N° d'ordre : 113/2021-C/MT

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de L'enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediène

Faculté de Mathématiques

Thèse de Doctorat

Présentée pour l'obtention du grade de Docteur

EN : MATHEMATIQUES

Spécialité : Recherche Opérationnelle et Mathématiques Discrètes.

Par : Nadia BABOU

Sujet

ORDONNANCEMENT D'ATELIER AVEC TEMPS DE PRÉPARATION

Soutenue publiquement à l'USTHB le 13/10/2021, devant les jurys :

M. CHERGUI Mohamed El-Amine	Professeur, USTHB	Président
M. BOUDHAR Mourad	Professeur, USTHB	Directeur de thèse
M. REBAINE Djamal	Professeur, UQAC (Canada)	Co-directeur de thèse
M. AIT-ZAI Abdelhakim	Professeur, USTHB	Examinateur
M. BAPTISTE Pierre	Professeur, PolyMtl (Canada)	Examinateur
M. AMROUCHE Karim	Maitre de conférence /A, Univ. Alger 3	Invité

Table des matières

Résumé

La théorie de l'ordonnancement s'intéresse aux problèmes d'allocation de ressources dans le temps pour réaliser un ensemble donné de tâches. Le nombre de ressources étant limité, la décision d'affecter une tâche à une ressource, à un instant donné, est cruciale pour atteindre le ou les objectifs fixés au départ. L'intérêt de cette théorie réside dans la richesse de l'interprétation des termes ressources et tâches. Ainsi, le terme ressource peut signifier processeurs, machines de production, énergie, radar, personnel, blocs opératoires, etc. L'interprétation des tâches peut quant à elle aller des produits manufacturiers aux processus informatiques ou physiques, en passant par des quarts de travail ou commandes à livrer. Cette multitude d'interprétations, surtout dans le monde industriel, a suscité l'intérêt de nombreux chercheurs.

La plupart des modèles de lignes de production, qu'on retrouve dans la littérature, sont supposés être automatiques. Cependant, la considération du facteur humain dans les systèmes semi-automatiques s'avère d'une grande importance, étant donné que ce n'est pas toujours possible d'avoir un système de production complètement automatisé. Du moins, ceci ne peut être assuré dans toutes les chaines de production et, dans certaines situations, le coût qui en serait engendré serait prohibitif. On s'intéresse, dans notre étude, au système OWMM (One Worker Multiple Machine) qui est largement utilisé dans les systèmes de fabrication Juste À Temps (JIT, Just In Time). Ainsi, dans ce système, un travailleur, appelé généralement un serveur, est une ressource supplémentaire autre que les machines. Le serveur s'occupe seulement de préparer les tâches sur les machines avant le début du traitement, et n'intervient pas durant cette phase de traitement. De plus, les préparations sont non-anticipatoires, c'est-à-dire que la préparation d'une tâche ne peut pas commencer sur une machine si la tâche est en cours de traitement sur une autre machine. Par ailleurs, le temps de préparation est indépendant de la séquence. Ainsi, ce temps dépend seulement de la tâche à préparer.

Les problèmes traités, dans cette thèse, sont des problèmes d'ordonnancement d'atelier avec un serveur pour la préparation des tâches. Le premier problème étudié est l'atelier à cheminement ouvert (open shop) avec deux machines. Tandis que le second est l'atelier à cheminement multiple (job shop) avec deux machines. Le critère que nous avons considéré est le makespan.

Pour chacun des deux problèmes, nous avons fait une étude de complexité approfondie, en étudiant la complexité de plusieurs cas particuliers des deux modèles, à savoir l'open shop et le job shop. Nous avons aussi développé des algorithmes de complexité polynomiale pour d'autres cas particuliers. De plus, nous avons établi des bornes inférieures pour le problème général et développé plusieurs méthodes pour sa résolution. En outre, nous avons mené une étude expérimentale approfondie pour tester les performances de ces méthodes.

Abstract

The difficult task of efficiently using scare resources in accomplishing variegated jobs has been a real challenge throughout the long history of human beings. Coping with such a challenge resulted, since 1954 with the Johnson's seminal work on flow shops, in an exciting discipline known as scheduling theory. Scheduling theory is therefore concerned with the optimal allocation, over time, of resources to activities. Problems of scheduling nature are encountered in many different fields such as computer science, production systems, project planning, etc. Thus, scheduling theory has drawn the interest of numerous researchers, so far mostly in industry.

Most models of production lines, that we can find in the literature, are supposed to be automatic. However, the consideration of the human factor in semi-automatic systems is of great importance, as it is not always possible to have a fully automated production system. At least, this cannot be ensured in all production chains and, in some cases, it would be too expensive to achieve. The present study focuses on OWMM (One Worker Multiple Machine) systems that are widely used in Just In Time (JIT) manufacturing systems. In an OWMM system, a worker, more generally referred to as a server, is an additional resource other than machines. The server only handles setting up the jobs on the machines before the processing can take place, and does not intervene in the latter. Furthermore, the set-ups are non-anticipatory and the set-up times are sequence-independent. Therefore, the preparation of a job cannot start on a machine if it is being processed on another machine. Moreover, the set-up time depends only on the job to process.

In this thesis, we considered the scheduling problem described above in two different environments : the first is the open shop and the second is the job shop environment. The criterion we focused on is the minimization of the overall completion time, known as the makespan.

For each one of the two models, we carried out an in-depth complexity study by investigating the complexity status of several particular cases. In addition, we provided polynomial time algorithms for other restricted cases. Moreover, we established lower bounds for the general problem and proposed several solving methods. We also conducted extensive experimental studies to test the performance of the proposed methods.

Dédicace

À toutes les personnes qui me sont chères.
À ma mère.

Remerciements

Cette thèse a été réalisée avec la contribution de beaucoup de personnes, que je tiens à remercier.

Je remercie mon directeur de thèse BOUDHAR Mourad, Professeur à l'Université des Sciences et de la Technologie Houari Boumediène, qui m'a encadré tout au long de cette thèse et qui m'a fait partager ses brillantes intuitions. Qu'il soit aussi remercié pour sa gentillesse, sa disponibilité permanente, ses judicieux conseils et les nombreux encouragements qu'il m'a prodigué.

J'exprime ma gratitude à mon co-directeur de thèse REBAINE Djamal, Professeur à l'Université du Québec à Chicoutimi, qui est une source d'inspiration, d'encouragement. Je le remercie pour sa patience, ses conseils et son soutien. J'ai grandement bénéficié de la richesse de ses connaissances et de sa rédaction méticuleuse, ce dont je lui suis reconnaissante.

J'adresse tous mes remerciements à CHERGUI Mohamed El Amine, Professeur à l'Université des Sciences et de la Technologie Houari Boumediène, de l'honneur qu'il m'a fait en acceptant d'être le président de jury. Je remercie AIT-ZAI Abdelhakim, Professeur à l'Université des Sciences et de la Technologie Houari Boumediène, BAPTISTE Pierre Professeur à l'École Polytechnique de Montréal, et AMROUCHE Karim, Maitre de conférence à l'Université d'Alger 3, qui ont accepté de faire patrie du jury de cette présente thèse de doctorat.

Je suis reconnaissante envers ma famille dont l'amour et le soutien constants m'ont gardé motivée et confiante. Un grand merci à mes frères et sœurs, qui me rappellent ce qui est important dans la vie. Je remercie également mes ami(e)s pour leur soutien et leurs encouragements.

Introduction générale

La théorie de l'ordonnancement est l'une des disciplines principales et populaires de la recherche opérationnelle. Dans la mesure où elle modélise et résout une grande variété de problèmes pratiques qu'on rencontre dans différents domaines. Par ailleurs, elle regorge de modèles classiques, qui fournissent une base théorique solide selon laquelle on peut structurer et concevoir de nouvelles études de recherche.

Dans un problème d'ordonnancement, on cherche à affecter un ensemble $T = \{T_1, \ldots, T_n\}$ de n tâches afin d'être traitées sur un ensemble $M = \{M_1, \ldots, M_m\}$ de m machines. Le passage de la tâche T_j pour son traitement sur la machine M_i est appelé opération, notée O_{ij}, $i = 1, \ldots, m$; $j = 1, \ldots, n$, et sa durée, appelée temps de traitement, est de $p_{ij} \geq 0$ unités de temps. Ainsi, on aura à ordonnancer une tâche sur une machine en lui attribuant une date de début et une date de fin de traitement. Par ailleurs, un ensemble de contraintes, qui dépendent de l'environnement considéré, doivent être respectées. Néanmoins, il existe des contraintes générales qu'on retrouve dans la plupart des problèmes d'ordonnancement, par exemple : une machine ne doit traiter qu'une seule tâche à la fois et une tâche ne peut être traitée que sur une seule machine à la fois. De surcroit, un ou plusieurs critères d'optimisation sont pris en compte pour évaluer la qualité d'une solution d'ordonnancement. Par conséquent, ces dates de début et de fin de traitement doivent être attribuées de sorte à minimiser ce ou ces critères, tout en respectant les contraintes.

On s'intéresse, dans notre étude, à deux environnements d'ordonnancement. Le premier est l'atelier à cheminement libre ou ouvert, dont l'appellation anglophone *open shop* est plus largement utilisée. Dans ce type d'atelier, le passage de la tâche T_j, $j = 1, \ldots, n$, sur les machines ne suit aucun ordre. Conséquemment, son traitement peut commencer sur n'importe quelle machine et suivre n'importe quel ordre. Ainsi, l'ordre de passage d'une tâche sur les machines sera défini lors de la construction de l'ordonnancement. Le second environnement est l'atelier à cheminement multiple, qu'on appelle aussi, *job shop*. Le passage de la tâche T_j, $j = 1, \ldots, n$, sur les machines dans ce type d'atelier, suit un ordre défini à l'avance. Ce dernier est appelé cheminement, par conséquent, dans un job shop chaque tâche possède son propre cheminement. Le critère d'optimisation, que nous considérons, est la minimisation du temps de fin de traitement final, appelé aussi le makespan, noté $C_{\max}$.

L'approche générale, en théorie de l'ordonnancement, est que tout temps de préparation nécessaire pour le traitement d'une tâche est compris dans le temps de traitement, ou bien est négligeable. Cette hypothèse peut refléter certaines applications réelles. Néanmoins, elle peut ne pas être valide dans d'autres cas. Ceci survient, par exemple, lorsque la préparation prend un temps considérablement long. Ainsi, dans les ateliers de peinture, un temps de séchage doit s'écouler entre le positionnement de deux couches de peinture. Ce temps de séchage est parfois bien plus long que le positionnement en soi. De ce fait, ces préparations ne sont nullement négligeables. D'autre part, si les préparations sont prises en charge par des ressources supplémentaires de capacité limitée alors on aura des contraintes sur la disponibilité de ces ressources, qui doivent être respectées lors de l'ordonnancement. Cependant, les précédentes contraintes ne sont pas assurément respectées, si l'ordonnancement est construit selon les méthodes qui résolvent les problèmes sans considération des temps de préparation.

Les problèmes traités, dans cette thèse, appartiennent à la classe de problèmes d'ordonnancement avec ressources supplémentaires pour la préparation des tâches. On s'intéresse donc à l'étude des problèmes d'ateliers à deux machines et un serveur. Des préparations indépendantes de la séquence et non-anticipatoires sont requises avant le traitement d'une tâche. Ces dernières sont faites par l'intervention du serveur. Ce genre de préparations se rencontre, par exemple, dans une chaine de production où un travailleur (serveur) procède au chargement de la tâche sur la machine et au lancement du traitement. Dans ce cas, le traitement ne peut pas démarrer sur une machine si le travailleur est occupé à préparer une tâche sur une autre machine. De surcroit, la phase de préparation d'une tâche ne peut commencer sur une machine si elle est en traitement sur une autre machine.

Le premier problème que nous traitons, est l'open shop à deux machines et un serveur. Nous rappelons que l'open shop classique à deux machines est résoluble en temps polynomial par l'algorithme Sahni et Gonzalez [27], ou l'algorithme Pinedo et Schrage [49, 50]. Cependant, nous allons voir que la version du problème avec des temps de préparation et un serveur est $\mathcal{NP}$-difficile au sens fort et ce, comme nous allons le voir plus loin, même pour des cas particuliers. Nous avons développé des algorithmes de résolution, pour les cas polynomiaux. Par ailleurs, nous avons proposé une résolution approchée et des bornes inférieures, pour le problème général. Nous avons, donc, conçu des heuristiques et une hyper-heuristique dont les performances ont été testées à travers une étude expérimentale que nous avons menée. Nous avons aussi, proposé deux formulations en programme mathématique linéaire mixte. Les résultats de l'étude de ce problème sont regroupés dans [12] et [13].

Le second problème étudié est le job shop à deux machines et un serveur. Le job shop à deux machines classique est résoluble en temps polynomial, par l'algorithme de Jackson [33]. Néanmoins, l'étude de complexité, que nous avons menée sur le job shop à deux machines et un serveur, a révélé que le problème est $\mathcal{NP}$-difficile au sens fort. Nous avons montré que deux cas particuliers du problème sont $\mathcal{NP}$-difficiles. En

outre, nous avons élaboré un algorithme polynomial pour résoudre deux autres cas particuliers. Nous avons proposé plusieurs heuristiques pour le cas général, que nous avons testées et qui sont ensuite utilisées pour obtenir les solutions initiales pour les méta-heuristiques. Nous avons, donc également, développé deux méta-heuristiques pour résoudre le problème général et fourni des bornes inférieures. Ainsi, nous avons proposé un recuit simulé et un algorithme génétique. Une batterie de tests a été réalisée, pour étudier les performances de ces méthodes. Les résultats de cette étude sont regroupés dans [14].

Le contenu de cette thèse peut être résumé comme suit. Le Chapitre 1 présente toutes les définitions et notions de base sur lesquelles s'appuie notre étude. Le Chapitre 2 introduit les problèmes que nous avons étudiés. D'abord, une présentation de la classe à laquelle appartiennent les problèmes est donnée, ainsi qu'un état de l'art de cette dernière. On conclut le chapitre par une description des deux problèmes, à savoir l'open shop et le job shop à deux machines et un serveur, suivi de l'énoncé de leurs bornes inférieures. L'étude de complexité fait l'objet du Chapitre 3, où sont présentés les cas $\mathcal{NP}$-difficiles des deux problèmes. Ensuite, les cas résolubles en temps polynomial sont présentés, dans le Chapitre 4. Le Chapitre 5 est consacré aux méthodes de résolution que nous avons élaborées pour résoudre le problème d'open shop à deux machines et un serveur, ainsi que l'étude expérimentale menée pour les tester. Au Chapitre 6 sont présentées les méthodes de résolution proposées pour le job shop, aussi bien que l'étude expérimentale que nous avons effectuée pour les tester. En guise de conclusion nous résumons le travail accompli et présentons des perspectives de futures recherches.

Avant de clore cette introduction générale, ci-dessous un récapitulatif des contributions réalisées dans le cadre de cette thèse :

- Dans le papier de conférence Babou et al. (2018) [9], et le papier Babou et al. (2021a) [12] publié dans *Theoretical Computer Science Journal*, nous avons traité le problème de l'open shop à deux machines et un serveur. Nous avons montré que trois cas particuliers du problème sont $\mathcal{NP}$-difficiles au sens fort. En outre, nous avons conçu un algorithme polynomial pour résoudre optimalement un autre cas. Pour le problème général, nous avons élaboré des bornes inférieures, ainsi que des heuristiques et une hyper-heuristique pour sa résolution.
- Dans le papier de conférence Babou et al. (2019a) [10] et le papier Babou et al. (2021b) [13] que nous avons soumis à un journal, nous avons traité le même problème. Nous avons prouvé que deux autres cas particuliers sont $\mathcal{NP}$-difficiles et nous avons conçu un algorithme polynomial pour un autre cas particulier. Nous avons aussi développé deux formulations en nombres entiers mixtes pour le problème général.
- Dans le papier de conférence Babou et al. (2019b) [11] et le papier Babou et al. (2021c) [14] qui est en phase de finalisation, nous avons traité le job shop à deux machines et un serveur. Ainsi, ces papiers contiennent les bornes inférieures, les résultats de complexité du problème, les cas résolubles en temps polynomial et finalement les méthodes de résolution.

1

Généralités

Nous présentons, dans ce chapitre, les définitions et concepts de base nécessaires à notre étude. Ce chapitre est, d'une part, une introduction, et de l'autre, un résumé de toutes les notions de base requises et nécessaires pour une meilleure compréhension du travail présenté dans cette thèse.

Nous commençons par quelques notions de complexité. Par la suite, nous énonçons les définitions de base en théorie de l'ordonnancement. Nous présentons les caractéristiques d'un problème d'ordonnancement, en utilisant la notation des trois champs, à savoir : les machines, les tâches et les critères d'optimisation. Puis, nous abordons la classification de quelques problèmes d'ordonnancement classiques et la hiérarchie de complexité existante entre ces classes. Enfin, nous décrivons, de façon globale, la méthodologie suivie dans l'étude des problèmes d'ordonnancement.

1.1 Notions de complexité

Nous définissons, d'abord, la complexité algorithmique, avant de passer à la complexité intrinsèque des problèmes dans le but d'établir une classification de la difficulté de résolution de ces derniers.

1.1.1 Complexité algorithmique

Nous présentons, ici, la complexité algorithmique et le principal outil permettant de la mesurer. Par ailleurs, on peut établir, en utilisant cet outil, une classification des algorithmes selon leur complexité.

Complexité d'un algorithme : Elle représente le nombre d'opérations élémentaires effectuées pour réaliser un calcul. Elle est en fonction de la taille des données en entrée.
Il est important de signaler que cette complexité est calculée pour les grandes valeurs de la taille des données. Ce calcul nous renseigne sur le comportement de l'algorithme dans le pire des cas, c'est à dire lorsqu'on a une grande masse de données. Pour cela, on considère le taux de croissance ou l'ordre de grandeur. On parle, dans ce contexte, de la notation Landau ou O, qui est définie comme suit.

Notation Landau (O) : Soit g une fonction définie sur $\mathbb{N}$. L'ensemble $O(g)$ est défini comme étant,

$$O(g) = \{f \text{ fonction définie sur } \mathbb{N}^+ / \exists c > 0, \exists n_0 \in \mathbb{N} \text{ tels que } \forall n \geq n_0, f(n) \leq c \times g(n)\}.$$

La notation O sert ainsi à comparer les limites en majorant une fonction. Par conséquent, la notation Landau est utilisée dans la mesure de complexité, car elle fournit une borne supérieure du cas le plus défavorable. Elle nous permet de comparer et de classer les algorithmes selon leur temps d'exécution, qui est en fonction de la taille des données en entrée.

Nous présentons, dans le Tableau 1.1.1, quelques classes de temps d'exécution et quelques exemples d'algorithmes, en commençant par les moins coûteux (en temps). Le plus coûteux (en temps) et le moins utilisé est le dernier type, c'est-à-dire la complexité exponentielle. Cette classe de complexité rend l'algorithme qui le génère inefficace puisque le temps de calcul augmente très rapidement même avec une légère augmentation de la taille des données.

1.1.2 Problème d'optimisation

On définit ci-après ce qui est un problème d'optimisation et le problème de décision correspondant.

Temps	Type de complexité	Description et exemple
$O(1)$	Constante	Le temps d'exécution, dans ce cas, ne dépend pas du nombre d'éléments en entrée. On a donc un nombre constant d'opérations. (par exemple, l'accès à un élément d'un tableau).
$O(\log n)$	Logarithmique	La durée augmente légèrement avec n. On rencontre ce genre de complexité lorsque la taille de données est divisée par un nombre constant à chaque itération. (par exemple, l'algorithme de recherche dichotomique).
$O(n)$	Linéaire	Ça décrit généralement un algorithme à une boucle de 1 à n, qui contient un nombre constant d'opérations élémentaires indépendant de n. (par exemple, recherche séquentielle d'un élément dans un tableau).
$O(n \log n)$	Linéarithmique	On a, dans ce cas, un algorithme qui effectue à chaque itération une division de la taille de données par une constante, et nous avons aussi à chaque fois un parcours linéaire de données. (par exemple, l'algorithme de tri par fusion).
$O(n^2)$	Quadratique (polynomiale)	Ceci décrit les algorithmes avec deux boucles imbriquées chacune allant de 1 à n, et dont le corps s'exécute en un temps constant (par exemple, le tri par insertion).
$O(2^n)$	Exponentielle	Ces algorithmes sont dits "naïfs" et ne sont pas efficaces. (par exemple, l'algorithme d'énumération pour le problème du sac à dos).

Tableau 1.1.1 – Exemples de complexité d'algorithmes

Problème d'optimisation : Il est défini par un ensemble dit réalisable ou admissible $\mathcal{X}$ et une fonction f, qui associe, à chaque élément de $\mathcal{X}$, un scalaire dans $\mathbb{R}$. Résoudre un problème d'optimisation revient à trouver une solution $x \in \mathcal{X}$ qui minimise ou maximise f. Une solution optimale x^*, pour un problème de minimisation, est donc définie comme suit :

$$x^* \in \mathcal{X} \text{ tel que } \forall x \in \mathcal{X}, f(x^*) \leq f(x).$$

Problème de décision : C'est un problème dont la solution est soit "oui", soit "non". Pour chaque problème d'optimisation, on a un problème de décision qui lui correspond. Le problème de décision, associé à un problème de minimisation, est comme suit :

Pour un $k \in \mathbb{R}$ donné, existe-t-il une solution $x \in \mathcal{X}$, pour laquelle $f(x) \leq k$?

Passons maintenant aux définitions principales permettant d'établir une classification des problèmes d'optimisation.

Réduction polynomiale : Un problème d'optimisation *se réduit* à un autre problème d'optimisation s'il existe un algorithme *polynomial* qui transforme toute instance du premier à une instance du second.

Machine déterministe : C'est une machine qui fait un seul calcul à la fois et qui constitue une étape élémentaire. Nos ordinateurs sont, par exemple, des machines déterministes.

Machine non-déterministe : C'est une machine avec un concept purement théorique : lorsque la machine est face à plusieurs choix dans son processus de calcul, cette dernière devine toujours le bon choix.

1.1.3 Classes de problèmes d'optimisation

Nous énonçons, dans ce qui suit, les principales classes de problèmes d'optimisation.

La classe $\mathcal{P}$: Les problèmes d'optimisation de la classe $\mathcal{P}$ (**P**olynomiaux), appelés aussi problèmes "faciles", sont les problèmes qu'on peut résoudre en temps polynomial, en utilisant une machine déterministe.

La classe $\mathcal{NP}$: Les problèmes de la classe $\mathcal{NP}$ (**N**on-déterministe **P**olynomiaux) sont les problèmes résolubles en un temps polynomial, en utilisant une machine non-déterministe.

Les problèmes $\mathcal{NP}$ peuvent être aussi définis comme étant l'ensemble de problèmes dont la vérification de la solution peut se faire en temps polynomial.

$\mathcal{NP}$-difficile : Un problème est dit $\mathcal{NP}$-difficile s'il est au moins aussi difficile que tous les problèmes dans $\mathcal{NP}$. Autrement dit, tout problème dans $\mathcal{NP}$ peut être réduit à ce problème en temps polynomial.

$\mathcal{NP}$-complet : Un problème est $\mathcal{NP}$-complet s'il est $\mathcal{NP}$-difficile et appartient à $\mathcal{NP}$.

Remarque 1.1. Si un problème se réduit polynomialement à un autre, cela signifie d'une part que résoudre l'un en temps polynomial impliquerait la résolution de l'autre. D'autre part, si l'un est $\mathcal{NP}$-complet alors l'autre est $\mathcal{NP}$-difficile.

Remarque 1.2. On dit qu'un problème est $\mathcal{NP}$-complet si on peut réduire un autre problème, reconnu $\mathcal{NP}$-complet, à ce dernier.

Remarque 1.3. Si le problème de décision associé à un problème d'optimisation est $\mathcal{NP}$-complet, alors le problème d'optimisation est $\mathcal{NP}$-difficile.

Si on prend en considération la célèbre conjecture $\mathcal{NP} \neq \mathcal{P}$ alors la classe $\mathcal{NP}$ peut être schématisée comme le montre la Figure 1.1.1.

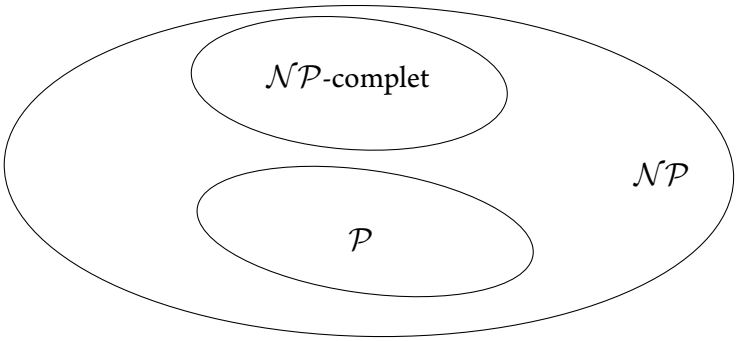


Figure 1.1.1 – Les classes $\mathcal{NP}$, $\mathcal{P}$ et $\mathcal{NP}$-complet

Algorithme pseudo-polynomial : Un algorithme est en temps pseudo-polynomial si sa complexité est polynomiale lorsque les données en entrées utilisent un codage long ou unaire, ce qui réduit artificiellement sa complexité [24].

