flow Shop with latency times.
The goal is to find an appropriate sequence of jobs based on latency times, so as to minimize Makespan (maximum execution time).
Several methods can be used to resolve this problem. We can find exact methods and approximate methods. And it is with this in mind that this thesis aims to implement Johnson's algorithm and the Branch and Bound algorithm to solve this type of problem.
Keywords: Scheduling, Flow Shop, two machines, latency times, Makespan, Johnson algorithm, Branch and Bound algorithm.