$\mathcal{NP}$-complet au sens faible : Pour un problème $\mathcal{NP}$-complet au sens faible (ou ordinaire), il peut exister un algorithme en temps pseudo-polynomial qui le résout.

Exemple 1.1. Problème de partition [24]
Étant donné un ensemble de $2n$ entiers positifs $a_1, \ldots, a_{2n}$, avec $\sum_{j=1}^{2n} a_j = 2B$, pour un certain entier B. Existe-t-il une partition de $A = \{1, \ldots, 2n\}$ en deux sous-ensembles A_1 et A_2 tels que $\sum_{j \in A_1} a_j = \sum_{j \in A_2} a_j = B$?

$\mathcal{NP}$-complet au sens fort : Pour un problème $\mathcal{NP}$-complet au sens fort, il n'existe pas d'algorithme pseudo-polynomial qui peut le résoudre, si $\mathcal{NP} \neq \mathcal{P}$.

Exemple 1.2. Problème de 3-partition [24]
Étant donné un ensemble de $3n$ entiers positifs $a_1, \ldots, a_{3n}$, avec $\sum_{j=1}^{3n} a_j = nB$, pour un certain entier B, avec $\frac{B}{4} < a_j < \frac{B}{2}, j \in A = \{1, \ldots, 3n\}$. Existe-t-il une partition de A en n sous-ensembles $A_\ell, \ell = 1, \ldots, n$ tels que $\sum_{j \in A_\ell} a_j = B$?

1.2 Problèmes d'ordonnancement

La théorie de l'ordonnancement est une branche importante de la recherche opérationnelle et de l'aide à la décision. Son importance et sa popularité viennent du fait qu'elle modélise et traite une large variété de problèmes réels, allant des applications dans la gestion des projets, jusqu'aux applications dans les systèmes de production (manufacturier, de biens et de service), en passant par les applications dans les systèmes informatiques. Nous définissons, dans ce qui suit, les problèmes d'ordonnancement et les notions de base utilisées dans cette discipline.

1.2.1 Définitions

Ordonnancement : Un ordonnancement est le processus de prise de décision qui permet d'affecter des tâches à des ressources ou machines pour une durée définie. Des dates de début et de fin de traitement sont donc déterminées pour chaque tâche. Celles-ci sont comptabilisées dans le calcul de la valeur du critère ou des critères à optimiser. Par ailleurs, ceci doit se faire en respectant certaines contraintes, telles que la disponibilité des tâches et des machines, les relations de précédence entre les tâches, etc.

Un ordonnancement est, par conséquent, une solution à un problème d'ordonnancement. Les problèmes d'ordonnancement sont une modélisation de problèmes qu'on peut rencontrer dans une chaîne de fabrication, un système informatique ou dans un projet, etc.

Tâche : C'est une entité de travail définie par une durée, et à laquelle on attribue une date de début et de fin de traitement. Une tâche peut être une opération dans un processus de fabrication, ou une étape dans un projet de construction. Le traitement des données, dans un système informatique, est une tâche. Une garde, que doit assurer un médecin dans un service hospitalier, est une tâche. Les interventions chirurgicales sont, aussi, considérées comme un ensemble de tâches.

Ressources ou machines : Ce sont les moyens matériels nécessaires pour la réalisation d'un travail. Une ressource peut être :

— À capacité limitée : La ressource ne peut prendre en charge qu'une seule tâche à la fois, ou bien un certain nombre limité de tâches à la fois ;
— Renouvelable : Après la fin du traitement d'une tâche, la ressource redevient disponible comme avant, avec la même capacité et la même disponibilité ;
— Non-renouvelable : À la fin de chaque traitement, la ressource n'a plus la même disponibilité. Sa capacité ou bien son temps de disponibilité peuvent diminuer.

Une ressource ou une machine peut être définie de plusieurs manières différentes, selon le problème modélisé. Dans une usine de fabrication les ressources peuvent être des machines de fabrication, des mains-d'œuvre ou des robots, etc. Lorsqu'on modélise un système informatique, la machine est, dans ce cas, un processeur. Un médecin d'un staff hospitalier est considéré comme une machine, dans une application modélisant un service hospitalier.

Critère d'optimisation : C'est une fonction qui dépend, essentiellement, des dates de fin et/ou de début de traitement des tâches. On cherche principalement à minimiser la valeur de ce critère lors de l'allocation des machines aux tâches. Ceci, tout en respectant les contraintes qu'on a sur les machines et les tâches. Un critère d'optimisation sert à évaluer la qualité d'une solution d'ordonnancement.

1.2.2 Caractéristiques

Nous détaillons ci-après les différentes caractéristiques d'un problème d'ordonnancement. Celles-ci peuvent être résumées dans les trois points suivants.

— le type de machines et leur nombre ;
— les paramètres des tâches, et les relations existantes entre elles ;
— le critère (ou les critères) d'optimisation.

Outre les précédentes caractéristiques, les hypothèses générales considérées en ordonnancement sont les suivantes :

— Une machine ne peut traiter qu'une seule tâche à la fois.
— Une tâche ne peut être traitée que sur une seule machine à la fois.

Par conséquent, les machines sont de capacité limitée, de plus, elles sont renouvelables. Ce qui signifie que, sur une machine, on a la même capacité de traitement durant toute la durée de l'ordonnancement.

Nous avons défini, ci-dessus, ce que c'est qu'un ordonnancement. Nous présentons, dans ce qui suit, les différents types de problèmes d'ordonnancement. Ces derniers peuvent être représentés par la notation à trois champs $\alpha|\beta|\gamma$ de Graham *et al.* [28]. Le premier champ, le champ α nous renseigne sur les machines et leurs caractéristiques. Le deuxième champ, le champ β est réservé aux tâches, leurs paramètres et les relations existantes entre elles. Le troisième et dernier champ, le champ γ, est réservé au(x) critère(s) d'optimisation considéré(s). Dans ce qui suit, nous listons les différents types de machines, puis les paramètres et caractéristiques des tâches, et enfin les principaux critères d'optimisation.

Types de machines

Les machines sont généralement représentées par l'ensemble $M = \{M_1, M_2, \ldots, M_m\}$ à m machines. Dans la notation décrivant un problème d'ordonnancement, nous avons un premier champ contenant le type de machines, suivi par leur nombre. Si le nombre de machines n'est pas indiqué, cela signifie qu'il est arbitraire. Il existe une grande variété de types de machines, nous mentionnons les principaux.

Une seule machine : Elle est représentée dans le premier champ par 1. Ceci est le cas le plus simple des problèmes d'ordonnancement. Il est généralement considéré comme un cas particulier d'un autre problème à plusieurs machines.

Machines parallèles : Ce sont des machines qui ont exactement la même fonction. On peut, dans ce cas, traiter une tâche sur n'importe quelle machine. Nous distinguons trois principaux types de machines parallèles, qui sont les suivants.

- Machines parallèles identiques : Si les machines ont la même vitesse de traitement alors elles sont dites machines parallèles identiques, notées *Pm* pour m machines.
- Machines parallèles uniformes : Si la vitesse de traitement diffère d'une machine à une autre et qu'il existe un coefficient de proportionnalité entre les vitesses des machines alors on parle de machines parallèles uniformes, notées *Qm*.
- Machines parallèles indépendantes : Si les vitesses des machines diffèrent et qu'il n'existe aucune relation de proportionnalité entre ces vitesses alors les machines sont dites parallèles indépendantes, et sont représentées par *Rm*.

Machines parallèles dédiées : Ce sont des machines spécialisées où chacune se charge d'un traitement distinct. L'ensemble des tâches sera ainsi partitionné en m sous-ensembles et chaque sous-ensemble

sera traité sur sa machine. Dans le cas général, nous avons un seul passage sur machine par tâche. Cependant, une tâche peut avoir à être traitée sur plusieurs machines.

Lorsqu'une tâche doit être traitée sur plusieurs machines alors on parle dans ce cas d'atelier. Il existe principalement trois types d'ateliers de machines spécialisées.

— Atelier à cheminement multiple (job shop) : Dans un job shop (Jm) chaque tâche a son propre ordre de passage sur les machines, dit cheminement, qui est défini dès le départ. De plus, une tâche peut n'avoir à être traitée que sur un sous-ensemble des machines.
— Atelier à cheminement unique (flow shop) : Dans ce genre d'atelier, les machines sont en série et l'ordre de passage sur les machines est le même pour toutes les tâches. Donc, elles ont toutes le même cheminement, qui est $M_1 \rightarrow M_2 \ldots \rightarrow M_m$. Ce type d'atelier est représenté par Fm, pour m machines.
— Atelier à cheminement libre ou ouvert (open shop) : Dans ce type d'atelier, l'ordre de passage d'une tâche n'est pas défini. Il est à déterminer lors de la construction d'une solution. Une tâche peut commencer sur n'importe quelle machine et avoir n'importe quel cheminement. Un open shop à m machines est représenté par Om.

Caractéristiques des tâches

À présent, nous passons aux tâches et leurs caractéristiques. L'ensemble des tâches est représenté par un ensemble $T = \{T_1, T_2, \ldots, T_n\}$ à n tâches. Une tâche est caractérisée par les paramètres de base suivants.

Opération : Le passage de T_j pour son traitement sur la machine M_i est dit opération et est notée O_{ij}, $i = 1, \ldots, m, j = 1, \ldots, n$.

Temps de traitement : La durée de traitement de la tâche T_j sur la machine M_i est dite temps de traitement, qui est de $p_{ij} \geq 0$ unités de temps, $i = 1, \ldots, m, j = 1, \ldots, n$.

Poids d'une tâche : Il est considéré comme étant le facteur de priorité ou l'importance de la tâche T_j, et est noté w_j, $j = 1, \ldots, n$.
D'autres paramètres et caractéristiques peuvent être considérés. Ceux-ci doivent être mentionnés dans le second champ de la notation du problème d'ordonnancement et qui sont les suivants.

Date de disponibilité : C'est la date à laquelle la tâche T_j entre dans l'atelier ; elle est notée $r_j \geq 0$, $j = 1, \ldots, n$. Avant cette date on ne peut traiter T_j sur aucune machine. Si les dates de disponibilité sont différentes de zéro alors elles seront indiquées par la notation r_j dans le champ β.

Date échue : Elle est définie comme étant la date avant laquelle la tâche T_j doit s'achever. Si la date de fin de traitement C_j dépasse la date échue d_j alors la tâche T_j, $j = 1, \ldots, n$ est considérée en retard. Si des dates échues doivent être prises en considération alors cela sera indiqué par d_j dans le champ β.

Relations de précédence : Un ordre de traitement peut être imposé entre les tâches. Dans ce cas, on a des relations de précédence qui sont représentées par un graphe de précédence. Dans ce dernier, une contrainte de précédence est modélisée par un arc. Cet arc va de la tâche qui doit être traitée en premier vers la tâche qui doit la succéder. Si une période doit s'écouler entre la date de fin de traitement et la date de début de la seconde tâche alors l'arc sera pondéré et son poids sera la durée imposée. Ces contraintes sont indiquées par *prec*, dans le second champ.

Préemption : Quand la préemption des tâches est autorisée alors il n'est plus obligatoire de garder une tâche sur une machine jusqu'à la fin de son traitement. Une tâche peut être interrompue durant son traitement et elle sera poursuivie plus tard sur cette machine. Cela est indiqué par *prmp*, dans le champ β.

Temps de préparation : Si une préparation est requise et que celle-ci est une partie distincte du traitement alors ceci sera mentionné dans le second champ en indiquant le type de temps de préparation. Nous allons voir plus de détails sur ce paramètre dans le chapitre suivant.

Critères d'optimisation

Dans le processus de construction d'une solution qui constitue un ordonnancement, on définit pour chaque tâche les dates suivantes.

Date de début de traitement : C'est la date à laquelle commence le traitement de la tâche T_j sur la machine M_i, dans un ordonnancement, et qui est notée $t_{ij} \geq 0$, $i = 1, \ldots, m$, $j = 1, \ldots, n$.

Date de fin de traitement : C'est la date à laquelle s'achève le traitement de la tâche T_j sur la machine M_i. Elle est représentée par c_{ij} et elle est égale à $c_{ij} = t_{ij} + p_{ij}$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, dans un ordonnancement sans préemption.
On définit aussi C_j comme étant la date de fin de traitement de la dernière opération de la tâche T_j, $j = 1, \ldots, n$.

Ces deux dates peuvent être comptabilisées dans la formule du critère d'optimisation. Par ailleurs, ils existent plusieurs critères d'optimisation en théorie de l'ordonnancement, nous citons quelques-uns.

Date de fin de traitement final : Elle correspond à la date de fin de traitement de la dernière tâche, et qui représente la longueur de l'ordonnancement. Il est aussi nommé le makespan, et est noté $C_{\max}$. Sa formule est,

$$C_{\max} = \max_{1 \leq j \leq n} \left\{ C_j \right\}.$$

Temps de fin de traitement total : Il représente la somme des temps de fin de traitement de toutes les tâches. Il est noté :

$$\sum C_j.$$

Temps de fin de traitement total pondéré : Il représente la somme des temps de fin de traitement des tâches et de leur poids. Il est noté,

$$\sum w_j C_j.$$

Retard total : Une tâche est en retard si $C_j > d_j$. Ainsi, elle est en retard de $C_j - d_j$ unités de temps. On note le retard de la tâche T_j par : $L_j = \max\{C_j - d_j, 0\}$.
Dans ce cas, le retard total est noté

$$\sum L_j.$$

Retard total pondéré : Comme son nom l'indique, c'est la somme des retards pondérés $w_j \times L_j$, $j = 1, \ldots, n$, qui sera noté

$$\sum w_j L_j.$$

Retard maximal : Il est noté L_{max}, et est le plus grand retard sur toutes les tâches. Il est donc égal à

$$L_{max} = \max_{1 \leq j \leq n} \{L_j\}.$$

Somme pondérée des tâches en retard : On définit d'abord l'indicateur de retard pour la tâche T_j, $j = 1, \ldots, n$, comme étant,

$$U_j = \begin{cases} 1 & \text{si } C_j > d_j ; \\ 0 & \text{sinon.} \end{cases}$$

Leur somme indique le nombre de tâches en retard. Ainsi, la somme pondérée des tâches en retard représente une pénalité unitaire sur ces dernières, et qui est égale à

$$\sum w_j U_j.$$

1.2.3 Hiérarchie de complexité

On parle de hiérarchie dans les problèmes d'ordonnancement lorsqu'en relaxant une contrainte (ex. permettre la préemption des tâches), ou en fixant la valeur d'un paramètre (ex. $d_j = 0, \forall j = 1, \ldots, n$), on obtient un autre problème. Dans ce cas, le premier problème est dit cas général et le second est son cas particulier. On retrouve cette hiérarchie dans les machines, les contraintes et les paramètres des tâches, ainsi que dans les critères d'optimisation. Par exemple, le flow shop est un cas particulier du job shop qu'on obtient lorsque les tâches ont toutes le même cheminement, qui est $M_1 \rightarrow M_2 \rightarrow \ldots \rightarrow M_m$. Les critères $C_{max}, \sum C_j, \sum w_j C_j$ sont les cas particuliers des critères $L_{max}, \sum L_j, \sum w_j L_j$, respectivement, qu'on obtient en posant $d_j = 0, \forall j = 1, \ldots, n$.

En se basant sur ce concept, on peut obtenir une chaine de réduction, selon laquelle, on peut définir cette hiérarchie. Nous présentons, ici, quelques exemples. La Figure 1.2.1 [49] représente un exemple de hiérarchie sur les machines.

La Figure 1.2.2 [49] illustre un exemple de hiérarchie par rapport aux critères optimisation.
L'importance de la hiérarchie de complexité réside dans les deux points suivants :

- Si un cas particulier est $\mathcal{NP}$-difficile alors cela implique que le cas général est aussi $\mathcal{NP}$-difficile, vu qu'il est au moins aussi difficile que son cas particulier. Cependant, si un cas général est $\mathcal{NP}$-difficile alors cela ne nous renseigne en rien sur le cas particulier.

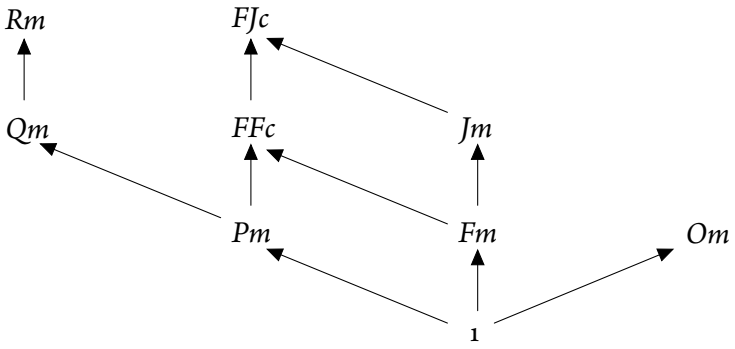


Figure 1.2.1 – La hiérarchie sur les machines

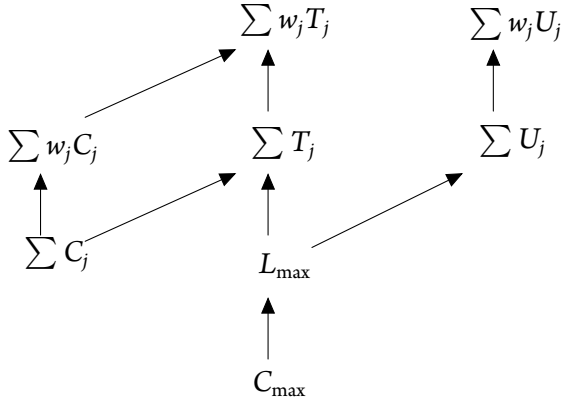


Figure 1.2.2 – La hiérarchie par rapport aux critères

- Si un cas général est polynomial alors le cas particulier l'est aussi, puisque l'algorithme qui résout le cas général le résout aussi. Néanmoins, si un cas particulier est polynomial alors cela n'implique pas forcément que le cas général l'est.

Dans la théorie de l'ordonnancement, les chercheurs ont porté un grand intérêt à bâtir la frontière séparant les problèmes polynomiaux des problèmes $\mathcal{NP}$-difficiles. Cet intérêt vient du fait que connaitre la complexité du problème soit primordial pour le choix de l'approche à utiliser afin d'aborder le problème.
Dans la Figure 1.2.3 [49] est donné un exemple de la hiérarchie de complexité existante entre quelques problèmes d'ordonnancement, avec une définition de cette frontière.

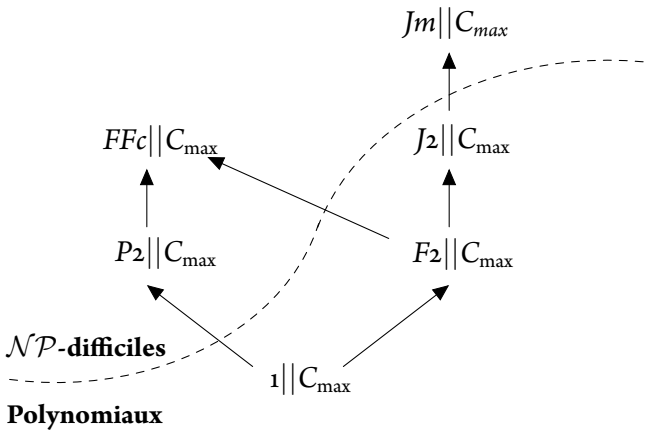


Figure 1.2.3 – La hiérarchie de complexité des problèmes

1.3 Approches de résolution

La méthodologie généralement suivie, dans l'étude d'un problème d'ordonnancement, peut être résumée en étapes, comme l'illustre le schéma de la Figure 1.3.1 [18].

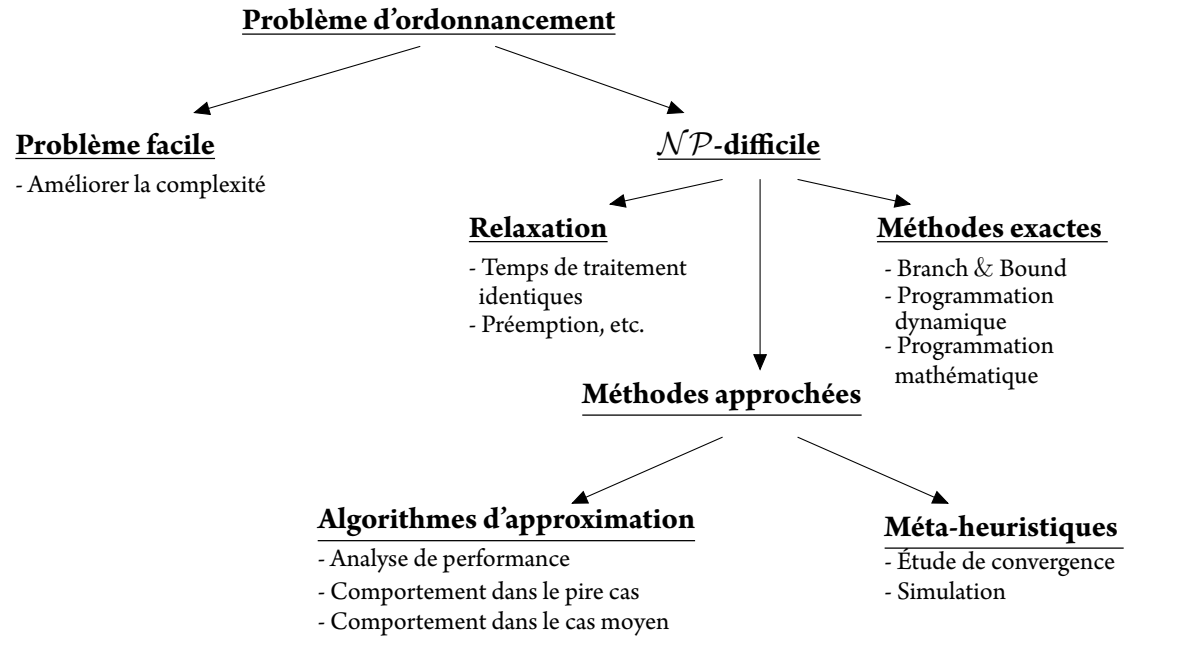


Figure 1.3.1 – Analyse d'un problème d'ordonnancement

Comme nous l'avons mentionné préalablement, la première et importante étape serait de bâtir la frontière séparant les cas résolubles en temps polynomial et ceux qui sont $\mathcal{NP}$-difficiles. Une fois la barrière bâtie, on traite chaque classe avec ses approches adéquates, que nous allons voir dans ce qui suit.

1.3.1 Problèmes faciles

La seule manière d'établir qu'un problème donné est de nature facile à résoudre est d'en exhiber un algorithme de résolution de complexité polynomiale. On en distingue plusieurs approches, parmi elles, l'approche gloutonne ou constructive.

Algorithme constructif : Il permet de construire une solution optimale du problème dans un temps polynomial. Il requiert, généralement, une bonne connaissance du problème et la structure de la solution.

Lorsqu'un problème est polynomial, on ne peut que chercher à trouver l'algorithme le moins complexe qui peut le résoudre.

Nous présentons quelques exemples de problème d'ordonnancement résoluble en temps polynomial, ainsi que l'algorithme constructif qui le résout.

Exemple 1.3. Le problème $F2||C_{max}$
Le flow shop à deux machines est résoluble à l'optimalité en $O(n \log n)$, par l'algorithme de Jonhson [34], qui est décrit dans Algorithme 1.

Algorithme 1 : Algorithme de Johnson

début
définir $U = \{T_j / p_{1j} < p_{2j}\}$ et $V = \{T_j / p_{1j} \geq p_{2j}\}$;
trier les tâches dans U par ordre croissant des p_{1j} ;
trier les tâches dans V par ordre décroissant des p_{2j} ;
la séquence de traitement des tâches est : U, V sur les deux machines ;

Exemple 1.4. Le problème $J2||C_{max}$
Le job shop à deux machines est le cas général du $F2||C_{max}$. Il est résoluble en $O(n \log n)$, par l'algorithme de Jakcson [33], qui est résumé dans Algorithme 2. Celui-ci est basé sur l'algorithme de Johnson présenté dans l'exemple précédent.

Exemple 1.5. Le problème $O2||C_{max}$

Algorithme 2 : Algorithme de Jackson

début

soient $N_1 = \{T_j/M_1 \to M_2\}, N_2 = \{T_j/M_2 \to M_1\}, N_3 = \{T_j/M_1\}$ et $N_4 = \{T_j/M_2\}$;
définir $N_1^1 = \{T_j \in N_1 \text{ t.q. } p_{1j} < p_{2j}\}$ et $N_1^2 = \{T_j \in N_1 \text{ t.q. } p_{1j} \geq p_{2j}\}$;
définir $N_2^1 = \{T_j \in N_2 \text{ t.q. } p_{1j} > p_{2j}\}$ et $N_2^2 = \{T_j \in N_2 \text{ t.q. } p_{1j} \leq p_{2j}\}$;
trier les tâches de N_1^1 et N_2^2 dans l'ordre croissant des temps de traitement sur M_1 ;
trier les tâches de N_2^1 et N_1^2 dans l'ordre décroissant des temps de traitement sur M_2 ;
la séquence de traitement des tâches sur M_1 est : $N_1^1, N_1^2, N_3, N_2^1, N_2^2$;
la séquence de traitement des tâches sur M_2 est : $N_2^1, N_2^2, N_4, N_1^1, N_1^2$;

L'algorithme, que nous énonçons ici, est nommé la règle GARD (Greatest And then RanDom), qui est décrite dans Algorithme 3. Elle résout le $O2||C_{\max}$ en $O(n)$. Par souci d'exhaustivité, nous présentons dans ce qui suit la preuve de son optimalité [1].

Algorithme 3 : La règle Greatest And then RanDom (GARD)

début

soit $p_{hk} = \max\{p_{ij} : i = 1, 2; j = 1, \ldots, n\}$;
traiter T_k sur la machine M_{3-h} ;
pour $j = 1$ *à* $n, j \neq k$ **faire**
    traiter T_j sur la première machine libre ;
traiter T_k, le plutôt possible, sur la machine M_h ;
pour $j = 1$ *à* $n ; j \neq k$ **faire**
    traiter l'opération restante de T_j le plutôt possible sur la machine correspondante ;

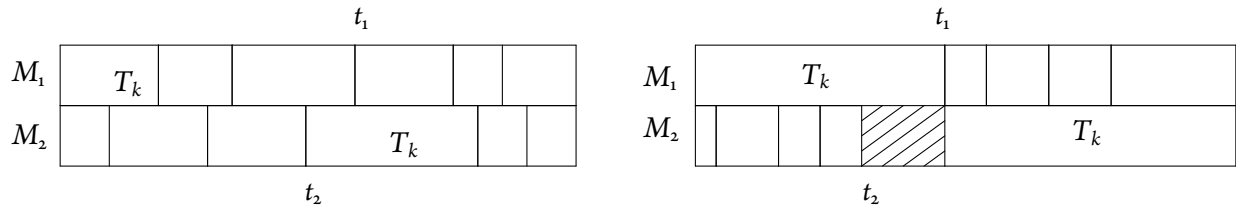
Théorème 1.1. *La règle GARD résout d'une manière optimale le problème $O2||C_{\max}$ en temps linéaire.*

Démonstration. Soient $F_i = \sum_{j=1}^{n} p_{ij}, i = 1, 2$, et $Q = \max_{1 \leq k \leq n}\{p_{1k} + p_{2k}\}$. Clairement, F_1, F_2 et Q sont des bornes inférieures du makespan. Sans perte de généralité, on suppose que $h = 2$. Soient t_1 et t_2 les temps de fin de traitement sur M_1 et M_2, respectivement, à la sortie de la première boucle *c.-à-d.,* après avoir traité les premières opérations des tâches sur les deux machines. Nous considérons les deux cas suivants :

Cas 1 : $t_1 > t_2$. D'emblée, remarquons que les secondes opérations des tâches sur M_1 peuvent être traitées directement sans temps mort. Si le nombre de tâches, traitées avant l'instant t_1, est supérieur à 1 alors, vu la nature gloutonne de l'instruction dans la première boucle et puisque la tâche T_k est la tâche la plus longue,

1. On fait remarquer qu'il existe deux versions de l'algorithme qui résolvent le problème. La première est présentée sans preuve dans [49], et la seconde est énoncée dans [50], dite LAPT, pour Longest Alternate Processing Time, dont le temps d'exécution est non-linéaire. En effet à chaque étape de cette version, la tâche la plus longue est choisie comme étant la prochaine tâche à ordonnancer. En procédant de la sorte, on obtient un algorithme en $O(n \log n)$. Hormis pour la première tâche, il n'est pas nécessaire de choisir à chaque étape le plus long temps de traitement.

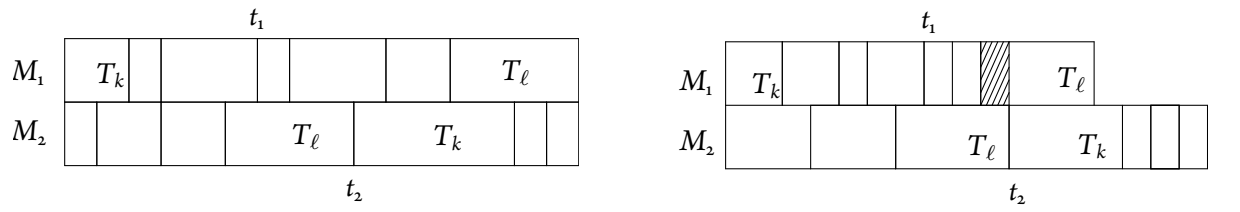
il s'ensuit que les secondes opérations peuvent être traitées sans un temps mort sur M_2 après l'instant t_2. Le makespan correspondant est donc $\max\{F_1, F_2\}$, comme illustré par la Figure 1.3.2a. Maintenant, si on suppose que ce nombre est égal à 1, alors T_k est la seule tâche traitée sur M_1 avant l'instant t_1. Dans ce cas, et comme l'illustre la Figure 1.3.2b, il y a un temps mort sur M_2, engendré par T_k car elle est toujours en cours de traitement sur M_1. Le makespan est donc $\max\{F_1, Q\}$.



(a) Aucun temps mort sur les deux machines **(b)** Temps mort sur M_2

Figure 1.3.2 – $t_1 > t_2$

Cas 2 : $t_1 \leq t_2$. Là encore, le traitement des secondes opérations des tâches traitées sur M_1, peuvent clairement être ordonnancées sans temps mort sur M_2, après l'instant t_2. Soit T_ℓ la dernière tâche traitée sur M_2, avant l'instant t_2. Supposons, d'abord, que le temps de traitement total sur M_1 des tâches traitées après l'instant t_1, sans compter T_ℓ, est supérieur ou égal à $(t_2 - t_1)$. En raison de la nature gloutonne de l'instruction dans la seconde boucle, ces tâches, y compris la tâche T_ℓ peuvent être clairement traitées continuellement sans temps mort sur M_1 après l'instant t_1. Le makespan est donc $\max\{F_1, F_2\}$, comme illustré par la Figure 1.3.3a.
À présent, si on suppose que le temps de traitement de ces tâches est inférieur à $(t_2 - t_1)$, alors T_ℓ engendrera un temps mort sur M_1, car elle est toujours en cours de traitement sur M_2. Puisque T_k est la tâche la plus longue, il s'ensuit que le makespan est F_2, comme illustré dans la Figure 1.3.3b.



(a) Aucun temps mort sur les deux machines **(b)** Temps mort sur M_1

Figure 1.3.3 – $t_1 \leq t_2$

□

Pour cet algorithme, nous considérons l'instance présentée au Tableau1.3.1. Ensuite, nous présentons la solution obtenue par la règle GARD.

T_j	T_1	T_2	T_3	T_4	T_5	T_6
p_{1j}	2	1	3	1	2	2
p_{2j}	5	3	1	3	1	4

Tableau 1.3.1 – Instance du $O2||C_{\max}$

La solution obtenue par la règle GARD est illustrée par la Figure 1.3.4. Dans cette solution $t_1 = 7, t_2 = 10$, et le $C_{\max} = 17$ correspondant, pour cette instance, au temps de traitement total de la machine M_2. Il s'ensuit que la solution produite est optimale.

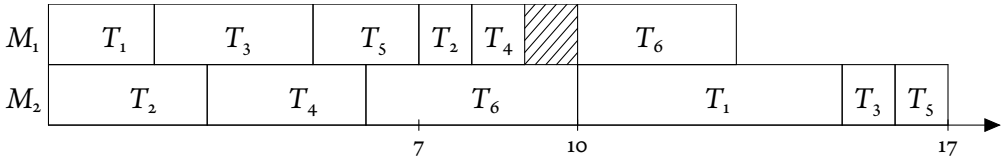


Figure 1.3.4 – Solution obtenue par la règle GARD

1.3.2 Problèmes $\mathcal{NP}$-difficiles

Lorsqu'un problème est $\mathcal{NP}$-difficile, cela signifie que la génération d'une solution optimale prend un temps prohibitif pour la majorité des instances de grande taille et parfois même pour les instances de taille moyenne. Une fois le problème étudié est prouvé $\mathcal{NP}$-difficile, nous aurons à choisir l'approche avec laquelle nous allons l'aborder, comme illustré par la Figure1.3.1. Nous avons, de façon globale, trois axes qu'on peut suivre : la relaxation du problème, la résolution exacte et la résolution approchée.

Relaxation du problème

L'une des possibilités qui s'offrent à nous pour résoudre un problème est de relaxer une ou plusieurs de ses contraintes. Par exemple, considérer le problème avec des temps de traitement unitaires, enlever des contraintes de précédences, etc. Cette manière de procéder permet de tracer la frontière entre les instances faciles et les instances difficiles.

Exemple 1.6. Le problème $Om|prmp|C_{\max}$ est polynomial
Comme on a vu dans l'Exemple 1.5, l'open shop à deux machines est résoluble en temps polynomial. Cependant, il devient $\mathcal{NP}$-difficile au-delà de trois machines [27]. En permettant la préemption des tâches, le problème correspondant, noté $Om|prmp|C_{\max}$, est résoluble en $O(n^2m^2)$ [27].

Les méthodes exactes fournissent systématiquement une solution optimale au problème. L'inconvénient de ces méthodes réside dans le temps qu'elles prennent pour obtenir la solution.

Programmation mathématique : Elle regroupe un ensemble de techniques qui permettent d'optimiser une fonction objectif, soumise à un ensemble de contraintes. Il existe une grande variété d'algorithmes dédiés à la résolution de ces modèles.

Exemple 1.7. Programmation linéaire
Un programme linéaire est un problème d'optimisation qui minimise ou maximise une fonction objectif linéaire, dont les variables sont soumises à un ensemble de contraintes d'égalités ou d'inégalités linéaires. Le modèle (M), qui suit, est une formulation en programme linéaire en nombres entiers mixtes du problème $Fm||C_{\max}$

$$
(\mathrm{M}) \left| \begin{array}{lll}
\min C & & \\
\text{s.c.} \quad \sum_{k=1}^{mn} x_{ijk} = 1; & & j = 1, \ldots, n, i = 1, \ldots, m; \\
\sum_{i=1}^{m} \sum_{j=1}^{n} x_{ijk} = 1; & & k = 1, \ldots, mn; \\
t_{ij} + p_{ij} \leq t_{i+1,j}; & & j = 1, \ldots, n, i = 1, \ldots, m-1; \\
t_{ij} + p_{ij} \leq C; & & j = 1, \ldots, n, i = 1, \ldots, m; \\
\sum_{j=1}^{n}((t_{ij} + p_{ij})x_{ijk}) \leq \sum_{j=1}^{n}(t_{ij}x_{ijk'}) + UB \times (1 - \sum_{j=1}^{n} x_{ijk'}); & & i = 1, \ldots, m, k = 1, \ldots, mn, \\
 & & k' = k+1, \ldots, mn; \\
x_{ijk} \in \{0, 1\}, t_{ij} \in \mathbb{N}^+, C \in \mathbb{N}^+; & & j = 1, \ldots, n, i = 1, \ldots, m \\
 & & k = 1, \ldots, mn;
\end{array} \right.
$$

Nous rappelons que t_{ij} et p_{ij} sont la date de début et la durée de traitement, respectivement, de la tâche T_j sur la machine M_i et UB une borne supérieure. Tandis que la variable booléenne x_{ijk} est définie comme suit :

$$
x_{ijk} = \begin{cases} 1 & \text{si } T_j \text{ sur } M_i \text{ est traitée à la position } k; j = 1, \ldots, n, i = 1, \ldots, m, k = 1, \ldots, mn; \\ 0 & \text{sinon.} \end{cases}
$$

Méthodes énumératives : Elles procèdent à un parcourt énumératif, plus au moins intelligent, de l'espace de recherche. Les points de recherche (solutions) sont parcourus partiellement ou complètement. Ceci dépend des techniques utilisées qui permettent de détecter le plus tôt possible les zones de recherche stériles.

Exemple 1.8. Programmation dynamique

La programmation dynamique est souvent basée sur l'invocation du principe de Bellman [17] pour établir une relation de récurrence servant à la résolution du problème (d'optimisation) en question. La résolution algorithmique de la relation de récurrence se fait en résolvant les sous problèmes, en commençant par ceux de taille plus petite avant d'arriver à ceux de grandes tailles. Cette façon de procéder permet d'éviter de résoudre plusieurs fois les mêmes sous problèmes. Ainsi, les temps d'exécution générés sont nettement améliorés et parfois même sont de nature polynomiale.

Exemple 1.9. Méthodes de séparation et d'évaluation
Le principe de ces méthodes est de décomposer l'espace de recherche en plusieurs sous-ensembles généralement disjoints. L'union de ces sous-ensembles couvre toutes les solutions possibles. Le principe de la séparation est récursif dont le déroulement engendre l'arbre de recherche. Les trois éléments essentiels dans les méthodes de séparations et évaluations sont :

- La méthode d'exploration : Il existe plusieurs façons d'explorer l'arbre de recherche. Les plus connues sont l'exploration en largeur d'abord (Depth First Search ou DFS) et l'exploration en profondeur d'abord (Breath First Search ou BFS).
- Les bornes inférieures et supérieures : Ces deux paramètres nous permettent de définir les zones de recherche stériles. Conséquemment, un bon calcul de ces bornes contribue à l'augmentation de la vitesse d'exploration de l'arbre de recherche.
- Les règles de dominance : Ces règles nous renseignent sur les solutions qui sont dominées et dont l'exploration est inutile.

Méthodes approchées

Les méthodes approchées permettent de trouver une ou plusieurs solutions réalisables du problème. Les deux principaux types de méthodes approchées appliquées sur les problèmes d'ordonnancement sont les heuristiques et les méta-heuristiques.

Heuristique : Le mot "heuristique" trouve ses origines dans le mot grec *heuriskein* qui signifie "trouver". Les heuristiques, dont le nom tombe sous le sens, permettent de "trouver" une solution en temps polynomial, ce qui fait qu'elles sont caractérisées par leur rapidité. Cependant, la solution n'est pas forcement optimale. De plus, par leur nature, elles sont des approches ad-hoc, dans le sens où elles sont spécifiques aux problèmes qu'elles résolvent.

Méta-heuristique : Le mot " méta " signifie " au-delà ", ce qui fait que les méta-heuristiques sont des heuristiques d'un niveau supérieur. Elles ont été développées pour combler les lacunes des heuristiques. Le

point fort des méta-heuristiques est le rapport entre le temps d'exécution et la qualité de la solution trouvée. De plus, elles ne sont pas spécifiques à un seul problème et de ce fait peuvent être utilisées pour la résolution de n'importe quel problème d'optimisation. Leur classification dépend généralement du type d'approche. On peut cependant distinguer :

— Les approches à trajectoire partent d'une solution initiale et explorent progressivement, selon une trajectoire, l'espace des solutions. Dans cette catégorie, nous avons le recuit simulé, la méthode tabou, la recherche par voisinage variable, etc.

— Les approches à population, dont le principe consiste à travailler avec un ensemble de solutions simultanément. Cet ensemble évolue graduellement. Dans cette catégorie, on retrouve : les algorithmes génétiques et les algorithmes par colonies de fourmis, pour ne citer que celles-là.

Conclusion

Nous avons répertorié, dans ce chapitre, les concepts de base nécessaires à la compréhension du travail accompli dans cette thèse. Ci-dessous un résumé des principales notations que nous allons utiliser par la suit.

$T = \{T_1, \ldots, T_n\}$:	ensemble de tâches.
$M = \{M_1, \ldots, M_m\}$:	ensemble de machines.
n :	nombre de tâches.
m :	nombre de machines.
Om :	open shop à m machines.
Fm :	flow shop à m machines.
Jm :	job shop à m machines.
O_{ij} :	opération ou passage de la tâche T_j sur la machine M_i, $j = 1, \ldots, n$, $i = 1, \ldots, m$.
p_{ij} :	temps de traitement de T_j sur M_i, $j = 1, \ldots, n$, $i = 1, \ldots, m$.
t_{ij} :	date de début de traitement de T_j sur M_i, $j = 1, \ldots, n$, $i = 1, \ldots, m$.
c_{ij} :	temps de fin de traitement de T_j sur M_i, $j = 1, \ldots, n$, $i = 1, \ldots, m$.
$C_{\max}$:	temps total d'accomplissement (makespan).

2

Positionnement des problèmes étudiés

Dans ce chapitre, nous énonçons les problèmes traités dans cette thèse. Néanmoins avant cela, nous introduisons d'abord la classe de problèmes d'ordonnancement à laquelle appartiennent nos deux problèmes. Il s'agit de problèmes d'ordonnancement avec temps de préparation. Afin de situer ces problèmes par rapport aux travaux qu'on peut trouver dans la littérature, nous décrivons les différentes hypothèses selon lesquelles la préparation des tâches peut être considérée. Ensuite, nous mettons la lumière sur une classe plus spécifique de ce type de problèmes, qui est la classe de problèmes d'ordonnancement avec ressources supplémentaires pour la préparation des tâches. Ceci en fournissant un état de l'art de cette dernière.

2.1 Considération des temps de préparation

Dans la théorie de l'ordonnancement, il est généralement supposé que tout temps de préparation nécessaire pour le traitement d'une tâche est soit inclus dans le temps de traitement soit nul ou négligeable. Ceci peut refléter certains cas réels. Cependant, cette hypothèse peut tout de même altérer la qualité de la solution dans le cas où les temps de préparation ne peuvent pas être négligés. Plusieurs études ont montré l'importance de considérer la préparation comme une partie distincte du traitement dans le processus de construction d'un ordonnancement [5].

L'intérêt des chercheurs pour la considération des temps de préparation ne fait qu'augmenter, et la littérature regorge de travaux qui ont traité les problèmes d'ordonnancement avec temps de préparation. Toutefois, les temps de préparation sont considérés sous plusieurs aspects ; nous présentons, dans ce qui suit, quelques-uns.

Temps de préparation dépendant de la séquence : Lorsqu'un temps de préparation dépend de la tâche précédente et celle à traiter alors il est défini comme étant dépendant de la séquence. Des préparations de ce genre peuvent être rencontrées dans des ateliers utilisant des fours. Le temps nécessaire pour chauffer un four à une température précise dépend de la température précédente et celle requise. Ce type de préparation est indiqué par *SDST* (pour Sequence Dependent Set-up Times) dans le champ β.

Temps de préparation indépendant de la séquence : Un temps de préparation est dit indépendant de la séquence si ce temps dépend uniquement de la tâche à traiter. Par exemple, le transfert de données avant leur traitement, dans une application informatique, est une préparation dont la durée dépend seulement de la taille des données à transférer. Ces temps de préparation sont désignés par s_{ij} ou s_j dans le second champ.

Préparations anticipatoires : Si la préparation se fait sur la machine, alors elle peut commencer sans que la tâche soit disponible. On parle, dans ce cas, de préparations anticipatoires. On retrouve ce type de préparations, par exemple, dans des ateliers de peinture. Le nettoyage des outils et machines qui se fait entre deux tâches est une préparation anticipatoire.

Préparations non-anticipatoires : Les préparations non-anticipatoires sont des préparations effectuées sur les tâches. Par exemple, dans l'industrie métallurgique, le chauffage des métaux est une préparation qui ne peut pas être effectuée tant qu'ils sont sur une machine. De plus, les préparations non-anticipatoires peuvent également nécessiter la disponibilité des machines. De telles préparations se produisent dans une application de trafic aérien. Avant un vol, la compagnie aérienne procède à l'enregistrement et à l'embarquement des passagers. Cette préparation se fait avec la disponibilité de l'avion, la présence des passagers et avec l'intervention des employés de l'entreprise affectés à l'aéroport.

Préparation par lot : On parle de préparation par lot (en anglais *batch*) lorsqu'une préparation se fait une seule fois pour traiter un sous-ensemble de tâches. On rencontre ce genre de situation lorsque des tâches ont les mêmes conditions de traitement. Ainsi, une fois que les machines sont réglées, on peut

traiter ces tâches sans avoir à refaire les préparations. Par exemple, si une matière doit passer par un four à une température précise. Lorsque les matières qui nécessitent la même température sont ordonnancées l'une après l'autre alors la préparation ne se fait qu'avant la première tâche du lot. On peut rencontrer cette situation, aussi, dans le cas où une machine peut traiter plusieurs tâches à la fois et dans ce cas la préparation de ces tâches se fait une seule fois.

Préparation sans ressources supplémentaires : Certaines préparations ne nécessitent pas des ressources supplémentaires. On cite, par exemple, le séchage de peinture dans un atelier. Ces préparations sont dépendantes de la séquence et ne nécessitent pas d'autres ressources. Dans d'autres cas, si les ressources sont disponibles en grande quantité alors ceci permet de lancer plusieurs préparations à la fois. Par conséquent, les ressources supplémentaires ne sont pas considérées.

Préparation avec ressources supplémentaires : Si des ressources de capacité limitée sont nécessaires pour préparer les tâches alors cela engendrera des contraintes de disponibilité qui doivent être prises en compte lors de la construction d'une solution. Par exemple, dans une chaine de production, un travailleur procède au chargement de la tâche sur la machine et au lancement du traitement. Dans ce cas, un traitement ne peut pas démarrer sur une machine si le travailleur est occupé à préparer une autre machine. On retrouve, dans la littérature, deux façons d'indiquer la présence de ressources supplémentaires pour la réalisation d'une préparation.

— La première est dans le cas où on a un seul type de ressource supplémentaire de capacité limitée. Dès lors, cette ressource est indiquée par la notation Sk dans le champ a, où le paramètre k désigne la quantité disponible de cette ressource. Ainsi, un seul serveur est indiqué par $S1$ ou bien S.
— La seconde indication de la présence de ressources est spécifiée par $res^{t}_{\lambda\sigma\delta}$ dans le champ β. Cette dernière est utilisée lorsqu'on a plusieurs types de ressources. Les variables λ, σ et δ désignent, respectivement, le nombre de ressources, la quantité totale de ressources disponibles et la quantité maximale de ressources nécessaires pour une préparation. Si un point «.» est attribué à l'un de ces paramètres alors cela indique que la variable correspondante peut prendre n'importe quelle valeur entière. Tandis qu'un entier positif indique que la variable correspondante est fixe. De plus, le paramètre t indique que ces ressources sont seulement nécessaires pour la préparation.

2.2 Revue de la littérature

On s'intéresse aux problèmes d'ordonnancement avec temps de préparation, lorsque les préparations sont prises en charge par des ressources supplémentaires, autres que les machines. Par ailleurs, ces ressources

sont de capacité limitée. Nous présentons, dans ce qui suit, une revue de littérature des travaux ayant traité ce type de problèmes. On commence par les travaux qui ont traité les machines parallèles, ensuite on passe à ceux qui ont étudié l'open shop. Finalement, on présente les travaux qui ont traité le flow shop. Il est évident qu'il s'agit de temps de préparation indépendants de la séquence. De plus tous ces travaux ont traité des ordonnancements sans lots. Quant à la propriété anticipatoire ou non-anticipatoire, il faut noter que lorsqu'il s'agit de machines parallèles cette propriété n'est pas considérée, vu que ceci n'est pas nécessaire. Tandis que, pour les autres ateliers d'ordonnancement, ceci sera précisé dans la description du problème.

Néanmoins, une revue complémentaire et détaillée sur les problèmes d'ordonnancement avec temps de préparation est fournie par les trois articles [4, 6, 7].

2.2.1 Machines Parallèles

Nous avons réparti les travaux en catégories qui sont selon le nombre de machines et le nombre de ressources. Nous commençons par les papiers ayant traité le problème avec deux machines et un serveur, ensuite, ceux qui ont étudié le problème avec m machines et un serveur, et enfin avec m machines et plusieurs serveurs.

Deux machines parallèles et un serveur

Koulamas (1996) [39] a traité le problème $P2, S|s_j$ avec l'objectif de minimiser le temps mort engendré par l'indisponibilité du serveur. Il a prouvé que le problème est $\mathcal{NP}$-difficile au sens fort et a proposé une heuristique de recherche faisceau efficace pour le résoudre. Hall et al. (2000) [29] ont prouvé que le problème $P2, S|s_j = s|\gamma, \gamma \in \{C_{\max}, \sum C_j\}$ est $\mathcal{NP}$-difficile. Ils ont aussi prouvé que le problème $P2, S|s_j, p_j = 1|\gamma$, avec $\gamma \in \{\sum T_j, \sum U_j w_j\}$ est $\mathcal{NP}$-difficile au sens ordinaire, tandis que pour l'objectif $\sum T_j w_j$, le problème est $\mathcal{NP}$-difficile au sens fort. Ils ont montré aussi que les problèmes $P, S|s_j, p_j = 1|\gamma$ avec $\gamma \in \{C_{\max}, L_{\max}, \sum C_j, \sum C_j w_j, \sum U_j\}$ sont résolubles en temps polynomial.

Abdekhodaee et Wirth (2002) [3] ont étudié le problème $P2, S|s_j|C_{\max}$. Ils ont prouvé que le problème est $\mathcal{NP}$-difficile au sens fort pour certains cas particuliers et ont proposé une formulation en programmation en nombre entier pour le problème général. Ils ont aussi proposé des algorithmes de résolution polynomiaux pour plusieurs cas particuliers. Abdekhodaee et al. (2004) [2] ont traité le même problème, mais pour le cas de temps de traitement identiques et le cas de temps de préparation identiques. Ils ont prouvé qu'ils sont $\mathcal{NP}$-difficile et ont proposé des heuristiques pour les deux cas. Abdekhodaee, Wirth et Gan (2006) [1] ont proposé deux versions d'une heuristique gloutonne, un algorithme génétique et une

version de l'algorithme de Gilmory-Gomory pour résoudre le problème précèdent. Cette étude a été étendue dans [23] où deux formulations en programme en nombres entiers et deux variantes d'un schéma de Branch and Price ont été présentées par Gan, Wirth et Abdekhodaee (2012) [23]. Hasani, Kravchenko et Werner (2013) [32] ont traité le même problème et ont proposé plusieurs formulations en programme mathématique en nombres entiers basées sur des blocs de préparation qui se sont révélées meilleures que les algorithmes présentés dans [1, 23]. Ils ont proposé dans [31] un algorithme génétique et un recuit simulé qui était plus performent que les autres méthodes et pouvaient résoudre des instances de grande taille. Ils ont proposé dans [30] deux heuristiques pour deux types distincts d'instances à très grandes tailles et ont montré leur efficacité par rapport aux méthodes existantes.

Plusieurs machines et un serveur

Kravechenko et Werner (1997) [38] ont traité le problème $P, S|sj = s|C_{max}$ et ils ont montré qu'il est $\mathcal{NP}$-difficile au sens fort. Ils ont aussi proposé quelques heuristiques pour le problème général et ont présenté des algorithmes polynomiaux pour quelques cas particuliers. Kravechenko et Werner (2001) [36] ont traité le même problème, mais avec un autre critère d'optimisation qui est $\sum C_j$, ainsi que des temps de préparation unitaires $(P, S|s_j = 1| \sum C_j)$. Ils ont proposé une heuristique et ont prouvé qu'elle a une erreur absolue délimitée par le produit du nombre de tâches courtes et $m - 2$. Une tâche est dite courte si son temps de traitement est inférieur à $m - 1$.

Glass et al. (2000) [26] ont traité le problème avec des machines dédiées et la minimisation du C_{max} comme critère d'optimisation. Ils ont prouvé que le problème avec deux machines est $\mathcal{NP}$-difficile au sens fort même si les temps de préparation sont identiques ou que les temps de traitement soient identiques. Ils ont montré qu'un simple algorithme glouton donne un ordonnancement au plus deux fois plus long que l'optimal pour le cas de m machines, et ils ont donné une heuristique avec un ratio du pire des cas de $3/2$ pour le cas de deux machines.

Wang et Cheng (2001) [55] ont étudié le problème $P, S|s_j| \sum C_j w_j$ et ont présenté un algorithme, avec une $(5 - 1/m)$-approximation, basé sur la relaxation linéaire. Ils ont aussi montré que la règle SPT (Shortest Processing Time) est d'une approximation de $3/2$ pour $P,S|s_j = s| \sum C_j$.

Brucker et al. (2002) [20] ont prouvé que le problème $P, S|s_j| \sum C_j$ est $\mathcal{NP}$-difficile, ils ont fourni un algorithme en $O(n^7)$ pour résoudre $P3, S|s_j = 1| \sum C_j$ ainsi que d'autres algorithmes polynomiaux pour différents cas particuliers. Kim et Lee (2012) [35] ont étudié le problème $P, S||C_{max}$; ils ont proposé une formulation de programmation linéaire en nombres entiers mixtes et un algorithme hybride pour la résolution de ce dernier.

Plusieurs machines et plusieurs serveurs

Kravechenko et Werner (1998) [37] ont considéré le problème avec m machines parallèles et $m-1$ serveurs qui effectuent la préparation des tâches. Le critère d'optimisation considéré est la minimisation du makespan. Ils ont proposé un algorithme pseudo-polynomial pour résoudre le problème. Werner et Kravchenko (2010) [56] ont traité le même problème, avec k serveurs. Ils ont montré que $Pm, Sk|s_j = 1|C_{\max}$ est $\mathcal{NP}$-difficile et que le problème $Pm, Sk|s_j = 1|L_{\max}$ est $\mathcal{NP}$-difficile au sens fort pour n'importe quel nombre de machines m et pour n'importe quel nombre de serveurs k, avec $k < m$, de même pour $P, Sk|s_j = 1|C_{\max}$. Ils ont aussi montré que le problème $P, Sk|p_j = 1|C_{\max}$ est $\mathcal{NP}$-difficile pour chaque $k > 1$ fixé. Ils ont fourni un algorithme polynomial qui résout le problème $Pm, S(m-1)|s_j = 1|C_{\max}$, ainsi que d'autres algorithmes polynomiaux pour différents cas particuliers avec différents critères d'optimisation.

Laabi, Boudhar et Oulamara (2017) [40] ont traité le problème avec plusieurs ressources. Ils ont montré que le problème $P2|res^t_{k11}, s_i \in \{1,3,5\}, p_i = 1|C_{\max}$ est $\mathcal{NP}$-difficile au sens fort pour $k \geq 7$ et aussi que $P2|res^t_{k11}, s_i \in \{1,3\}, p_i = 1|C_{\max}$ est $\mathcal{NP}$-difficile au sens fort pour $k \geq 5$. Ils ont proposé des algorithmes polynomiaux pour résoudre les problèmes $P2|res^t_{111}, s_i > 1, p_i = 1|C_{\max}$ et $P2|res^t_{.11}, s_i = s, p_i = 1, |C_{\max}$. Ils ont aussi élaboré deux bornes inférieures pour le problème général ainsi que quelques heuristiques pour sa résolution.

2.2.2 Open shop

Dans ce qui suit, nous présentons les travaux ayant traité l'open shop avec plusieurs ressources supplémentaires puis ceux qui ont abordé le problème avec un serveur.

Open shop avec plusieurs ressources

Oulamara, Rebaine et Serairi (2013) [48] ont traité le problème $O2|res^t_{1..}|\ C_{\max}$, où un ensemble de ressources supplémentaires, dont la demande est unitaire, s'occupe de préparations non-anticipatoires sur les tâches. Ils ont montré que, dans le cas préemptif, le problème est $\mathcal{NP}$-difficile au sens ordinaire, tandis que le cas non-préemptif est $\mathcal{NP}$-difficile au sens fort. Ils ont montré que ce dernier reste $\mathcal{NP}$-difficile même lorsque les temps de préparation sont identiques ou que les temps de traitement sont identiques. Ils ont fourni un algorithme polynomial pour le cas avec des temps de préparation identiques et des temps de traitement identiques, ainsi que deux bornes inférieures pour le problème général et plusieurs heuristiques pour le résoudre.

Open shop avec un seul serveur

Glass et al. (2000) [26] ont montré que le problème $O2, S||C_{\max}$ est $\mathcal{NP}$-difficile au sens fort lorsque des préparations anticipatoires sont effectuées par un seul serveur.

Baki et Vickson (2003) [16]ont étudié à la fois les problèmes d'open shop et de flow shop avec deux machines en présence d'un opérateur. Des préparations indépendantes de la séquence ont lieu à chaque fois que l'opérateur change de machine. L'opérateur ne peut traiter aucune tâche pendant la préparation. Cependant, il est également nécessaire pour le traitement. Par conséquent, les préparations sont non-anticipatoires. Ce problème est équivalent au problème à une seule machine avec des temps de préparation indépendants de la séquence et par lots. Ce problème est résoluble en $O(n \log n)$ par l'algorithme de Wagelmaus et Gerodimos [54] lorsque l'objectif est de minimiser le retard maximum. Ils ont montré dans [15] que le problème est $\mathcal{NP}$-difficile au sens ordinaire lorsque l'objectif est de minimiser le nombre pondéré de tâches en retard. Ils ont fourni deux algorithmes pseudo-polynomiaux basés sur l'approche de programmation dynamique pour résoudre ce dernier.

2.2.3 Flow shop

Nous passons, à présent, aux travaux ayant traité le flow shop avec un serveur. Il faut noter que les préparations considérées dans ces travaux sont anticipatoires.

Cheng et al. (1999) [21] ont abordé le problème de flow shop à deux machines et un opérateur, avec des temps d'installation et de retrait. Des préparations dépendantes de la séquence sont nécessaires à chaque fois que l'opérateur change de machine. De plus, des temps de retrait indépendants de la séquence sont effectués après la fin d'un traitement. L'opérateur est nécessaire aussi bien pour les préparations que pour les retraits. Ils ont montré que le problème est $\mathcal{NP}$-difficile au sens fort. Ils ont également proposé certaines heuristiques et testé leurs performances.

Cheng et Kovalyov (2003) [22] ont considéré le même problème sans temps de retrait. Plusieurs critères d'optimisation ont été considérés. Ils ont fourni des algorithmes pour les cas où le problème est polynomial et ils ont prouvé que d'autres cas sont $\mathcal{NP}$-difficiles.

Glass et al. (2000) [26] ont prouvé que le flow shop avec deux machines et un seul serveur est $\mathcal{NP}$-difficile au sens fort. Dans ce problème, le serveur doit effectuer des préparations indépendantes de la séquence et anticipatoires. Ils ont également montré que la version no-wait du problème peut être réduite au problème du voyageur de commerce de Gilmore-Gomory, résoluble en temps polynomial. Su et Lee (2008) [53]

ont également étudié le no-wait flow shop avec deux machines et un serveur, cependant, l'objectif qu'ils ont considéré est la minimisation du temps de fin de traitement total. Ils ont montré que le problème est $\mathcal{NP}$-difficile au sens fort et ont proposé pour sa résolution des heuristiques et un algorithme Branch and Bound.

Lim et al. (2006) [41] ont proposé plusieurs méta-heuristiques pour résoudre le problème du flow shop à deux machines et un serveur avec des temps de préparation anticipatoires et indépendants de la séquence. L'étude expérimentale, que ces auteurs ont menée sur ces méthodes, a montré que les solutions obtenues sont proches de l'optimum. Ling et Xue-guang (2011) [42] ont étudié le même problème avec les temps de traitement unitaires. Ils ont montré que le problème est $\mathcal{NP}$-difficile au sens fort. Ils ont également montré dans [43], que le problème avec des temps de préparation unitaires est $\mathcal{NP}$-difficile et ont proposé pour sa résolution une heuristique gloutonne avec un rapport de pire cas de 3/2. Gharbi et al. (2013) [25] ont proposé des bornes inférieures pour le même problème, cependant le critère est la minimisation du temps de fin de traitement total. Ces bornes inférieures sont basées sur trois schémas de relaxation, à savoir le schéma de relaxation basé sur le temps d'attente, le schéma de relaxation d'une machine et le schéma de relaxation Lagrangienne.

Brucker et al. [19, 20] ont résolu le problème avec m machines et un serveur. Ils ont dérivé des résultats de complexité pour certains cas particuliers et ont montré que d'autres cas peuvent être résolus en temps polynomial. Par exemple, ils ont montré que $F2, S1|p_{ij} = p|C_{\max}$ est $\mathcal{NP}$-difficile au sens ordinaire. Ils ont examiné d'autres critères d'optimisation, notamment $\sum C_j$, $\sum C_j w_j$, $\sum T_j$, $\sum T_j w_j$ et $L_{\max}$, où ils ont identifié quelques cas polynomiaux. Par exemple, ils ont montré que $F, S1|p_{ij} = 1, s_{ij} = s, r_j|\sum C_j$ est résoluble en temps polynomial.

2.3 Open shop à deux machines et un serveur

Dans ce qui suit, nous allons décrire le premier problème que nous avons traité. Il s'agit de problème d'ordonnancement open shop à deux machines et un serveur. L'essentiel des résultats obtenus lors de l'étude de ce problème ont été répartis en deux papiers [12] et [13]. Nous donnons, dans ce qui suit, une description du problème, ensuite, un exemple numérique du problème et des exemples d'application. Enfin, nous présentons les bornes inférieures que nous avons élaborées.

2.3.1 Définition

L'open shop à deux machines et un serveur est défini comme suit. Étant donné un ensemble $T = \{T_1, \ldots, T_n\}$ de n tâches à traiter sur deux machines M_1 et M_2 en présence d'un serveur $S1$. La tâche T_j comprend deux opérations O_{1j} et O_{2j}, $j = 1, \ldots, n$, qui seront traitées sur M_1 et M_2 respectivement. L'ordre de traitement des tâches sur les machines est arbitraire et sera défini lors de la construction d'une solution. L'opération O_{ij} doit être préparée par le serveur, ensuite sera traitée sans préemption sur la machine M_i, $i = 1, 2$. Cependant, si une tâche a un temps de préparation nul alors elle sera directement traitée sur la machine appropriée. En outre, le serveur est seulement nécessaire pour la préparation et sera instantanément libre pour préparer une autre tâche à la fin de cette phase. Pour une opération O_{ij}, $i = 1, 2$, $j = 1, \ldots, n$, nous définissions :

— Le temps de préparation $s_{ij} \geq 0$.
— Le temps de traitement $p_{ij} \geq 0$.
— La durée d'exécution de l'opération O_{ij} : $P_{ij} = s_{ij} + p_{ij}$, et qui est la durée totale de l'opération.

De plus, les préparations sont non-anticipatoires et leurs durées indépendantes de la séquence. L'objectif est de trouver un ordonnancement réalisable qui minimise le temps de fin de traitement final, *c.-à-d.* le makespan. Le problème est, dans ce cas, noté $O2, S1|s_{ij}|C_{\max}$.

Exemple 2.1. Nous présentons, maintenant, un exemple numérique du problème $O2, S1|s_{ij}|C_{\max}$. D,abord, nous considérons l'instance du $O2||C_{\max}$ résumée dans le Tableau 2.3.1.

T_j	T_1	T_2	T_3
p_{1j}	3	4	3
p_{2j}	5	1	3

Tableau 2.3.1 – Une instance du $O2||C_{\max}$

Une solution optimale de cette instance, obtenue par la règle GARD (voir l'Algoritme 3), est représentée par la Figure 2.3.1.

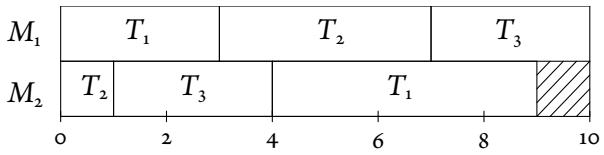


Figure 2.3.1 – L'ordonnancement obtenu par la règle GARD pour le $O2||C_{\max}$

Si des préparations sont requises et sont prises en charge par un serveur, dit $S1$, alors on doit réordonnancer

les tâches pour obtenir une solution réalisable. Pour illustrer ce propos, considérons les temps de préparation résumés dans le Tableau 2.3.2.

T_j	T_1	T_2	T_3
s_{1j}	2	4	1
s_{2j}	3	1	2

Tableau 2.3.2 – Les temps de préparation

Si on garde la même séquence des tâches sur les deux machines telles qu'obtenues par la règle GARD, alors on obtient l'ordonnancement S, dans lequel le traitement commence sur M_1. Si on change la machine sur laquelle commence le traitement, en gardant la même séquence, alors on obtient l'ordonnancement S'. Les deux ordonnancements sont représentés dans Figure 2.3.2a et 2.3.2b avec $C_{\max}(S) = C_{\max}(S') = 18$, dans les deux cas.

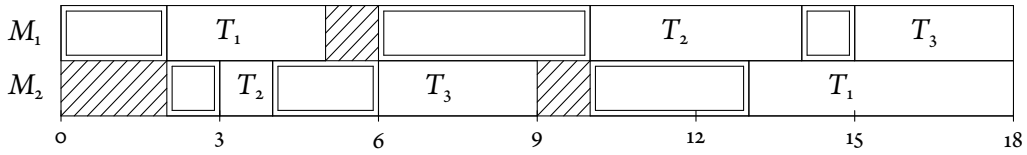


Figure 2.3.2a – L'ordonnancement obtenu par la règle GARD en commençant sur M_1

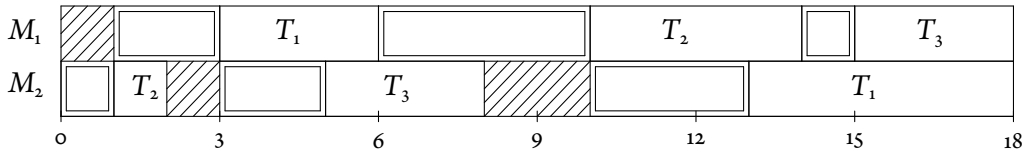


Figure 2.3.2b – L'ordonnancement obtenu par la règle GARD en commençant sur M_2

Une meilleure solution est obtenue en changeant la séquence comme l'illustre la Figure 2.3.3, avec un makespan $C_{\max}(S^*) = \sum_{j=1}^{n} s_{1j} + p_{1j}$, donc optimal.

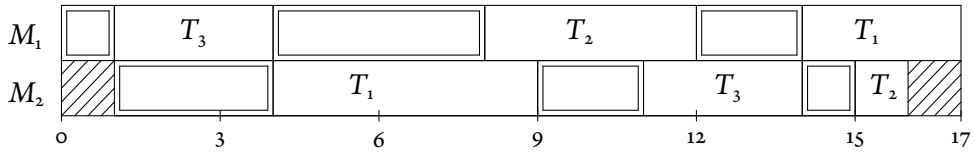


Figure 2.3.3 – L'ordonnancement optimal de l'Exemple 2.1

2.3.2 Motivation

On retrouve des problèmes avec les caractéristiques que nous avons présentées ci-dessus, par exemple, dans un centre de recyclage qui s'occupe du tri sélectif des déchets. Ceci se fait par deux machines qui filtrent les objets par rapport à leurs caractéristiques : poids et solidité. Ainsi, la première machine récupère le verre et le métal, et la seconde récupère le papier et le plastique, tandis que le reste est considéré comme des déchets. Un ouvrier procède au chargement d'un lot d'objets à trier sur une machine avant de lancer la machine. De plus, les lots peuvent passer sur les machines dans n'importe quel ordre. Nous avons, dans ce cas, un problème d'open shop à deux machines et un serveur. Les machines sont les machines de tri, le serveur est l'ouvrier qui fait le chargement, et les tâches sont les lots d'objets.

Un autre problème qui pourrait se modéliser de la sorte se rencontre dans une compagnie de location de voitures. Lorsqu'une voiture est réservée, elle doit passer par une vérification dans le garage de maintenance et elle doit aussi être nettoyée. Ces deux opérations peuvent se faire dans n'importe quel ordre. Ainsi, une voiture peut passer par une vérification ensuite sera nettoyée, ou bien l'inverse. Cependant, un employé procède à l'installation de la voiture sur la machine de nettoyage et aussi à l'installation de la voiture sur l'élévateur ou dans la zone de vérification. Après la fin d'un passage d'une voiture dans une zone (nettoyage ou vérification), c'est l'équipe qui travaille dans cette zone qui s'occupe de faire sortir la voiture à la fin de la tâche. Nous avons dans cette situation un open shop à deux machines et un serveur. La première machine est la zone de nettoyage. Tandis que la seconde machine est l'équipe de maintenance qui procède à la vérification de l'état de la voiture. Le serveur est employé qui s'occupe d'installer la voiture dans la zone associée. Les tâches sont les voitures réservées pour les prochains jours.

2.3.3 Bornes inférieures

Dans cette section, il est question de bornes inférieures avec considération des temps de préparation, relativement au critère du makespan.

La première est basée sur la relaxation des contraintes d'indisponibilité du serveur. Si on ignore ces contraintes alors le problème sera réduit au problème open shop classique. Puisqu'on a des temps de préparation et des temps de traitement pour chaque tâche alors LB_1 est clairement une borne inférieure du makespan de n'importe quel ordonnancement S [12].

$$C_{\max}(S) \geq LB_1 = \max\left\{\sum_{j=1}^{n}(s_{1j}+p_{1j}), \sum_{j=1}^{n}(s_{2j}+p_{2j}), \max_{1\leq j\leq n}(s_{1j}+p_{1j}+s_{2j}+p_{2j})\right\}. \qquad (2.1)$$

La deuxième borne inférieure est basée sur l'occupation continue du serveur. Si on a des temps de préparation plus grands que les temps de traitement alors le serveur sera continuellement occupé durant l'intervalle $[0, \sum_{1\leq j\leq n}(s_{1j} + s_{2j})]$. Il passera d'une préparation à une autre jusqu'à la fin de toutes les préparations suivies par le traitement de la dernière tâche. Ainsi donc, LB_2 est une borne inférieure du makespan de tout ordonnancement S [12].

$$C_{\max}(S) \geq LB_2 = \sum_{1\leq j\leq n}(s_{1j} + s_{2j}) + \min\{\min_{1\leq j\leq n}\{p_{1j}\}, \min_{1\leq j\leq n}\{p_{2j}\}\}. \tag{2.2}$$

La troisième borne est une modification de la première et qui plus est, prend en compte la préparation de la première tâche ordonnancée. Supposant que le traitement commence sur M_1, alors la première préparation engendrera un temps mort sur M_2. Le temps mort est d'au moins $\min_j\{s_{1j}\}$ unités de temps. Le temps de fin de traitement sur M_2 sera égal à :

$$LB_3^1 = \min_{1\leq j\leq n}\{s_{1j}\} + \sum_{j=1}^{n}(s_{2j} + p_{2j});$$

Si on suppose qu'on commence le traitement sur M_2 alors le temps de fin de traitement sur M_1 sera d'au moins :

$$LB_3^2 = \min_{1\leq j\leq n}\{s_{2j}\} + \sum_{j=1}^{n}(s_{1j} + p_{1j}).$$

La borne inférieure sera égale au minimum des deux temps de fin de traitement précédents, ce qui nous donne la borne inférieure suivante :

$$C_{\max}(S) \geq LB_3 = \min\left\{\max\{LB_3^1, \sum_{j=1}^{n}(s_{1j} + p_{1j})\}, \max\{LB_3^2, \sum_{j=1}^{n}(s_{2j} + p_{2j})\}\right\}. \tag{2.3}$$

2.4 Job shop à deux machines et un serveur

Le second problème que nous avons considéré est l'atelier à cheminement multiple (job shop) à deux machines et un serveur. Les résultats obtenus de son étude sont résumés dans le papier [14]. Nous donnons, dans ce qui suit, la description du problème, un exemple numérique et un exemple d'application. Enfin, nous présenterons les bornes inférieures du problème.

2.4.1 Définition

Ce second problème a les mêmes hypothèses que $O2, S1|s_{ij}|C_{\max}$, présenté dans la sous-section précédente. Par conséquent, le problème est décrit comme suit. Étant donné un ensemble $T = \{T_1, \ldots, T_n\}$ de n tâches à traiter sur deux machines M_1 et M_2 en présence d'un serveur $S1$. L'opération O_{ij} de la tâche T_j, $j = 1, \ldots, n$, doit être préparée par le serveur, ensuite sera traitée sans préemption sur la machine M_i, $i = 1, 2$. Les préparations sont non-anticipatoires et leurs durées indépendantes de la séquence.

En outre, la tâche T_j a son propre ordre de passage sur les machines, défini à l'avance et appelé cheminement. Le cheminement de T_j est noté $R_j \in \{M_1 \to M_2, M_2 \to M_1, M_1, M_2\}, j = 1, \ldots, n$. Ainsi, l'ensemble des tâches est partitionné en quatre sous-ensembles disjoints qui sont définis selon le cheminement des tâches. On définit donc les sous-ensembles N_1, N_2, N_3, N_4 et leur cardinal n_1, n_2, n_3, n_4 comme suit.

$$N_1 = \{T_j, M_1 \to M_2\} \text{ et } n_1 = |N_1|;$$
$$N_2 = \{T_j, M_2 \to M_1\} \text{ et } n_2 = |N_2|;$$
$$N_3 = \{T_j, M_1\} \text{ et } n_3 = |N_3|;$$
$$N_4 = \{T_j, M_2\} \text{ et } n_4 = |N_4|.$$

Il faut noter que, si $T_j \in N_3$ (resp. N_4) alors son temps de préparation et de traitement seront nuls sur la machine M_2 (resp. M_1). L'objectif est de trouver un ordonnancement réalisable qui minimise le makespan. Le problème est, ainsi, noté $J2, S1|s_{ij}|C_{\max}$.

Considérons l'instance suivante du $J2, S1|s_{ij}|C_{\max}$. Les temps de préparation et de traitement ainsi que le cheminement des tâches sont résumés dans le Tableau 2.4.1.

	T_1	T_2	T_3	T_4
s_{1j}	3	4	4	2
p_{1j}	1	5	3	3
s_{2j}	0	2	3	0
p_{2j}	5	4	3	0
R_j	$M_1 \to M_2$	$M_2 \to M_1$	$M_1 \to M_2$	M_1

Tableau 2.4.1 – Instance du $J2, S1|s_{ij}|C_{\max}$

Puisque le temps de traitement total sur M_1 est égal au makespan alors la solution suivante est optimale. D'abord, nous présentons la solution par un digramme de Grantt des tâches dans la Figure 2.4.1.

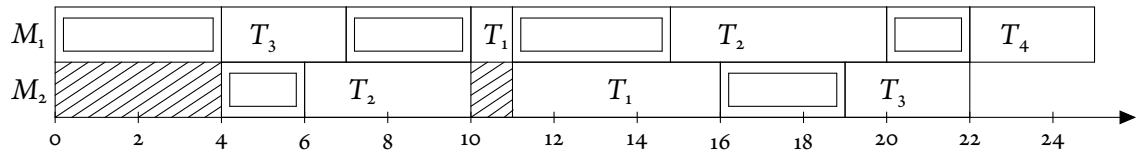


Figure 2.4.1 – Diagramme de Grantt des tâches

Ensuite, dans la Figure 2.4.2 nous présentons le diagramme de Grantt des machines, dans lequel, on peut clairement voir le cheminement des tâches.

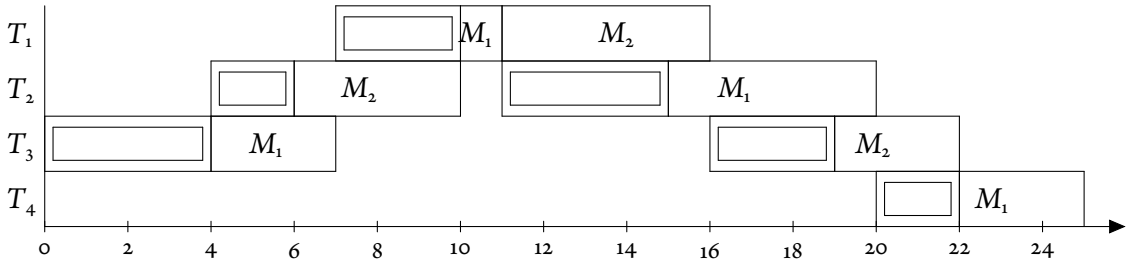


Figure 2.4.2 – Diagramme de Grantt des machines

2.4.2 Motivation

Un exemple d'application de ce modèle est décrit dans ce qui suit. Une petite clinique de chirurgie générale dispose d'une salle d'opération et d'une salle de soins intensifs, qui ne peuvent accueillir qu'un seul patient à la fois. Une salle commune, de plus grande taille, peut accueillir des patients dans un état stable. Pour des raisons d'efficacité, un patient qui sera opéré peut être placé en salle de réanimation pour une période d'observation, soit avant soit après la chirurgie. Un patient peut être admis pour observation sans avoir à subir une intervention chirurgicale. Dans le cas d'une intervention chirurgicale simple, le patient ne sera placé en salle de soins intensifs ni avant ni après l'intervention. Une préparation est faite avant une opération, qui consiste à anesthésier le patient et à préparer la salle d'opération. De plus, l'installation du patient dans la salle de soins est également une préparation. Ces préparations sont effectuées par une équipe d'infirmiers qui peuvent s'occuper de l'installation d'un patient à la fois. Les machines sont, dans ce cas, la salle d'opération et la salle de soins intensifs, tandis que les tâches sont les patients. Ainsi, nous avons un job shop à deux machines, avec des préparations indépendants de la séquence et non-anticipatoires car elles nécessitent la disponibilité de la salle et la présence du patient. De plus, ces préparations sont gérées par une ressource à capacité limitée, qui est l'équipe d'infirmières.

2.4.3 Bornes inférieures

Avant de clore ce chapitre, nous présentons les bornes inférieures du job shop à deux machines et un serveur avec la considération des temps de préparation, toujours par rapport au critère du makespan. Nous en avons présenté quatre bornes. Il va sans dire que le maximum de ces bornes inférieures est aussi une borne inférieure.

La première borne inférieure prend en considération la première préparation. Si toutes les tâches ont un temps de préparation supérieur à zéro, alors la première préparation effectuée par le serveur engendrera un temps mort sur l'autre machine. Ce temps mort dépend de la machine sur laquelle débute le traitement. La borne inférieure L_1 [14] est donnée par le cas qui produit le plus petit temps de fin de traitement global, par conséquent

$$L_1 = \min \left\{ \max\{\sum_{j=1}^{n}(s_{1j}+p_{1j}) + \min_{T_j \notin N_3 \cup N_1}\{s_{2j}\}, \sum_{j=1}^{n}(s_{2j}+p_{2j})\}; \max\{\sum_{j=1}^{n}(s_{1j}+p_{1j}), \sum_{j=1}^{n}(s_{2j}+p_{2j}) + \min_{T_j \notin N_4 \cup N_2}\{s_{1j}\}\} \right\}. \tag{2.4}$$

Les deux parties de la formule représentent le temps de fin de traitement au cas où la première préparation se produirait sur M_1 ou M_2. Notez que la première préparation sur M_1 (resp M_2) ne peut pas être une tâche de N_2 (resp N_1) car elle doit d'abord être traitée sur M_2 (resp M_1). Il ne peut pas non plus s'agir d'une tâche de N_4 (resp N_3). L_1^1 et L_1^2 sont les temps de fin de traitement au cas où la première préparation se produirait sur M_1 ou M_2, respectivement.

La deuxième borne inférieure, L_2 [14], est basée sur une occupation continue du serveur. Si les temps de préparation sont plus longs que les temps de traitement, le serveur sera continuellement occupé pendant $[0, \sum_{j=1}^{n} s_{1j} + s_{2j}]$. Il effectuera une préparation directement après l'autre, jusqu'à l'achèvement de toutes les tâches suivies du traitement de la dernière tâche. Par conséquent, ce qui suit est une borne inférieure du makespan :

$$C_{\max}(S) \geq L_2 = \sum_{j=1}^{n}(s_{1j}+s_{2j}) + \min \left\{ \min_{T_j \notin N_4 \cup N_1}\{p_{1j}\}, \min_{T_j \notin N_3 \cup N_2}\{p_{2j}\} \right\}. \tag{2.5}$$

Notez que dans L_2, la dernière tâche traitée sur M_1 (resp M_2) ne peut pas être une tâche de N_1 (resp N_2) car elle doit être traitée sur M_2 (resp M_1) par la suite. De plus, il ne peut pas s'agir d'une tâche de N_4 (resp N_3).

La troisième borne inférieure, L_3 [14], est une version modifiée de la seconde. Dans L_2, on suppose clairement que le nombre d'opérations est le même sur les deux machines, ce qui pourrait être le cas. Cela permet au serveur d'être continuellement occupé, car il peut démarrer instantanément une autre tâche sur l'autre machine à la fin d'une préparation. Cependant, si le nombre d'opérations sur les deux machines diffère alors nous obtenons la borne inférieure suivante.

On suppose que $n_3 > n_4$ (l'autre cas est symétrique). La troisième borne inférieure est

$$C_{\max}(S) \geq L_3 = \sum_{j=1}^{n}(s_{1j} + s_{2j}) + p_{1k_1} + p_{1k_2} + \ldots + p_{1k_{n_3-n_4}}, \tag{2.6}$$

où,
$p_{1k_1} = \min_{T_j \notin N_4}\{p_{1j}\}$;
$p_{1k_\ell} = \min_{T_j \notin N_4}\{p_{1j}/j \neq k_1, \ldots, k_{\ell-1}\}, \ell = 2, \ldots, n_3 - n_4$;
Cette borne inférieure est basée sur des périodes d'arrêt minimales du serveur. Le serveur est censé être occupé lors du traitement des premières opérations n_4 sur chaque machine. Ensuite, il sera inactif pendant les phases de traitement des $(n_3 - n_4)$ opérations restantes. Il est à noter que les $(n_3 - n_4)$ opérations peuvent être au début ou à la fin de l'ordonnancement ou dans les deux parties.

La dernière borne inférieure, L_4 [14], est obtenue en appliquant l'algorithme de Jackson [27] sans tenir compte des contraintes de disponibilité du serveur. Les valeurs C_1^{Jck} et C_2^{Jck} correspondent au temps de fin de traitement sur M_1 et M_2, respectivement, obtenu en appliquant la règle de Jackson qui est résumée dans l'Algorithme 2. Cependant, on considère pour chaque opération la durée d'exécution au lieu du temps de traitement. La borne inférieure est donc,

$$C_{\max}(S) \geq L_4 = \max\{C_1^{Jck}, C_2^{Jck}\}. \tag{2.7}$$

Conclusion

Nous avons présenté, dans ce chapitre, les deux problèmes traités dans cette thèse. D'abord, nous avons énoncé les différentes hypothèses selon lesquelles les temps de préparation sont pris en compte dans les problèmes d'ordonnancement. Par la suite, nous avons fourni un état de l'art des problèmes d'ordonnancement avec temps de préparation en présence de ressources supplémentaires. Pour terminer, nous avons

présenté les deux problèmes traités, à savoir l'open shop et le job shop à deux machines et un serveur. Pour chacun des deux problèmes, nous avons donné une description détaillée, un exemple numérique et un exemple d'application. De plus nous avons présenté les bornes inférieures respectives de chaque problème. Nous résumons, dans ce qui suit, les notations à retenir de ce chapitre.

S_1 :	le seveur.
s_{ij} :	temps de préparation de la tâche T_j pour son traitement sur $M_i, j = 1, \ldots, n, i = 1, \ldots, m$.
p_{ij} :	temps de traitement de la tâche T_j sur $M_i, j = 1, \ldots, n, i = 1, \ldots, m$.
P_{ij} :	durée d'exécution de la tâche T_j sur M_i qui égale à $s_{ij} + p_{ij}, j = 1, \ldots, n, i = 1, \ldots, m$.
R_j :	cheminement de la tâche $T_j, j = 1, \ldots, n$, avec $R_j \in \{M_1 \to M_2, M_2 \to M_1, M_1, M_2\}$.
N_1 :	le sous-ensemble de tâches ayant le cheminement $R_j = M_1 \to M_2$; $n_1 = \lvert N_1 \rvert$.
N_2 :	le sous-ensemble de tâches ayant le cheminement $R_j = M_2 \to M_1$; $n_2 = \lvert N_2 \rvert$.
N_3 :	le sous-ensemble de tâches qui seront traitées seulement sur M_1 $(R_j = M_1)$; $n_3 = \lvert N_3 \rvert$.
N_4 :	le sous-ensemble de tâches qui seront traitées seulement sur M_2 $(R_j = M_2)$; $n_4 = \lvert N_4 \rvert$.

3

Résultats de complexité

Comme on l'a vu dans le Chapitre 1, la première démarche à suivre dans l'étude d'un problème est de déterminer sa complexité. Ainsi, dans ce chapitre nous allons présenté l'ensemble des résultats de complexité issus de l'étude des deux problèmes, à savoir, l'open shop et le job shop à deux machines et un serveur. Nous avons utilisé, dans le processus de réduction, le problème de partition (Exemple 1.1) et le problème de 3-partition (Exemple 1.2), connus $\mathcal{NP}$-difficile au sens ordinaire et fort, respectivement[24].

3.1 Open shop à deux machines et un serveur

Dans cette section, nous présentons les résultats de complexité obtenus à partir de l'étude du problème $O2, S1|s_{ij}|C_{\max}$. Nous avons étudié la complexité de cinq cas particuliers. Le premier est le cas avec des temps de traitement identiques, qui est noté $O2, S1|s_{ij}, p_{ij} = p|C_{\max}$. Le second est le cas avec des temps de préparation identiques, noté $O2, S1|s_{ij} = s|C_{\max}$. Le troisième est le cas avec des temps de préparation et de traitement proportionnels. Ceci est le sous-problème dans lequel les temps de préparation et de traitement des deux opérations de chaque tâche sont identiques. Ce dernier est noté $O2, S1|s_{1j} = s_{2j} = s_j, p_{1j} = p_{2j} = p_j|C_{\max}$. Le quatrième sous-problème que nous avons étudié est celui où les temps de prépa-

ration et de traitement d'une tâche sont identiques sur chaque machine. Ainsi, ce sous-problème est noté $O2, S1|s_{1j} = p_{1j}, s_{2j} = p_{2j}|C_{\max}$. On termine cette section par un cas particulier du précédent sous-problème. Dans ce cas, les temps de traitement et les temps de préparation, sur l'une des deux machines sont identiques. Ce dernier sous-problème, noté $O2, S1|s_{1j} = p_{1j} = a, s_{2j} = p_{2j}|C_{\max}$, est $\mathcal{NP}$-difficile au sens ordinaire, tandis que les précédents sont $\mathcal{NP}$-difficiles au sens fort.

3.1.1 Temps de traitement identiques

Nous commençons par la complexité du cas particulier du problème avec des temps de traitement identiques, *c. à d.* $O2, S1|s_{ij}, p_{ij} = p|C_{\max}$.

Théorème 3.1. *$O2, S1|s_{ij}, p_{ij} = p|C_{\max}$ est $\mathcal{NP}$-difficile au sens fort [9, 12].*

Démonstration. Étant donnée une instance arbitraire de 3-partition, on définit l'instance suivante du $O2, S1|s_{ij}, p_{ij} = p|C_{\max}$ avec $3n + 1$ tâches, $T_1, \ldots, T_{3n+1}$. Les temps de traitement sont tous égaux à B, tandis que les temps de préparation sont donnés dans ce qui suit.

Sur la machine M_1 :

- $s_{1j} = a_j, j = 1, \ldots, 3n$ (notées tâches de P).
- $s_{1,3n+1} = 0$.

Sur la machine M_2 :

- $s_{2j} = B, j = 1, \ldots, n-2$ (notées opérations de V).
- $s_{2,n-1} = 2B$.
- $s_{2j} = 0, j = n, n+1, \ldots, 3n+1$ (notées opérations de U).

Pour montrer que le problème $O2, S1|s_{ij}, p_{ij} = p|C_{\max}$ est $\mathcal{NP}$-complet au sens fort, on montre que le problème de décision associé a une solution S avec $C_{\max}(S) \leq y = (4n+1)B$ si, et seulement si, 3-partition a une solution.

La preuve est en deux parties ; dans la première, on montre que si le problème de 3-partition admet une solution alors le problème d'ordonnancement, décrit en haut, a une solution S avec $C_{\max}(S) = (4n+1)B$. Dans la seconde partie, on montre que si le problème d'ordonnancement a une solution S avec $C_{\max}(S) = (4n+1)B$ alors ceci implique que le problème 3-partition a une solution.

Supposons, d'abord, que le problème 3-partition a une solution, et $A_\ell, \ell = 1, \ldots, n$, sont les sous-ensembles de A. Soient A'_ℓ les sous-ensembles des tâches définis comme, $A'_\ell = \{T_j, j \in A_\ell\}, \ell = 1, \ldots, n$. L'ordonnancement S ayant un makespan égal à $y = (4n+1)B$ peut être décrit comme suit.

— Il n'y a aucun temps mort sur aucune des deux machines. De plus, la machine M_1 traite les tâches selon la séquence suivante : $(A'_1, T_{3n+1}, A'_2, A'_3, \ldots, A'_n)$.
— Tandis que, la machine M_2 traite les tâches dans l'ordre suivant. Elle commence par trois opérations de U, ensuite T_{n-1}. La séquence ayant l'ordre suivant : deux opérations de U suivies d'une opération V, est répétée $(n-2)$ fois, jusqu'à l'ordonnancement de toutes les opérations de V. Finalement, on ordonnance les trois dernières opérations de U.
— Il est à noter qu'une tâche ne doit pas être traitée en même temps sur les deux machines. Dès lors, on traite les tâches sur M_2 dans l'ordre inverse de la séquence $A'_1, A'_2, A'_3, \ldots, A'_n$, tout en vérifiant l'ordre décrit précédemment.

L'ordonnancement réalisable S, avec $C_{\max}(S) = (4n+1)B$, est représenté dans la Figure 3.1.1.

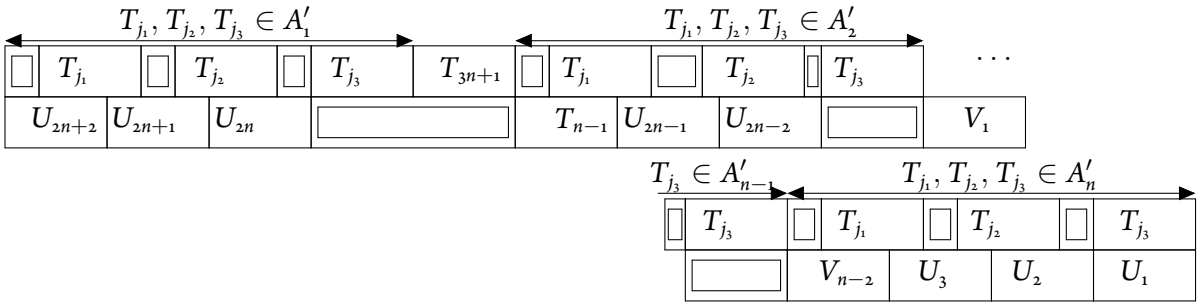


Figure 3.1.1 – Structure de l'ordonnancement

On suppose, désormais, que le problème d'ordonnancement a une solution S avec $C_{\max}(S) = (4n+1)B$, ensuite on montre que 3-partition admet une solution. Dans ce qui suit, on décrit l'ordonnancement S et on montre qu'un tel ordonnancement ne peut exister que si 3-partition a une solution.

On précise qu'ayant un temps de fin de traitement final égal au temps de traitement total sur les deux machines, fait qu'il n'y a aucun temps mort sur aucune des deux machines. Par conséquent, tout temps mort augmentera la valeur du makespan à plus de $(4n+1)B$ unités de temps.

Sur la machine M_1, la tâche T_{3n+1} doit être traitée en parallèle de la préparation de la tâche T_{n-1} sur la machine M_2. De plus, elle doit être précédée par le traitement d'une tâche de P, comme le montre la Figure 3.1.2, à défaut de quoi, la préparation de T_{n-1} sur M_2 causera un temps mort sur M_1.

La préparation de chaque opération de V sur M_2 doit être ordonnancée en parallèle du traitement d'une tâche de P sur M_1, comme l'illustre la Figure 3.1.2. Autrement, M_1 sera en arrêt durant les phases de préparation.

On définit les $(n-2)$ paires d'opérations et les trois opérations $O_{2,n-1}$, $O_{1,3n+1}$ et la tâche de P, représentées dans la Figure 3.1.2, comme étant les blocs, BL_k, $k = 1, \ldots, n-1$. On suppose, sans perte de généralité, qu'ils sont ordonnancés selon l'ordre de leur indice.

Les blocs BL_k, $k = 1, \ldots, n-1$ ne sont pas traités directement l'un après l'autre. Cela afin d'éviter que M_1 soit en arrêt durant l'intervalle de temps entre la fin de la tâche de P d'un bloc et le début de la tâche de P du bloc qui le succède. Pour s'assurer que les machines soient occupées, durant les intervalles de temps séparant le traitement de deux blocs, il faut qu'au moins une tâche de P et une opération de U soient en traitement sur M_1 et M_2, respectivement, durant ces intervalles.

Avant le traitement du bloc BL_1, au moins une tâche est traitée sur M_2. Si ce n'est pas le cas alors la préparation de la tâche de P causera un temps mort sur M_2. De plus, au moins une opération de U est traitée sur M_1, sans quoi, on aura un temps mort en parallèle du traitement de la tâche qui précède bloc BL_1 sur M_2.

Après BL_{n-1}, au moins une tâche est traitée sur M_1, pour éviter qu'elle soit en arrêt lorsque T_{n-1} sera traitée sur M_2. En outre, au moins une tâche est traitée sur M_2 après BL_{n-1}. Dans le cas contraire, et vu que la préparation et le traitement de la tâche ordonnancée sur M_1 prennent plus que B unités de temps, alors cette tâche sur M_1, qui est après BL_{n-1}, causera un temps mort sur M_2.

Soient P_k les sous-ensembles des tâches de P traitées sur M_1 avant BL_k, $k = 1, \ldots, n-1$, y compris la tâche de P dans BL_k. P_n représente les tâches de P traitées sur M_1 après BL_{n-1}. Le traitement et la préparation des tâches qui précèdent la préparation des V_k sur M_2 et celles traitées après BL_{n-1} sont égaux à

$$\sum_{j \in P_k} (s_{1j} + p_{1j}) = B.|P_k| + \sum_{j \in P_k} a_j = I_k, k = 1, \ldots, n.$$

Les intervalles de temps qui sont avant, entre et après les blocs BL_k, $k = 1, \ldots, n-1$, sur M_2 sont tous de longueur multiple de B. En effet, toutes les tâches traitées sur M_1, durant ces intervalles, ont un temps de préparation nul et un temps de traitement égal à B, comme on peut le voir dans la Figure 3.1.2.

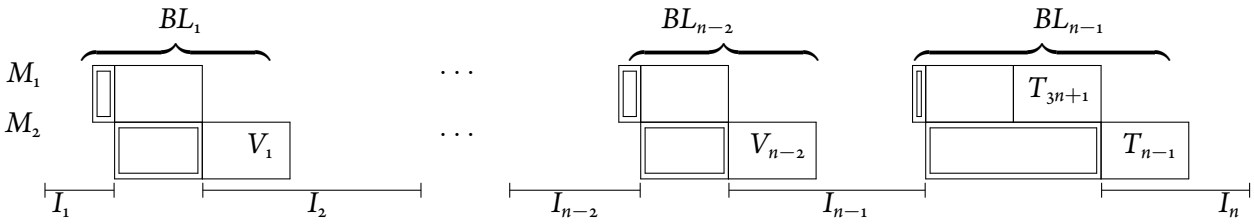


Figure 3.1.2 – Ordonnancement par blocs

Dans ce cas, $\sum_{j \in P_\ell} a_j$, $\ell = 1, \ldots, n$ sont des multiples de B. Puisque $\frac{B}{4} < a_j < \frac{B}{2}$, alors il y a exactement n intervalles de temps durant lesquels $3n$ tâches sont préparées. On doit avoir $|P_\ell| = 3$ et $\sum_{j \in P_\ell} a_j = B$,

$\ell = 1, \ldots, n$. Si on définit les sous-ensembles $A_\ell = P_\ell$ alors A_ℓ, $\ell = 1, \ldots, n$ représente une solution du 3-partition.

On a montré que si le problème d'ordonnancement a une solution S avec $C_{\max}(S) = (4n+1)B$ alors 3-partition a une solution. De ce fait, on termine la preuve par laquelle on a montré que le problème d'ordonnancement a une solution S avec $C_{\max}(S) = (4n+1)B$ si, et seulement si le problème de 3-partition a une solution. □

3.1.2 Temps de préparation identiques

Passons, maintenant, à la complexité du cas où on a des temps de préparation identiques, *i.e.* le problème $O2, S1|s_{ij} = s|C_{\max}$.

Théorème 3.2. *$O2, S1|s_{ij} = s|C_{\max}$ est $\mathcal{NP}$-difficile au sens fort [9, 12].*

Démonstration. Étant donnée une instance arbitraire de 3-partition, on construit l'instance suivante du $O2, S1|s_{ij} = s|C_{\max}$, avec $4n$ tâches. Les temps de préparation sont identiques, ainsi $s_{ij} = B$, $i = 1, 2$, $j = 1, \ldots, 4n$. Les temps de traitement sont résumés dans le Tableau 3.1.1.

	$P : T_j, j = 1, \ldots, 3n$	$P' : T_j, j = 3n+1, \ldots, 4n$
p_{1j}	a_j	$3B$
p_{2j}	0	$5B$

Tableau 3.1.1 – Les temps de traitement de l'instance du $O2, S1|s_{ij} = s|C_{\max}$

Afin de prouver que le problème $O2, S1|s_{ij} = s|C_{\max}$ est $\mathcal{NP}$-difficile au sens fort, on montre qu'un ordonnancement réalisable S avec $C_{\max}(S) \leq y = 9nB$ existe si, et seulement si, le problème 3-partition a une solution. Ceci se fait en deux parties. Dans un premier temps, on montre que si le problème 3-partition a une solution alors le problème d'ordonnancement donné en haut a une solution S avec $C_{\max}(S) = 9nB$. Ensuite, on montre que si le problème d'ordonnancement admet une solution S avec $C_{\max}(S) = 9nB$ alors ceci implique que 3-partition a une solution.

On suppose, d'abord, que 3-partition admet une solution. Soient A_ℓ, $\ell = 1, \ldots, n$, les sous-ensembles de A. A'_ℓ représentent les sous-ensembles de tâches définis comme suit, $A'_\ell = \{T_j, j \in A_\ell\}, \ell = 1, \ldots, n$. L'ordonnancement S peut être décrit comme suit :

— Il n'y a aucun temps mort sur M_2 puisque le temps de fin de traitement et le temps de traitement total sont égaux sur cette machine.
— Sur M_1, on a exactement nB unités de temps mort qui correspondent à la différence entre le temps de fin de traitement final et le temps de traitement total sur M_1.
— On commence le traitement avec une tâche de P' sur M_2 suivie de trois tâches de P, ensuite une tâche P' suivie de trois tâches de P, ainsi de suite.
— Sur M_1, on traite les tâches de A'_1 suivies d'une tâche de P', ensuite les tâches de A'_2 suivies d'une tâche de P', ainsi de suite.

L'ordonnancement obtenu est tel que illustré par la Figure 3.1.3, avec un makespan $y = 9nB$.

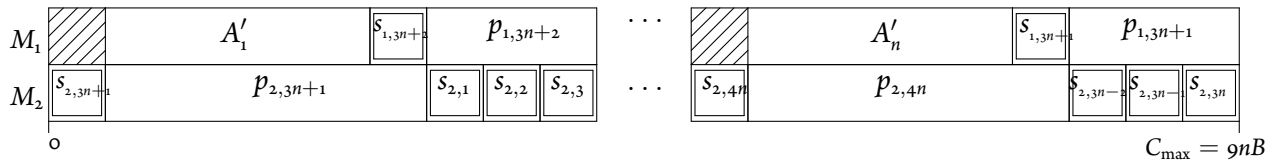


Figure 3.1.3 – La structure de l'ordonnancement

À présent, on suppose qu'un ordonnancement S, avec $C_{\max}(S) = 9nB$, existe et on montre qu'il existe une solution du problème 3-partition.

Pour construire la preuve on commence par la description de l'ordonnancement S et on montre qu'un tel ordonnancement ne peut exister que si 3-partition a une solution. Il est à noter que puisque le temps de traitement total sur M_2 est égal au makespan alors, il n'y a aucun temps mort sur M_2. Tandis que, M_1 est en arrêt durant $C_{\max}(S) - \sum_{j=1}^{4n}(s_{1j} + p_{1j}) = nB$ unités de temps. On précise aussi que tout temps mort sur M_2 et tout temps mort supplémentaire sur M_1 augmenterait la valeur du makespan.

Toutes les préparations sur M_1 doivent être ordonnancées en parallèle du traitement des tâches de P' sur M_2. Autrement, on aura un temps mort sur M_2. De plus, il n'y a aucune intersection entre l'intervalle de temps où une tâche de P' est traitée sur M_1 et l'intervalle de temps où une autre tâche de P' est traitée sur M_2. Supposons le cas contraire, on définit $I_k \geq 0$ comme étant la longueur de l'intersection des deux intervalles de temps. Nous rappelons qu'un temps mort est dû à deux cas. Le premier est le cas où la seule tâche restante sur une machine est encore en traitement sur l'autre machine. Le second cas est lorsque le serveur est occupé et qu'une tâche attend d'être préparée sur une machine. On a $4nB$ unités de temps de préparation sur M_2. On doit éviter que le second cas augmente le temps mort sur M_1 à plus de nB unités de temps. Pour cela, $3nB$ parmi $4nB$ unités de temps de préparation sur M_2 doivent être ordonnancées en parallèle des phases de traitement des tâches sur M_1. Les intervalles de temps libres sur M_2 et qui sont en

parallèle des phases de traitement sur M_1 et ont une longueur totale de

$$\sum_{k=1}^{n}(p_{1,3n+k} - I_k) = 3nB - \sum_{k=1}^{n} I_k,$$

qui doit être égal à $3nB$ unités de temps. Donc,

$$\sum_{k=1}^{n}(p_{1,3n+k} - I_k) = 3nB - \sum_{k=1}^{n} I_k = 3nB.$$

On conclut que,

$$\sum_{k=1}^{n} I_k = 0.$$

On déduit, ainsi, que l'ordonnancement sera une succession d'intervalles où la phase de préparation d'une tâche de P' sur M_1 se termine exactement en même temps que la fin de la phase de traitement d'une autre tâche de P' sur M_2.

On a $3n$ unités de temps de préparation sur M_1 qui sont en parallèle des traitements des tâches de P' sur M_2. Deux possibilités peuvent survenir : soit deux préparations des tâches de P sont suivies d'une préparation d'une tâche de P', soit trois préparations de tâches de P seront ordonnancées dans cet intervalle, suivies de la préparation d'une tâche de P'. Dans les deux cas, la dernière préparation causera un temps mort sur M_1. Puisque le temps de traitement des tâches de P est nul alors soit la préparation de n tâches de P causeront nB unités de temps d'arrêt sur M_1 et qui seront ordonnancées vers la fin, soit les préparations des tâches de P' causeront ce temps d'arrêt. Les deux cas possibles sont représentés par la Figure 3.1.4a et la Figure 3.1.4b. Dans les deux cas, toutes les opérations sur M_2 sont ordonnancées de sort à ce qu'il n'y ait aucun temps mort supplémentaire sur M_1.

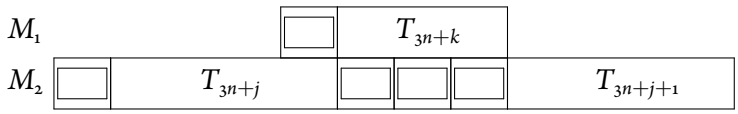


Figure 3.1.4a – Première possibilité

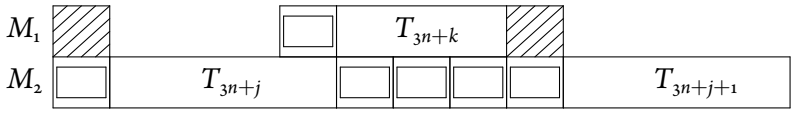


Figure 3.1.4b – Seconde possibilité

Il reste à ordonnancer sur M_1 les tâches de P. Cependant, il y a exactement n intervalles de temps libres

qui ont la même longueur, qui est égale à $4B$ unités de temps. On suppose que P_ℓ, $\ell = 1, \ldots, n$, sont les sous-ensembles des tâches de P traitées dans l'intervalle ℓ. Puisque la longueur des intervalles de temps ℓ est égale à $4B$ unités de temps alors,

$$\sum_{j \in P_\ell} (s_{1j} + p_{1j}) = 4B, \ell = 1, \ldots, n.$$

Ce qui fait que,

$$\sum_{j \in P_\ell} (a_j + B) = 4B, \ell = 1, \ldots, n.$$

On a alors,

$$\sum_{j \in P_\ell} (a_j + |P_\ell|B) = 4B, \ell = 1, \ldots, n.$$

On a n sous-ensembles et $\frac{B}{4} < a_j < \frac{B}{2}, j = 1, \ldots, 3n$, alors nécessairement $|P_\ell| = 3, \ell = 1, \ldots, n$. Par conséquent, $\sum_{j \in P_\ell} a_j = B, \ell = 1, \ldots, n$. En posant, $P_\ell = A_\ell, \ell = 1, \ldots, n$, on obtient une solution du problème 3-partition.

Ceci clôt la seconde partie de la preuve. Ainsi, nous avons montré que si le problème d'ordonnancement a une solution S avec $C_{\max}(S) = 9nB$ alors 3-partition a une solution. Nous avons, donc, prouvé le problème d'ordonnancement a une solution S avec $C_{\max}(S) = 9nB$ si, et seulement si le problème 3-partition admet une solution. □

3.1.3 Temps de préparation et de traitement proportionnels

Considérons, maintenant, le problème lorsque les temps de préparation et de traitement des deux opérations de chaque tâche $T_j, j = 1, \ldots, n$, sont identiques $(s_{1j} = s_{2j} = s_j, p_{1j} = p_{2j} = p_j)$ *c.-à-d.* proportionnels.

Théorème 3.3. *$O2, S1|s_{1j} = s_{2j} = s_j, p_{1j} = p_{2j} = p_j|C_{\max}$ est $\mathcal{NP}$-difficile au sens fort [12].*

Démonstration. Étant donnée une instance arbitraire de 3-partition, on construit l'instance suivante du $O2, S1|s_{1j} = s_{2j} = s_j, p_{1j} = p_{2j} = p_j|C_{\max}$ avec $7n$ tâches. L'ensemble de tâches est composé du sous-ensemble de tâches $P = \{T_1, \ldots T_{3n}\}$, et de $4n$ tâches supplémentaires devisées en 4 groupes, $G_1 = \{T_{3n+j}, j = 1, \ldots, n\}$, $G_2 = \{T_{4n+j}, j = 1, \ldots, n\}$, $G_3 = \{T_{5n+j}, j = 1, \ldots, n\}$ et $G_4 = \{T_{6n+j}, j = 1, \ldots, n\}$. Les temps de préparation et de traitement sont résumés dans le Tableau 3.1.2.

On montre, dans ce qui suit, que le problème $O2, S1|s_{1j} = s_{2j} = s_j, p_{1j} = p_{2j} = p_j|C_{\max}$ a une solution S avec $C_{\max}(S) = 26nB$ si, et seulement si, le problème 3-partition admet une solution. Pour cela, on montre,

	$T_j, j = 1, \ldots, 3n$	$T_j \in G_1$	$T_j \in G_2$	$T_j \in G_3$	$T_j \in G_4$
s_j	a_j	0	B	$4B$	$6B$
p_j	a_j	$3B$	$4B$	$6B$	0

Tableau 3.1.2 – Instance du $O2, S1|s_j, p_j|C_{\max}$

en premier lieu, que si le problème 3-partition a une solution alors le problème d'ordonnancement a une solution S avec $C_{\max}(S) = 26nB$. Ensuite, on montre que si le problème d'ordonnancement a une solution S dont le makespan est égal à $y = 26nB$ alors ceci implique que 3-partition a une solution.

D'abord, on suppose que 3-partition a une solution et A_ℓ, $\ell = 1, \ldots, n$ sont les sous-ensembles de $A = \{1, \ldots, 3n\}$ correspondants. Soient A'_ℓ les sous-ensembles de tâches définis comme, $A'_\ell = \{T_j, j \in A_\ell\}, \ell = 1, \ldots, n$. L'ordonnancement S ayant un makespan égal à $26nB$ peut être décrit comme suit.

On n'a aucun temps mort sur aucune des deux machines. L'ordre de tâches, sur M_1, est :

$$T_{3n+1}, T_{5n+1}, A'_1, T_{4n+1}, T_{6n+1}, \ldots, T_{4n}, T_{6n}, A'_n, T_{5n}, T_{7n};$$

tandis que, sur M_2, l'ordre est :

$$A'_1, T_{4n+1}, T_{6n+1}, T_{3n+1}, T_{5n+1}, \ldots, A'_n, T_{5n}, T_{7n}, T_{4n}, T_{6n}.$$

L'ordonnancement S résultant est une succession directe de n blocs, comme l'illustre la Figure 3.1.5.

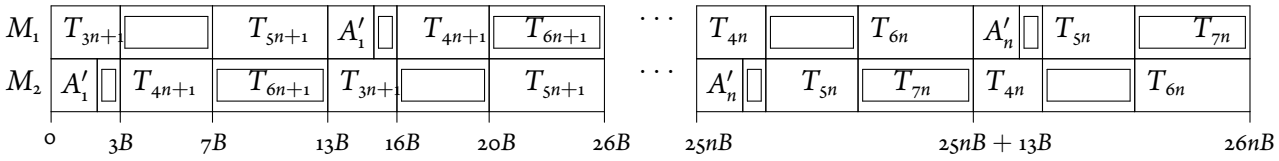


Figure 3.1.5 – La structure de l'ordonnancement

Maintenant, on suppose qu'il existe un ordonnancement S, avec $C_{\max}(S) = 26nB$ et on montre que 3-partition admet une solution. Pour cela, on décrit l'ordonnancement S et on montre qu'un tel ordonnancement ne peut exister que si le problème 3-partition a une solution. La structure de l'ordonnancement S vérifie ce qui suit :

- Il n'y a aucun temps mort sur les deux machines puisque le temps de fin de traitement final est égal

au temps de traitement total sur chaque machine, comme le montre la Figure 3.1.5.

- Puisque les tâches de G_1 sont les seules tâches ayant un temps de préparation nul, alors une tâche de G_1 doit être traitée en premier sur l'une des machines. Autrement, on aura un temps mort engendré par la préparation de la première tâche.
- La préparation de chaque tâche de G_4, sur les deux machines, doit être ordonnancée complètement en parallèle avec le traitement d'une tâche de G_3 sur l'autre machine. Dès lors que, c'est le seul temps de traitement qui soit aussi long que le temps de préparation d'une tâche de G_4.
- La préparation d'une tâche de G_3, sur les deux machines, sera ordonnancée complètement en parallèle du traitement d'une tâche de G_2 sur l'autre machine pour éviter d'avoir un temps mort. Puisque c'est la seule tâche qui n'est pas encore ordonnancée ayant un temps de traitement qui ne soit pas plus court que le temps de préparation des tâches de G_3.
- De plus, la préparation d'une tâche de G_2, sur les deux machines, doit être ordonnancée en parallèle du traitement d'une tâche de G_1. Il faut noter que, le traitement de la tâche de G_2 est complétement en parallèle avec la préparation de la tâche de G_3. Ce qui fait qu'il n'y a aucune intersection entre les intervalles où une tâche de G_2 est traitée et celui où une tâche de G_1 est traitée.
- On suppose que les traitements de deux tâches de G_1 sont ordonnancés en parallèle avec la préparation d'une tâche de G_4. Dans ce cas, la préparation de la tâche suivante, autre qu'une tâche de G_1, causerait un temps mort et de ce fait augmenterait la valeur du makespan. Maintenant, si on suppose qu'on ordonnance le traitement d'une tâche de G_3 suivi d'une tâche de G_1, en parallèle avec la préparation d'une tâche de G_4, alors la préparation de la tâche de G_3 causerait un temps mort qui augmenterait aussi la valeur du makespan.
- Le même scénario se produirait, si on choisit d'ordonnancer la préparation d'une tâche de G_3 en parallèle avec le traitement d'une tâche de G_1.

La structure de l'ordonnancement, par conséquent, est une succession directe de n ordonnancements partiels, comme celui représenté dans la Figure 3.1.6, avec $k = 0, \ldots, n-1$.

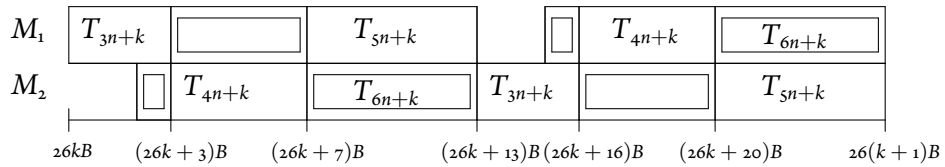


Figure 3.1.6 – Structure partielle de l'ordonnancement

On a n intervalles de temps libres sur les deux machines, tous de longueur $2B$. Cependant, il reste à ordonnancer les tâches de P sur les deux machines. On suppose que $T_j, j \in P_\ell$, $\ell = 1, \ldots, n$ correspondent aux

sous-ensembles de tâches traitées durant l'intervalle ℓ, donc, les tâches doivent vérifier ce qui suit :

$$\sum_{j\in P_\ell}(s_{1j}+p_{1j})=2B \Longleftrightarrow \sum_{j\in P_\ell}(2a_j)=2B \Longleftrightarrow \sum_{j\in P_\ell}a_j=B.$$

$$\sum_{j\in P_\ell}(s_{2j}+p_{2j})=2B \Longleftrightarrow \sum_{j\in P_\ell}(2a_j)=2B \Longleftrightarrow \sum_{j\in P_\ell}a_j=B.$$

On a n sous-ensembles et $\frac{B}{4}<a_j<\frac{B}{2}, j=1,\ldots,3n$, alors nécessairement $|P_\ell|=3, \ell=1,\ldots,n$. Il s'en suit que, $\sum_{j\in P_\ell}a_j=B, \ell=1,\ldots,n$. Ainsi, si on pose $P_\ell=A_\ell, \ell=1,\ldots,n$ alors on obtient une solution de 3-partition.

En conclusion, dans cette seconde partie de la preuve, on a montré que si le problème d'ordonnancement a une solution avec un makespan égal à $y=26nB$ alors 3-partition a une solution. Ainsi donc, on a montré que le problème d'ordonnancement a une solution S avec solution S avec $C_{\max}(S)=26nB$ si, et seulement si le problème 3-partition admet une solution. □

3.1.4 Temps de préparation et de traitement égaux

Finalement, nous présentons les derniers cas particuliers $\mathcal{NP}$-difficiles du $O2,S1|s_{ij}|C_{\max}$.

Cas $\mathcal{NP}$-difficile au sens fort

Le premier cas est celui où le temps de préparation d'une opération est égal à son temps de traitement. Ce sous-problème est noté $O2,S1|s_{1j}=p_{1j},s_{2j}=p_{2j}|C_{\max}$.

Théorème 3.4. *$O2,S1|s_{1j}=p_{1j},s_{2j}=p_{2j}|C_{\max}$ est $\mathcal{NP}$-difficile au sens fort [10, 13].*

Démonstration. Étant donnée une instance arbitraire du problème 3-partition, on construit l'instance suivante du $O2,S1|s_{1j}=p_{1j},s_{2j}=p_{2j}|C_{\max}$ avec $4n+1$ tâches. L'ensemble de tâches est composé du sous-ensemble $P=\{T_1,\ldots,T_{3n}\}$ et du sous-ensemble $P'=\{T_{3n+1},\ldots,T_{4n}\}$. Les temps de préparation et de traitement sont donnés dans le Tableau 3.1.3.

Dans ce qui suit, on montre que le problème de décision associé au problème $O2,S1|s_{1j}=p_{1j},s_{2j}=p_{2j}|C_{\max}$ a une solution S avec $C_{\max}(S)\leq y=(6n+1)B$ si, et seulement si, 3-partition admet une solution. Pour

	$P : T_j, j = 1, \ldots, 3n$	$P' : T_j, j = 3n+1, \ldots, 4n$	T_{4n+1}
s_{1j}	0	$3B$	$B/2$
p_{1j}	0	$3B$	$B/2$
s_{2j}	a_j	B	0
p_{2j}	a_j	B	0

Tableau 3.1.3 – Instance du $O2, S1|s_{1j} = p_{1j}, s_{2j} = p_{2j}|C_{\max}$

prouver ceci, on montre d'abord que si le problème 3-partition admet une solution alors le problème d'ordonnancement a une solution S avec $C_{\max}(S) = 6(n+1)B$. Par la suite, on montre que si le problème d'ordonnancement a une solution S ayant un makespan $C_{\max}(S) = 6(n+1)B$ alors cela implique que 3-partition a une solution.

On suppose, d'abord, que 3-partition admet une solution, et $A_\ell, \ell = 1, \ldots, n$, sont les sous-ensembles de $A = \{1, \ldots, 3n\}$ correspondants. Soient A'_ℓ les sous-ensembles des tâches définis comme, $A'_\ell = \{T_j, j \in A_\ell\}, \ell = 1, \ldots, n$. L'ordonnancement S peut être décrit comme suit.

— Le traitement sur M_1 est sans temps mort, selon l'ordre suivant :

$$(T_{3n+3}, T_{3n+4}, \ldots, T_{4n}, T_{3n+1}, T_{3n+2}, T_{4n+1}).$$

— Le temps mort total sur M_2 est égal à $(2n+1)B$ unités de temps. Le traitement, sur cette machine, commence à l'instant $t = 3B$, avec l'ordre suivant des tâches :

$$(T_j, j \in A_1, T_{3n+1}, T_j, j \in A_2, T_{3n+2}, \ldots, T_j, j \in A_{n-1}, T_{4n-1}, T_j, j \in A_n, T_{4n}).$$

— De plus, un temps mort, de $2B$ unités de temps, survient entre le traitement de chaque tâche $T_{3n+\ell}$ et les tâches $T_j, j \in A_\ell, \ell = 1, \ldots, n$.

L'ordonnancement S est tel que représenté dans la Figure 3.1.7.

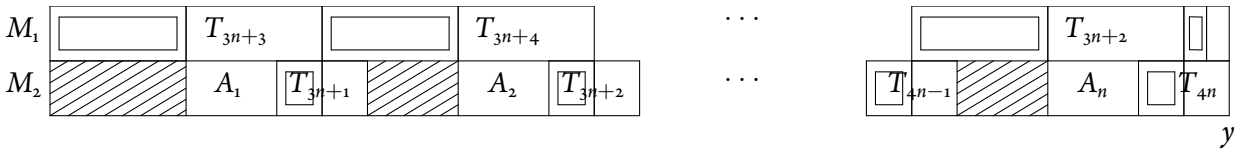


Figure 3.1.7 – La structure de l'ordonnancement

À présent, on suppose que le problème d'ordonnancement a une solution avec un makespan égal à $y = (6n+1)B$, et on montre qu'un tel ordonnancement ne peut exister que si 3-partition a une solution. L'ordonnancement a la structure suivante.

Il n'y a aucun temps mort sur M_1, vu que le temps de traitement total sur cette machine est égal au makespan. Le traitement commence sur M_1, à défaut de quoi, la première préparation causera un temps mort sur cette machine.

Les préparations sur M_2 seront ordonnancées en parallèle des phases de traitement sur M_1, sans quoi, on aura un temps mort sur M_1, dû à l'indisponibilité du serveur. Les temps de préparation des tâches de P' sont tous égaux à B. Les seuls temps de traitement supérieurs à B sont les temps de traitement des tâches de P' sur M_1. Il en résulte que, la préparation de chaque tâche de P' sur M_2 doit être ordonnancée en parallèle du traitement d'une autre tâche de P' sur M_1.

Le temps mort total sur M_2 est égal à $(2n+1)B$ unités de temps. Ce dernier est la différence entre le temps de fin de traitement final et le temps de traitement total sur cette machine. Tout temps mort supplémentaire augmentera la valeur du makespan.

Puisque le traitement commence sur M_1 alors la première préparation sur cette machine causera un temps mort sur M_2, qui est de $3B$ unités de temps. Aucune tâche ne peut commencer sur M_2 lorsque le serveur prépare les tâches sur M_1. Pour éviter que cet intervalle de temps soit un temps d'arrêt, une phase de traitement d'une tâche doit être lancée au même temps que le début de ces préparations. Les plus longs temps de traitement sur M_2 sont ceux des tâches de P'. Si la phase de traitement d'une tâche de P' sur M_2 commence au même temps que la phase de préparation d'une autre tâche de P' sur M_1, alors on obtient le temps mort le plus court possible sur M_2. Ce temps mort inévitable et égal à $3B + 2(n-1)B$ unités de temps. La structure partielle de l'ordonnancement est représentée par la Figure 3.1.8.

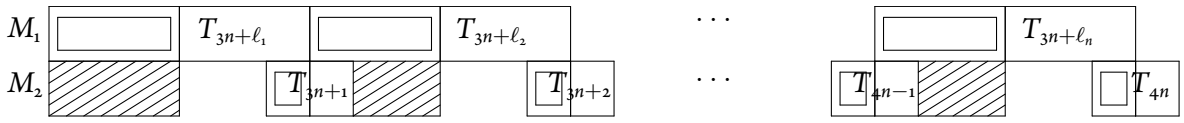


Figure 3.1.8 – La structure partielle de l'ordonnancement

Il ne reste, sur M_1, que l'ordonnancement de la tâche T_{4n+1}. On a deux possibilités : l'ordonnancer en première ou en dernière position. Si on l'ordonnance en première position alors le makespan dépassera y. Ce qui impose que, T_{4n+1} soit ordonnancée en dernière position.

Les tâches qui restent jusqu'ici sont les tâches de P sur M_2. Afin d'avoir un makespan égal à y, ces tâches doivent être ordonnancées dans les intervalles de temps libres sur M_2. On a n intervalles de temps libres de même longueur, égale à $2B$ unités de temps.

On suppose que $T_j, j \in P_\ell$ soient les tâches de P ordonnancées durant l'intervalle $\ell, \ell = 1, \ldots, n$. Les équations suivantes doivent être vérifiées lors de l'ordonnancement de ces tâches.

$$\sum_{j\in P_\ell}(s_{2j}+p_{2j})=2B, \ell=1,\ldots,n.$$

Donc,

$$\sum_{j\in P_\ell}2a_j=2B, \ell=1,\ldots,n.$$

Ainsi,

$$\sum_{j\in P_\ell}a_j=B, \ell=1,\ldots,n.$$

Si on définit les sous-ensembles $A_\ell = P_\ell$, alors $A_\ell, \ell = 1,\ldots,n$ représentent une solution de 3-partition.

Nous avons montré que, pour le problème d'ordonnancement décrit en haut, il ne peut exister une solution S avec $C_{\max}(S) = 6(n+1)B$ que si le problème 3-partition a une solution. Ceci clôt la preuve à travers laquelle on a montré que le problème d'ordonnancement a une solution S avec $C_{\max}(S) = 6(n+1)B$ si, et seulement si 3-partition a une solution. □

Cas $\mathcal{NP}$-difficile au sens ordinaire

On termine cette section avec le cas particulier du problème précèdent. Ce cas est noté $O2, S1|s_{1j} = p_{1j} = a, s_{2j} = p_{2j}|C_{\max}$.

Théorème 3.5. *$O2, S1|s_{1j} = p_{1j} = a, s_{2j} = p_{2j}|C_{\max}$ est $\mathcal{NP}$-difficile au sens ordinaire [13].*

Démonstration. Étant donnée une instance arbitraire du problème de partition, on construit l'instance suivante du $O2, S1|s_{1j} = p_{1j} = a, s_{2j} = p_{2j}|C_{max}$ comme suit. L'ensemble des tâches est composé du sous-ensemble $P = \{T_1,\ldots,T_n\}$, et de cinq tâches supplémentaires notées $P' = \{T_{n+k}, k = 1,\ldots,5\}$. Les temps de préparation et de traitement sont résumés dans le Tableau 3.1.4.

	$P: T_j, j=1,\ldots,n$	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
$s_{1j}=p_{1j}$	$5B$	$5B$	$5B$	$5B$	$5B$	$5B$
$s_{2j}=p_{2j}$	a_j	$(2n+4)5B-3B$	$3B$	$3B$	$3B$	$3/2B$

Tableau 3.1.4 – Instance du $O2, S1|s_{1j} = p_{1j} = a, s_{2j} = p_{2j}|C_{max}$

Dans ce qui suit, on montre que le problème de décision associé au problème $O2, S1|s_{1j} = p_{1j} = a, s_{2j} = p_{2j}|C_{\max}$ a une solution S avec $C_{\max}(S) \leq y = (4n+13)5B$ si, et seulement si, le problème de partition admet une solution.

Pour établir la preuve, on montre dans un premier temps que $5(4n+13)B$ est une borne inférieure du makespan de toute solution de l'instance donnée dans le Tableau 3.1.4. Ainsi, tout ordonnancement S ayant $C_{\max}(S) = 5(4n+13)B$ est une solution optimale du problème d'ordonnancement. On passe ensuite au reste de la preuve. On montre que si le problème de partition a une solution alors le problème d'ordonnancement a une solution S avec $C_{\max}(S) = 5(4n+13)B$. Finalement, on montre que si le problème d'ordonnancement admet une solution S avec $C_{\max}(S) = 5(4n+13)B$ alors ceci implique que le problème de partition a une solution.

On commence par l'énoncé du Lemme 3.1.

Lemme 3.1. $\sum_{j=1}^{n+5}(p_{2j}+s_{2j}) + 6B = 5(4n+13)B$ *est une borne inférieure de l'instance donnée dans le Tableau 3.1.4.*

Démonstration. Les temps morts peuvent être dû à l'indisponibilité du serveur. Ceci se produit lorsque le serveur prépare une tâche sur une machine et qu'une autre tâche est en attente de préparation sur l'autre machine. Puisque tous les temps de préparation sont non nuls alors lorsque le serveur est occupé aucune autre tâche ne peut commencer sur l'autre machine. Pour éviter que ces périodes ne soient un temps d'arrêt, on doit ordonnancer les phases de préparation en parallèle avec les phases de traitement.

Sur M_1, on a $(n+5)$ temps de préparation tous égaux à $5B$ unités de temps. Pour éviter que ces temps de préparation ne causent un temps mort sur l'autre machine, elles doivent être ordonnancées en parallèle de temps de traitement sur M_2. On peut ordonnancer sur M_1, en parallèle du traitement de T_{n+1} sur M_2, $n+2$ tâches parmi les $n+5$ tâches ayant des temps de préparation identiques sur M_1 comme l'illustre la Figure 3.1.9. Cependant, trois autres préparations, celle de T_{n+1} incluse, ne seront pas ordonnancées en parallèle du traitement de T_{n+1} sur M_2.

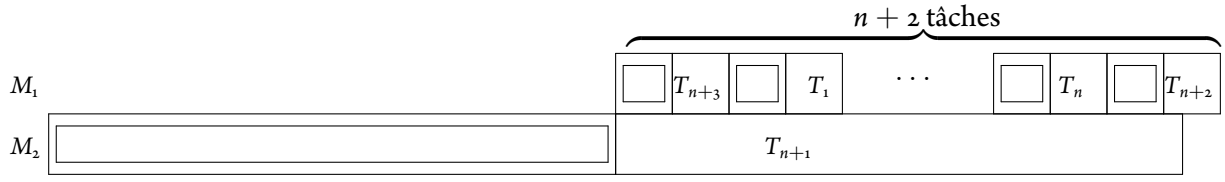


Figure 3.1.9 – Ordonnancement de T_{n+1} sur M_2

Les temps de traitement les plus longs sur M_2, autre que celui de T_{n+1}, sont les temps de traitement des tâches de P'. Si la phase de traitement des tâches de P' sur M_2 commence en même temps que la phase de préparation de l'une des trois tâches restantes sur M_1, alors on obtient un temps mort minimum sur M_2. Ce temps mort est égal à $6B$ unités de temps.

On a $6B$ unités de temps mort inévitables sur M_2. Ainsi, ce qui suit est clairement une borne inférieure du makespan,

$$\sum_{j=1}^{n+5}(s_{2j}+p_{2j})+6B=5(4n+13)B.$$

Par conséquent, l'énoncé du lemme est établi.

□

À présent, on suppose que le problème de partition a une solution et A_1, A_2 sont les sous-ensembles de A correspondants. On définit A'_1 et A'_2 comme étant les sous-ensembles des tâches définis comme, $A'_1 = \{T_j, j \in A_1\}$, $A'_2 = \{T_j, j \in A_2\}$. Puis, on montre que le problème d'ordonnancement a une solution S avec $C_{\max}(S) = 5(4n+13)B$.

L'ordonnancement S est décrit comme suit :

- On commence avec T_{n+2} sur M_2 suivie de T_{n+3} sur M_1 et puis T_{n+1} sur M_2.
- En parallèle du traitement de T_{n+1} sur M_2, la machine M_1 traite $(n+2)$ tâches parmi les $(n+5)$ tâches identiques.
- Par la suite, on ordonnance T_{n+4} sur M_2, puis T_{n+1} sur M_1.
- La machine M_2 traite A'_1 suivi de T_{n+3}.
- Finalement, on ordonnance T_{n+2} sur M_1, puis, la machine M_2 traite T_{n+5} ensuite A'_2.

L'ordonnancement est représenté par la Figure 3.1.10.

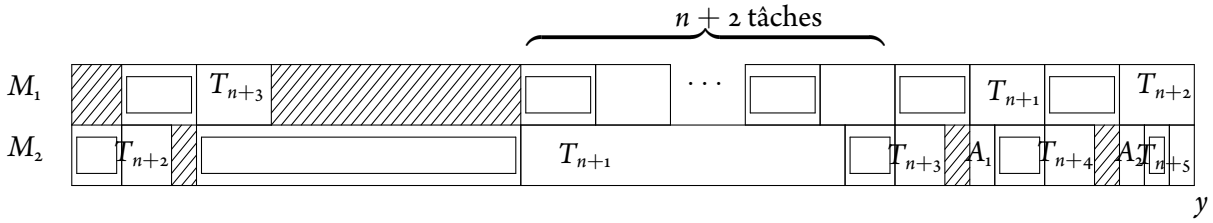


Figure 3.1.10 – La structure de l'ordonnancement

Maintenant, on suppose que le problème d'ordonnancement a une solution S, avec $C_{\max}(S) \leq y = 5(4n+13)B$, et on montre qu'un tel ordonnancement ne peut exister que si le problème de partition admet une solution. L'ordonnancement, dans ce cas, est comme suit.

Sur la machine M_1, on a $(n+2)$ tâches à partir de $(n+5)$ tâches identiques qui seront traitées en parallèle avec le traitement de T_{n+1} sur M_2, afin d'éviter un temps mort supplémentaire, comme expliqué dans la

démonstration du Lemme 3.1. Supposons que, de plus que T_{n+1}, les tâches qui ne sont pas encore ordonnancées sur M_1 sont T_{n+2} et T_{n+3}.

La phase de traitement des tâches T_{n+2} , T_{n+3} et T_{n+4} sur M_2 commenceront au même instant que les phases de préparation de T_{n+3}, T_{n+1} et T_{n+2} sur M_1 respectivement, autrement le temps mort sur M_2 dépassera $6B$ (voir démonstration du Lemme 3.1).

À partir des points précédents, on obtient les blocs suivants :

- BL_1 : (T_{n+3} sur M_2, T_{n+1} sur M_1).
- BL_2 : (T_{n+2} sur M_2, T_{n+3} sur M_1).
- BL_3 : (T_{n+4} sur M_2, T_{n+2} sur M_1).
- BL_4 : (T_{n+1} sur M_2, $T_j, j = 1, \ldots, n, n+4, n+5$ sur M_1).

Il faut noter aussi que, le temps mort sur M_1 ne peut pas dépasser $5(2n+4)B - 5B$ unités de temps, qui est la différence entre le makespan et le temps de traitement total sur cette machine. Tout temps mort supplémentaire augmentera la valeur du makespan. Il y a une condition à vérifier lors de l'ordonnancement des blocs, pour éviter un surplus de temps mort. La condition est que la dernière tâche du bloc ne soit pas la même avec la première tâche du bloc qui le succède. En vérifiant cette condition et avec BL_4 en première position, le temps mort sur M_1 sera égal à $5(2n+4)B - 3B$ qui est supérieure à $5(2n+4)B - 5B$. Il en résulte que, la seconde condition, à vérifier, est que BL_4 ne doit pas être ordonnancé en première position.

Puisque BL_4 est toujours précédé par un autre bloc alors, sans perte de généralité, on suppose que BL_2 est le bloc qui le précède. Le bloc BL_2 peut être permuté avec BL_3 et ceci ne modifie ni la structure ni la longueur de l'ordonnancement. On redéfinit les blocs comme suit :

- BL_1 : (T_{n+3} sur M_2, T_{n+1} sur M_1).
- BL_2 : (T_{n+4} sur M_2, T_{n+2} sur M_1, T_{n+1} sur M_2, $T_i, i = 1, \ldots, n, n+4, n+5$ sur M_1).
- BL_3 : (T_{n+4} sur M_2, T_{n+2} sur M_1).

La structure de l'ordonnancement dépend de la position de BL_2. On peut ordonnancer BL_3 avant BL_2 puisque la dernière tâche de BL_3 est T_{n+2} qui est aussi la première tâche de BL_2. On aura donc trois ordres possibles, qui sont :

- BL_1, BL_2, BL_3.
- BL_2, BL_1, BL_3.
- BL_3, BL_1, BL_2.

L'ordre des blocs illustré dans la Figure 3.1.11 est BL_2, BL_1, BL_3.

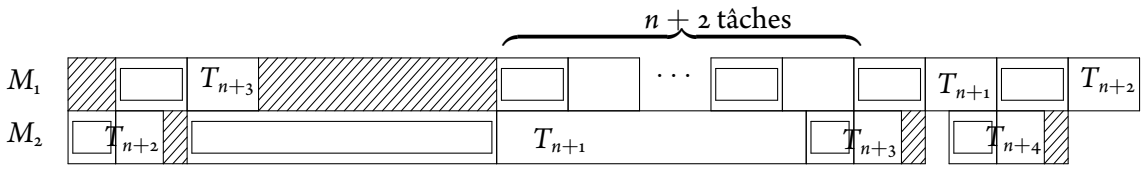


Figure 3.1.11 – La structure partielle de l'ordonnancement

Il ne reste qu'à ordonnancer T_{n+5} et les tâches de P sur M_2. On fait remarquer qu'on a deux intervalles de temps libres dans les deux premiers cas et trois dans le troisième cas. La tâche T_{n+5} peut uniquement être ordonnancée dans un seul intervalle, qui est celui qui a une longueur supérieure à la durée d'exécution de cette tâche. Ainsi, elle sera ordonnancée en parallèle de T_{n+2} sur M_1, dans les deux premiers cas. Tandis que dans le troisième cas elle sera ordonnancée en parallèle de la dernière opération sur M_1.

Après avoir ordonnancé T_{n+5} sur M_2, il ne nous reste que deux intervalles de temps libres dans les trois cas, qui sont de la même longueur, $2B$ unités de temps. Pour ordonnancer les tâches de P sur M_2 et en supposant que P_1 et P_2 soient les sous-ensembles de tâches ordonnancées dans le premier et le second intervalle, respectivement, les tâches doivent vérifier :

$$\sum_{T_j \in P_1}(s_{1j} + p_{1j}) = 2B \Longleftrightarrow \sum_{T_j \in P_1}(2a_j) = 2B \Longleftrightarrow \sum_{T_j \in P_1} a_j = B.$$

$$\sum_{T_j \in P_2}(s_{1j} + p_{1j}) = 2B \Longleftrightarrow \sum_{T_j \in P_2}(2a_j) = 2B \Longleftrightarrow \sum_{T_j \in P_2} a_j = B.$$

Si on définit $A_1 = P_1, A_2 = P_2$ alors A_1 et A_2 représentent une solution du problème de partition.

Ainsi, on a montré que si le problème d'ordonnancement a une solution S avec $C_{\max}(S) = 5(4n + 13)B$ alors le problème de partition a une solution. Ce qui clôt la preuve par laquelle on a montré que le problème d'ordonnancement présenté en haut admet une solution S avec $C_{\max}(S) = 5(4n + 13)B$ si, et seulement si le problème de partition a une solution.

□

3.2 Job shop à deux machines et un serveur

Cette section est consacrée aux résultats de complexité relatifs au problème $J2, S1|s_{ij}|C_{\max}$. Nous montrons la $\mathcal{NP}$-complétude au sens fort de deux cas particuliers du problème *c.-à-d.* avec des temps de traitement

identiques, noté $F2, S1|s_{ij}, p_{ij} = p|C_{\max}$, et avec des temps de préparation identiques, noté $J2, S1|s_{ij} = s|C_{\max}$.

3.2.1 Temps de traitement identiques

Nous commençons par le sous-problème avec des temps traitement identiques. Nous montrons que même lorsque toutes les tâches ont le même cheminement (flow shop) et avec des temps de traitement identiques le problème est $\mathcal{NP}$-difficile.

Théorème 3.6. *$F2, S1|s_{ij}, p_{ij} = p|C_{\max}$ est $\mathcal{NP}$-difficile au sens fort [14].*

Démonstration. Étant donnée une instance arbitraire de 3-partition, on définit l'instance suivante du $F2, S1|s_{ij}, p_{ij} = p|C_{\max}$ avec $3n+2$ tâches. Les temps de traitement sont tous de B unités de temps. Les temps de préparation sont résumés dans le Tableau 3.2.1.

	$P : T_j, j = 1, \ldots, 3n$	$P' : T_{k_1}, \ldots, T_{k_{n+1}}$	T_a, T_β	T_γ
s_{1j}	a_j	0	0	$(n+2)B$
s_{2j}	0	$2B$	0	0

Tableau 3.2.1 – Temps de préparation de l'instance du $F2, S1|s_{ij}, p_{ij} = p|C_{\max}$

Pour montrer que le problème $F2, S1|s_{ij}, p_{ij} = p|C_{\max}$ est $\mathcal{NP}$-complet au sens fort, on montre que le problème de décision associé a une solution S avec $C_{\max}(S) \leq y = (6n+7)B$ si, et seulement si, la 3-partition a une solution.

Afin de construire cette preuve, on montre dans un premier temps que si 3-partition a une solution alors le problème d'ordonnancement a une solution S avec $C_{\max}(S) = (6n+7)B$. Par la suite, on montre que si le problème d'ordonnancement admet une solution S avec un makespan égal à $y = (6n+7)B$ alors ceci implique que 3-partition a une solution.

On suppose d'abord que 3-partition a une solution, avec $A_\ell, \ell = 1, \ldots, n$, les sous-ensembles de A. Soient A'_ℓ les sous-ensembles de tâches définis comme, $A'_\ell = \{T_j, j \in A_\ell\}, \ell = 1, \ldots, n$. L'ordonnancement S est comme suit.

Il n'y a aucun temps mort sur les deux machines. L'ordre des tâches sur M_1 est le suivant :

$$T_{k_1} \in P', T_a, T_{j_1}, T_{j_2}, T_{j_3} \in A'_1, T_{k_2} \in P', T_{j_4}, T_{j_5}, T_{j_6} \in A'_2, \ldots, T_{j_{3n-2}}, T_{j_{3n-1}}, T_{j_{3n}} \in A'_n, T_\beta, T_\gamma.$$

Tandis que l'ordre sur M_2 est le suivant :

$$T_{k_1}, T_a, T_{j_1}, T_{k_2}, T_{j_2}, T_{j_3}, T_{k_3}, \ldots T_{k_n}, T_{j_{2n-2}}, T_{j_{2n-1}}, T_{k_{n+1}}, T_{j_{2n}}, \ldots T_{j_{3n}}, T_\beta, T_\gamma.$$

L'ordonnancement réalisable S avec $C_{\max}(S) = (6n+7)B$ est représenté dans la Figure 3.2.1.

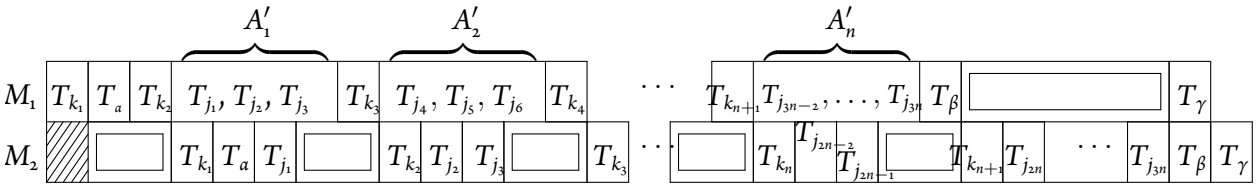


Figure 3.2.1 – La structure de l'ordonnancement

À présent, on suppose que le problème d'ordonnancement a une solution avec un makespan égal à $y = (6n+7)B$, et on montre que 3-partition admet une solution. On décrit, dans ce qui suit, l'ordonnancement S et on montre qu'un tel ordonnancement ne peut exister que si 3-partition a une solution. La structure de l'ordonnancement S est la suivante.

- Aucun temps mort ne se produit pendant le traitement des tâches sur les deux machines. Puisque toutes les tâches ont le même cheminement $M_1 \rightarrow M_2$ alors le traitement commence sur M_1. Le temps de fin de traitement sur cette machine est égal à $(6n+6)B$, qui sera suivi du traitement de la dernière tâche sur M_2. Tandis que sur M_2, le traitement commence à l'instant $t = B$ et le temps de fin traitement sur cette machine est égal au makespan. Ainsi, tout temps mort sur une machine augmentera la valeur du makespan à plus de $(6n+7)B$ unités de temps.
- Parallèlement à la préparation de chaque tâche de P' sur M_2 est ordonnancée une tâche avec un temps de préparation nul, de sorte que le traitement de ce dernier se termine en même temps que la préparation. Si ce n'est pas le cas, alors M_1 sera inactive pendant les B dernières unités de temps de la préparation sur M_2. Puisque les tâches à temps de préparation nul sur M_1 sont les tâches de P' et T_a, T_β alors soit une tâche de P' ou T_a, T_β sur M_1 sera ordonnancée en parallèle de chaque préparation d'une tâche de P' sur M_2. De plus, parallèlement à la préparation de la dernière tâche de P' sur M_2 sera ordonnancée soit T_a soit T_β. On suppose, sans perte de généralité, que cette dernière est T_β.
- La première tâche ordonnancée est une tâche de P'. Si nous supposons que ce n'est pas le cas et que la première tâche est T_a alors nous aurons les deux cas suivants. Le premier cas est que la deuxième tâche est une tâche de P, par conséquent, M_2 sera inactive en attendant la fin de la tâche de P sur M_1. Le deuxième cas est qu'une tâche de P' est la deuxième tâche. Afin d'éviter que M_1 ne soit inactive pendant la préparation de cette tâche, au moins deux tâches de P' seront ordonnancées en parallèle

de cette préparation. Ce qui contredit le point précédent où nous avons montré qu'en parallèle de la préparation d'une tâche de P' sur M_2 est ordonnancée une tâche ayant un temps de préparation nul sur M_1. En effet, nous avons au moins un temps mort sur M_1 de B unités de temps dû à la préparation d'une tâche P'. Ainsi, dans les deux cas, lorsque la première tâche est T_α (ou T_β), un temps mort se produit sur l'une des machines. Certes, la première tâche sur M_1 doit être une tâche ayant un temps de préparation nul, cependant, elle ne peut pas être T_α ni T_β et donc elle doit être une tâche de P'.

- Pour les mêmes raisons que dans le point précédent, la deuxième tâche traitée sur M_1 est T_α (T_β est supposée être ordonnancée en parallèle de la dernière tâche de P' sur M_2). Si nous supposons que ce n'est pas le cas, alors ordonnancer une tâche de P en deuxième position engendrera un temps mort sur M_1. Tandis que l'ordonnancement d'une tâche de P' en deuxième position engendrera un temps mort sur M_1.
- On définit les $(n+1)$ paires de tâches, comme représentées sur la Figure 3.2.2, comme étant les paires, Pr_ℓ, $\ell = 1, \ldots, n$. Nous supposons, sans perte de généralité, qu'elles sont traitées selon l'ordre de l'indice de la tâche de P' sur M_2.
- La tâche T_γ est ordonnancée après les tâches de P'. Parallèlement à sa préparation sur M_1, $(n+2)$ tâches doivent être traitées sur M_2. D'abord, on nomme la distance entre le traitement d'une tâche sur M_1 et son traitement sur M_2 longueur de décalage dans l'ordonnancement. La longueur du décalage après l'ordonnancement de T_α (voir Figure 3.2.2) est égale à $2B$ unités de temps. Elle augmentera de B unités de temps après le traitement de chaque paire Pr_ℓ, $\ell = 1, \ldots, n$, et n'atteindra $(n+2)B$ unités de temps qu'après avoir ordonnancé toutes les paires. Afin de s'assurer que $(n+2)$ tâches sont traitées sur M_2 en parallèle de la préparation de T_γ sur M_1, toutes les paires doivent être déjà traitées. Ainsi, T_{4n+5} est ordonnancée après les tâches de P' sur M_1.
- De plus, le traitement de la tâche de P' de la dernière paire est complètement parallèle de la préparation de T_γ. Nous supposons que ce n'est pas le cas, et que parallèlement au traitement de la dernière tâche de P' est ordonnancée une tâche de P, disons T_j, alors cela augmentera certainement la valeur du makespan de $B + a_j$ unités de temps.
- Si une tâche de P est ordonnancée après T_γ alors M_2 sera inactive en attendant la fin de traitement de la tâche sur M_1.
- La forme de l'ordonnancement, tel qu'il est décrit par les points précédents, est illustrée par la Figure 3.2.2.

- Les paires Pr_ℓ, $\ell = 1, \ldots, n$, ne sont pas traitées directement les unes après les autres. Sinon, M_1 sera inactive pendant l'intervalle de temps entre l'achèvement de la première paire et le début de la tâche de la paire traitée après. Pour s'assurer que les machines sont occupées durant ces intervalles de temps, au moins une tâche, pas encore ordonnancée, sera traitée sur M_1 et M_2.

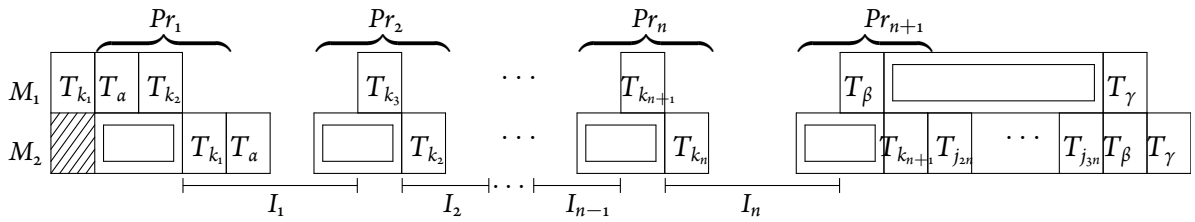


Figure 3.2.2 – La structure partielle de l'ordonnancement

- Entre chaque paire $Pr_\ell, \ell = 1, \ldots, n+1$, nous ordonnançons les tâches de P sur M_1. Soit I_ℓ l'intervalle de temps après $Pr_\ell, \ell = 1, \ldots, n$ comme illustré dans la Figure 3.2.2. La longueur de ces intervalles est multiple de B, puisque les tâches ordonnancées sur M_2 ont un temps de traitement de B unités de temps. Les tâches ordonnancées précédemment créent un décalage de $2B$ d'unités de temps après la première paire. De plus, le décalage augmente de B unités de temps après le traitement de chaque paire. Ainsi, cela garantira qu'il n'y aura pas d'intersection entre le traitement d'une tâche sur M_1 et sa préparation sur M_2. De plus, nous avons $\frac{B}{4} < a_j < \frac{B}{2}, j = 1, \ldots, 3n$, ainsi la somme des temps de préparation de deux tâches de P est inférieure à B, tandis que la somme des temps de préparation de quatre tâches de P est supérieure à B. Ce qui implique qu'exactement trois tâches de P sont ordonnancées durant $I_\ell, \ell = 1, \ldots, n$, de plus, la somme de leur temps de préparation est égale à B.

Soient $T_j, j \in P_\ell$ les sous-ensembles de tâches traitées sur M_1 durant $I_\ell, \ell = 1, \ldots, n$. Si nous définissons les sous-ensembles $A_\ell = P_\ell$ alors $A_\ell, \ell = 1, \ldots, n$ représentent une solution de 3-partition.

On a ainsi, montré que si le problème d'ordonnancement, tel que décrit en haut, a une solution S avec un makespan est égal à $y = (6n + 7)B$ alors 3-partition a une solution. En conclusion, on a montré que le problème d'ordonnancement a une solution S avec $C_{\max}(S) = (6n + 7)B$ si, et seulement si 3-partition a une solution. □

3.2.2 Temps de préparation identiques

Le dernier résultat de complexité, que nous présentons dans ce chapitre, concerne le cas particulier avec des temps de préparations identiques.

Théorème 3.7. *$J2, S1|s_{ij} = s|C_{\max}$ est $\mathcal{NP}$-difficile au sens fort [14].*

Démonstration. Étant donnée une instance arbitraire du problème 3-partition, on définit l'instance suivante du $J2, S1|s_{ij} = s|C_{\max}$, avec $4n + 1$ tâches. L'ensemble des tâches est partitionné en deux sous-ensembles.

Le sous-ensemble P contient les tâches $T_1, \ldots, T_{3n}$. Tandis que, le sous-ensemble P' contient les tâches, $T_{3n+1}, \ldots, T_{4n+1}$. Les temps de préparation sont tous égaux à B. Quant aux temps de traitement ainsi que les cheminements des tâches sont résumés dans le Tableau 3.2.2. La valeur de c est définie comme suit $\max_{1 \leq j \leq 3n}\{a_j\} < c < B$.

	$T_j, j = 1, \ldots, 3n$	$T_j, j = 3n+1, \ldots, 4n-2$	T_{4n-1}	T_{4n}	T_{4n+1}
p_{1j}	a_j	$5B$	$B + c$	$5B$	$5B$
p_{2j}	a_j	$5B$	$5B$	$5B$	c
R_j	$M_2 \to M_1$	$M_2 \to M_1$	$M_2 \to M_1$	$M_1 \to M_2$	$M_1 \to M_2$

Tableau 3.2.2 – Temps de traitement de l'instance du $J2, S1|s_{ij} = s|C_{\max}$

Pour montrer que le problème $J2, S1|s_{ij} = s|C_{\max}$ est $\mathcal{NP}$-complet au sens fort, on montre que le problème de décision qui lui correspond a une solution S avec $C_{\max}(S) \leq y = (10n + 2)B + c$ si et seulement si 3-partition a une solution. Pour cela, on montre d'abord que si le problème 3-partition a une solution alors le problème d'ordonnancement admet une solution S avec un makespan égal à $y = (10n + 2)B + c$. Par la suite, on montre que si le problème d'ordonnancement a une solution S avec $C_{\max}(S) = y$ alors ceci implique que 3-partition a une solution.

On commence par supposer que 3-partition a une solution, avec $A_\ell, \ell = 1, \ldots, n$, les sous-ensembles de A. On définit aussi A'_ℓ, comme étant les sous-ensembles de tâches $A'_\ell = \{T_j, j \in A_\ell\}, \ell = 1, \ldots, n$. Puis, on montre que le problème d'ordonnancement admet une solution S avec $C_{\max}(S) = (10n + 2)B + c$. L'ordonnancement S peut être décrit comme suit.

Le traitement commence sur M_1, et le temps de traitement total sur cette machine est égal à y. L'ordre des tâches sur M_1 est le suivant :

$$T_{4n}, A'_1, T_{4n+1}, A'_2, T_{k_1} \in P', A'_3, \ldots, T_{k_{n-1}} \in P', A'_n, T_{4n-1}.$$

Sur M_2, la première tâche traitée doit attendre que le serveur redevienne libre. Ainsi, le traitement commence à l'instant $t = B$, après lequel aucun temps mort n'est admis sur cette machine. La fin de traitement sur M_2 est égal à y et la séquence des tâches est la suivante :

$$A'_1, T_{k_1} \in P', A'_2, T_{k_2} \in P', \ldots, T_{k_n} \in P', T_{4n+1}.$$

L'ordonnancement S est illustré dans la Figure 3.2.3.

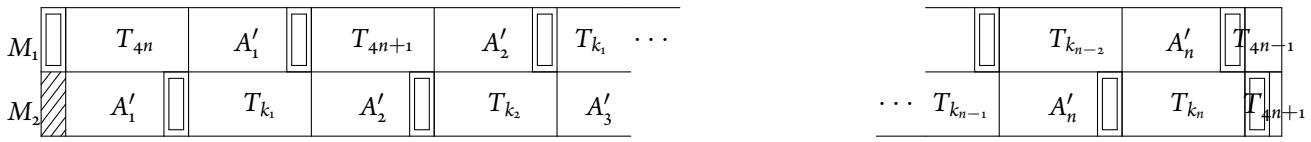


Figure 3.2.3 – La structure de l'ordonnancement

Nous passons, à présent, à la seconde partie de la preuve. Ainsi, on suppose que le problème d'ordonnancement admet une solution avec un makespan égal à $y = (10n + 2)B + c$, et on montre que 3-partition a une solution. Afin de prouver cela, nous allons décrire la structure de l'ordonnancement S et montrer qu'un tel ordonnancement ne peut exister que si 3-partition a une solution. L'ordonnancement S est décrit, dans les points suivants.

- Le traitement commence sur M_1, et son temps de traitement total est égal à y. Par conséquent, tout temps mort qui aura lieu sur cette machine augmenterait la valeur du makespan. Ainsi, aucun temps mort n'est permis sur M_1.
- Le traitement sur M_2 commence à l'instant $t = B$, puisque le serveur est occupé à préparer la première tâche sur M_1, et il n'y a aucune tâche avec un temps de préparation nul. Le temps de fin de traitement sur les deux machines est le même et égal à y.
- Chaque préparation sur M_1, hormis la première, et chaque préparation sur M_2 doivent être ordonnancées en parallèle de la phase de traitement d'une tâche sur l'autre machine, et ce pour éviter le temps mort dû à l'indisponibilité du serveur. Plus précisément, ces préparations doivent être ordonnancées en parallèle du traitement d'une tâche de P', puisque ce sont les tâches ayant des temps de traitement supérieur à B unités de temps.
- Si une tâche de P est ordonnancée en parallèle de T_{4n-1} sur M_1 ou de T_{4n+1} sur M_2 alors on aura l'une des situations suivantes. La première est qu'un temps mort sera engendré par la préparation de la tâche ordonnancée juste après. La seconde est que le temps de fin de traitement ne sera pas le même sur les deux machines. Donc, le temps de fin de traitement sur l'une ou les deux machines dépassera y.
- Comme nous l'avons expliqué précédemment la préparation de chaque tâche de P' est ordonnancée en parallèle du traitement d'une autre tâche de P'. À présent, nous allons montrer qu'il n'y a pas d'intersection entre les intervalles de temps où deux tâches de P', ayant un temps de traitement de $5B$, sont traitées. Sans perte de généralité, on définit I_k comme étant la longueur de l'intersection entre le traitement de la $k^{éme}$ tâche de P' sur M_1 est le traitement de k^{me} tâche de P' sur M_2. En conséquence, la longueur totale des intervalles de temps libres en parallèle du traitement d'une tâche de P' (ayant un temps de traitement de $5B$ unités de temps) est de $(4nB - \sum_{k=1}^{n} I_k)$ unités de temps sur M_1 et sur M_2. Cependant, le temps de préparation et de traitement des tâches de P qui seront entièrement

ordonnancées en parallèle de ces intervalles est de $4nB$ unités de temps sur les deux machines. Il est évident que, pour pouvoir ordonnancer toutes les préparations en parallèle des traitements des tâches de P' on doit avoir $\sum_{k=1}^{n} I_k = 0$.

- Les préparations des tâches de P sont ordonnancées en parallèle des traitements des tâches de P' ayant un temps de traitement de $5B$ unités de temps. De surcroit, la préparation de T_{4n-1} sur M_1 est aussi en parallèle du traitement d'une tâche de P' ayant un temps de traitement supérieur à B unités de temps. Pour éviter le temps mort que peut causer T_{4n-1}, celle-ci doit être ordonnancée en dernière position sur M_1. De plus, T_{4n+1} sera ordonnancée, sur M_2, complètement en parallèle de T_{4n-1}.
- Les premières tâches traitées sur M_1 sont T_{4n} et T_{4n+1} puisque ce sont les seules tâches ayant le cheminement $M_1 \rightarrow M_2$.
- On fait remarquer que les tâches T_{4n-1} et T_{4n} sur M_2 sont l'une des tâches T_{k_ℓ}, $\ell = 1, \ldots, n-1$ et $\ell = 2, \ldots, n$, respectivement.

La structure partielle de l'ordonnancement est représentée dans la Figure 3.2.4.

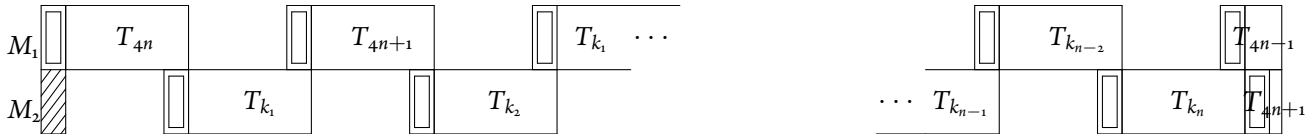


Figure 3.2.4 – Structure partielle de l'ordonnancement

Il reste à ordonnancer, sur les deux machines, les tâches de P. Cependant, il ne reste que n intervalles de temps libres de même longueur $4B$ unités de temps. Soient $T_j, j \in P_\ell$, $\ell = 1, \ldots, n$, les sous-ensembles des tâches de P ordonnancées dans l'intervalle de temps ℓ. Afin de respecter le cheminement des tâches, les mêmes tâches ordonnancées dans l'intervalle ℓ sur M_2 seront ensuite traitées dans l'intervalle ℓ sur M_1. Puisque la longueur de chaque intervalle est égale à $4B$ unités de temps alors,

$$\sum_{j \in P_\ell} (s_{2j} + p_{2j}) = 4B, \ell = 1, \ldots, n.$$

Il s'en suit que,

$$\sum_{j \in P_\ell} (a_j + B) = 4B, \ell = 1, \ldots, n.$$

Ainsi,

$$\sum_{j \in P_\ell} (a_j + |P_\ell| B) = 4B, \ell = 1, \ldots, n.$$

On a n sous-ensembles et $\frac{B}{4} < a_j < \frac{B}{2}, j = 1, \ldots, 3n$, alors nécessairement $|P_\ell| = 3, \ell = 1, \ldots, n$. Par conséquent, $\sum_{j \in P_\ell} a_j = B, \ell = 1, \ldots, n$. Ainsi, $P_\ell = A_\ell, \ell = 1, \ldots, n$, est une solution de 3-partition. Par conséquent, on a montré que si le problème d'ordonnancement a une solution S avec $C_{\max}(S) = (10n + 2)B + c$ alors le problème 3-partition a une solution. Ceci est la seconde partie de la preuve par laquelle on a montré que le problème d'ordonnancement admet une solution S avec un makespan égal à $y = (10n + 2)B + c$ si, et seulement si 3-partitiona une solution. □

Conclusion

Dans ce chapitre, nous avons présenté les résultats de complexité issus de l'étude faite sur les deux problèmes que nous avons traités, *c.-à-d.* l'open shop et le job shop. Le Tableau 3.2.3, résume les sous-problèmes étudiés ainsi que leur complexité.

Sous-problème	Complexité
$O2, S1 \mid s_{ij} = s \mid C_{\max}$	$\mathcal{NP}$-difficile au sens fort [9, 12]
$O2, S1 \mid s_{ij}, p_{ij} = p \mid C_{\max}$	$\mathcal{NP}$-difficile au sens fort [9, 12]
$O2, S1 \mid s_{1j} = s_{2j} = s_j, p_{1j} = p_{2j} = p_j \mid C_{\max}$	$\mathcal{NP}$-difficile au sens fort [12]
$O2, S1 \mid s_{1j} = p_{1j}, s_{2j} = p_{2j} \mid C_{\max}$	$\mathcal{NP}$-difficile au sens fort [10, 13]
$O2, S1 \mid s_{1j} = p_{1j} = a, s_{2j} = p_{2j} \mid C_{\max}$	$\mathcal{NP}$-difficile au sens ordinaire [13]
$J2, S1 \mid s_{ij} = s \mid C_{\max}$	$\mathcal{NP}$-difficile au sens fort [11, 14]
$F2, S1 \mid s_{ij}, p_{ij} = p \mid C_{\max}$	$\mathcal{NP}$-difficile au sens fort [14]

Tableau 3.2.3 – Résultats de complexité

4

Résolution de cas particuliers

Nous avons vu dans le chapitre précédent que les deux problèmes sont $\mathcal{NP}$-difficiles au sens fort et ce même après la relaxation du problème. Cependant, certains cas particuliers se sont avérés résolubles en temps polynomial. Dans ce chapitre nous énonçons, pour chacun des deux problèmes étudiés, les cas résolubles en temps polynomial. On commence par ceux de l'open shop, ensuite, on passera à ceux du job shop.

4.1 Cas particuliers de l'open shop

Nous considérons, dans ce qui suit, les deux cas particuliers de l'open shop à deux machines et un serveur, que nous montrons résolubles en temps polynomial. Le premier est le cas avec des temps de préparation identiques et des temps de traitement identiques. Le second cas est celui où les durées d'exécution sont identiques sur chacune des deux machines.

4.1.1 Temps de préparation et de traitement identiques

Nous présentons ici le cas avec des temps de préparation identiques et des temps de traitement identiques, qu'on note $O2, S1|s_{ij} = s, p_{ij} = p|C_{\max}$. Pour cela on établit d'abord le lemme suivant.

Lemme 4.1. *[12] Le $O2, S1|s_{ij} = s, p_{ij} = p|C_{\max}$ a la borne inférieure LB, sur le makespan de n'importe quel ordonnancement S :*

$$C_{\max}(S) \geq LB = \begin{cases} 2ns+p & si\ s > p, \\ s+n(s+p) & sinon. \end{cases}$$

Démonstration. Si $s > p$ alors évidement on a

$$LB_2 = \sum_{i,j} s_{ij} + \min\{\min_{1\leq j\leq n}\{p_{1j}\}, \min_{1\leq j\leq n}\{p_{2j}\}\} = 2ns + p.$$

À présent, on considère le cas $p \geq s$. Puisqu'on a supposé que $s \neq 0$, alors une période d'arrêt de s unités de temps est inévitable sur une des deux machines. Ainsi le temps de fin de traitement total est égal à la borne inférieure LB_3

$$LB_3 = \min\left\{\max\{\sum_{j=1}^{n}(s_{1j}+p_{1j})+s, \sum_{j=1}^{n}(s_{2j}+p_{2j})\}, \max\{\sum_{j=1}^{n}(s_{1j}+p_{1j}), \sum_{j=1}^{n}(s_{2j}+p_{2j})+s\}\right\}$$

□

On présente maintenant le résultat suivant.

Théorème 4.1. *$O2, S1|s_{ij} = s, p_{ij} = p|C_{\max}$ est optimalement résoluble en $O(1)$ [9, 12].*

Démonstration. Si $s > p$, le traitement peut commencer sur n'importe quelle des deux machines. La séquence sur M_1 est $T_1, \ldots, T_n$. Sur M_2, elle sera $T_n, T_1, \ldots, T_{n-1}$, ce qui permit d'éviter que deux opérations de la même tâche ne soient ordonnancées simultanément. Dans ce cas, le serveur sera occupé durant l'intervalle $[0, 2n.s]$. Le traitement de la dernière opération commence à l'instant $t = 2ns$, et dure p unités de temps. Le temps de fin de traitement total sera ainsi $C_{\max}(S) = 2ns + p$.

Supposons maintenant que $s \leq p$. Dans ce cas aussi, le traitement peut commencer sur n'importe quelle des deux machines. On garde les mêmes séquences de tâches sur les deux machines comme dans le cas précèdent et ce pour les mêmes raisons. Si on suppose qu'on commence le traitement sur M_1, alors il n'y aura aucun temps mort sur cette machine durant $[0, n(s+p)]$. Le traitement sur M_2 commence à l'instant $t = s$ et la machine sera occupée durant $[s, n(s+p)+s]$.

Le makespan dans les deux cas est égal à la borne inférieure donnée dans le Lemme 4.1, ainsi il est optimal. Le premier et le second cas sont représentés dans la Figure 4.1.1a et Figure 4.1.1b, respectivement. En ce qui concerne la complexité de l'algorithme, il est clair qu'elle est en $O(1)$.

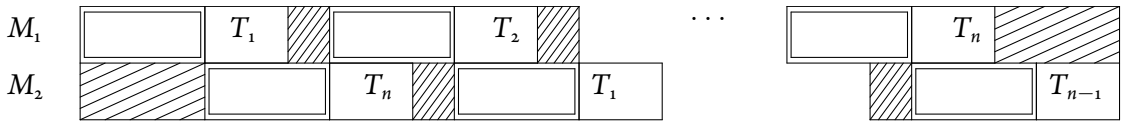


Figure 4.1.1a – Ordonnancement optimal du cas $s > p$

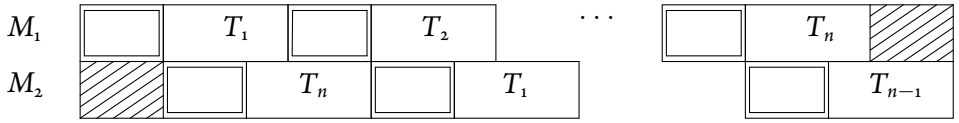


Figure 4.1.1b – Ordonnancement optimal du cas $p \geq s$

□

4.1.2 Durées d'exécution identiques sur chaque machine

On présente maintenant le second cas polynomial, qui est le cas où les durées d'exécution sont identiques sur chacune des deux machines. Le problème est noté $O2, S1|p_{1i} = s_{1i} = a, p_{2i} = s_{2i} = b|C_{\max}$.

Théorème 4.2. *$O2, S1|p_{1i} = s_{1i} = a, p_{2i} = s_{2i} = b|C_{\max}$ est résoluble en $O(1)$, avec un makespan $C_{\max}(S^*) = 2n\max\{a,b\} + \max\{2\min\{a,b\} - \max\{a,b\}, 0\}$ [13].*

Démonstration. On suppose que $a \geq b$; l'autre cas étant symétrique. Il s'en suit que M_1 est la machine ayant le plus long temps de traitement total. Si le traitement commence sur M_1 alors la fin de traitement sera sur M_2. Le seul temps d'arrêt de M_1 sur toute la longueur d'ordonnancement se produit après la fin de traitement sur cette machine.

Si le traitement commence sur M_2, alors la fin de traitement sera sur M_1. La seule période d'arrêt sur M_1 est au début, et qui est de b unités de temps.

Dans les deux cas, M_1 est uniquement en arrêt durant une seule période. Par conséquent, la solution optimale correspond au cas ayant le plus cours temps d'arrêt sur M_1. Le temps d'arrêt est de $\max\{(2b-a), 0\}$ unités de temps dans le premier cas, et il est de b unités de temps dans le second cas. Puisque $a \geq b$, alors il s'en suit que $\max\{(2b-a), 0\} \leq b$.

La solution optimale est obtenue en commençant le traitement sur M_1. Le makespan est ainsi égal à $2na + \max\{(2b-a), 0\}$. Ce qui veut dire qu'il faudrait commencer le traitement sur la machine ayant le plus long temps de traitement total afin d'avoir la solution optimale. L'ordonnancement doit aussi vérifier que deux opérations de la même tâche ne soient pas traitées en même temps. Pour cela, la séquence de traitement sur M_1 est $T_1, \ldots, T_n$ tandis que sur M_2 elle est : $T_n, T_1, \ldots, T_{n-1}$. L'ordonnancement illustré dans la Figure 4.1.2 est la solution optimale du cas $a > b$.

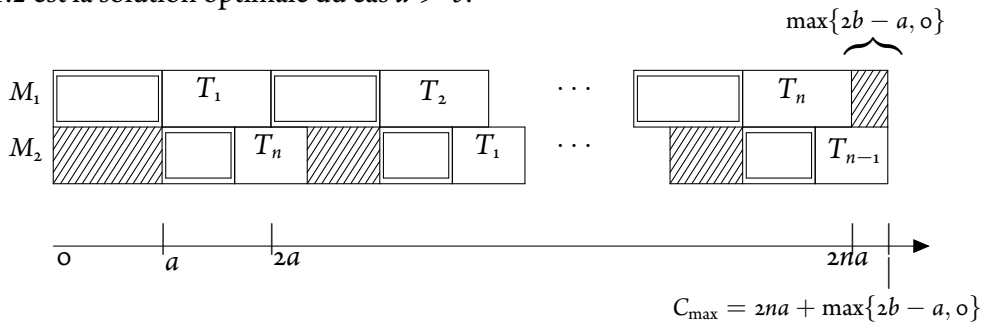


Figure 4.1.2 – La structure de l'ordonnancement (Le cas $a > b$)

□

4.2 Cas particuliers du job shop

Dans ce qui suit, nous présentons deux cas résolubles en temps polynomial. Le premier est celui où nous avons des temps de préparation identiques et des temps de traitement identiques. Le problème est noté $J2, S1|s_{ij} = s, p_{ij} = p|C_{max}$. Le second est le cas où nous avons des temps de traitement identiques et de longs temps préparation, le problème est ainsi noté $J2, S1|p_{ij} = p, s_{ij} \geq p|C_{max}$.

4.2.1 Temps de préparation et de traitement identiques

On commence avec le cas où nous avons des temps de préparation identiques et des temps de traitement identiques *c. à d.* $J2, S1|s_{ij} = s, p_{ij} = p|C_{\max}$.

Théorème 4.3. *L'Algorithme 4 résout le* $J2, S1|s_{ij} = s, p_{ij} = p|C_{max}$ *en* $O(n)$ *[14].*

Algorithme 4 :

début
- soient Q_1, Q_2, Q_3 et Q_4 les séquences arbitraires des tâches de N_1, N_2, N_3 et N_4, respectivement ;
- définir $n_{13} = n_1 + n_3$ et $n_{24} = n_2 + n_4$;
- **si** $n_{24} = 1, n_1 \geq 1, n_3 \leq n_4$ **alors**
  - définir M_1 comme étant la machine prioritaire ;
- **sinon si** $n_{13} = 1, n_2 \geq 1, n_4 \leq n_3$ **alors**
  - définir M_2 comme étant la machine prioritaire ;
- **sinon**
  - **si** $(n_3 > n_4)$ *ou* $((n_3 = n_4)$ *et* $(n_1 \geq n_2))$ **alors**
    - définir M_1 comme étant la machine prioritaire ;
  - **sinon**
    - définir M_2 comme étant la machine prioritaire ;
- définir Q_1, Q_3, Q_2 et Q_2, Q_4, Q_1 comme la séquence des tâches sur M_1 et M_2, respectivement ;
- **tan que** il reste des tâches non-ordonnancées **faire**
  - **si** les deux machines sont libres et les premières tâches de chaque séquence sont disponibles **alors**
    - ordonnancer la tâche sur la machine prioritaire ;
  - **sinon**
    - ordonnancer la tâche sur la première machine libre ;

Démonstration. On considère d'abord l'ordre de traitement des tâches sur les machines, les séquences sont les suivantes :
Sur M_1 : Q_1, Q_3, Q_2.
Sur M_2 : Q_2, Q_4, Q_1.
Celles-ci correspondent à l'ordre donné par la règle de Jackson présentée dans l'Algorithme 2. Cet ordre des tâches garantit en partie qu'il n'y a pas de temps d'arrêt dû à l'indisponibilité des tâches.

Le second point important de l'algorithme est la définition de la machine prioritaire. Lorsqu'une machine est définie comme prioritaire, si à un instant t les deux machines sont libres et il y a au moins une tâche disponible sur chaque machine, alors nous choisissons d'ordonnancer d'abord la tâche sur la machine prioritaire. Si à un instant t les deux machines sont libres et qu'une seule d'entre elles a des tâches disponibles, alors il est évident qu'il n'y a pas de choix à faire.

Le choix de la machine prioritaire dépend principalement du nombre d'opérations sur chaque machine. Ainsi, dans le cas général, la machine prioritaire est celle avec le plus grand nombre d'opérations. Cependant, il existe deux cas particuliers qui ne suivent pas la règle générale lors de la définition de la machine

prioritaire. Ces cas sont discutés dans le Cas 2.1 et pris en charge par les deux premières conditions de l'Algorithme 4.

Nous définissons $n_{13} = n_1 + n_3$ et $n_{24} = n_2 + n_4$, et on note également que les tâches α, β, a et b désignent les tâches de N_1, N_2, N_3 et N_4, respectivement. Les différents cas sont construits selon la valeur de n_{24}. Les cas qui énumèrent les valeurs de n_{13} sont des cas symétriques. Par ailleurs, on précise que les sous-cas sont construits par rapport à la forme de l'ordonnancement et la formule du makespan de la solution optimale.

Cas 1 : $n_{24} = 0$. Les différents sous-cas sont les suivants.

Cas 1.1 : $n_1 \geq 1, n_3 \leq 1$. Nous considérons d'abord les cas particuliers, ensuite le reste des cas.

Cas 1.1.1 : $n_1 = 0, n_3 = 1$. Nous n'avons qu'une seule tâche ayant une seule opération. Par conséquent, le makespan est égal à $C_{\max}(S) = \sum_{i=1}^{2}(s_{i1} + p_{i1}) = s + p$.

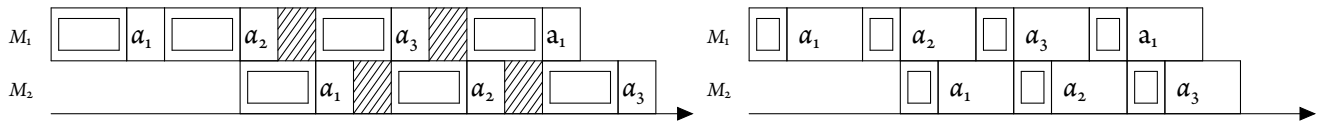
Cas 1.1.2 : $n_1 = 1, n_3 = 0$. Dans ce cas, nous n'avons qu'une seule tâche et celle-ci a deux opérations. Ainsi, le makespan est égal à $C_{\max}(S) = \sum_{i=1}^{2}(s_{i1} + p_{i1}) = 2s + 2p$.

Cas 1.1.3 : $n_1 = 1, n_3 = 1$. Nous n'avons trois opérations, dont deux sur M_1. Le makespan est égal à $C_{\max}(S) = 2s + p + \sum_{j=1}^{n}(s_{2j} + p_{2j}) = 3s + 2p$.

Cas 1.1.4 : $n_1 > 1, n_3 \leq 1$.

— Si $n_3 = 1$ alors la machine prioritaire est M_1 puisque $n_3 > n_4$. La décision de la prioriser intervient à l'instant $t = s + p$, où a_2 sur M_1 est ordonnancée avant a_1 sur M_2. Si ce n'est pas le cas, et avec M_2 comme machine prioritaire, M_1 sera inactive pendant s unités de temps. De plus, M_2 doit attendre la fin de chaque tâche sur M_1 pour démarrer sa deuxième opération, ce qui produit un ordonnancement plus long.

— Pour le cas $n_3 = 0$, nous avons $n_3 = n_4$ et $n_1 > n_2$, donc, M_1 est la machine prioritaire. De même qu'avec $n_3 = 1$, si nous priorisons M_2 à l'instant $t = s + p$ alors le makespan obtenu est supérieur à celui obtenu lorsque nous priorisons M_1.



(a) Pour $s > p$ **(b)** Pour $s \leq p$

Figure 4.2.1 – Cas : $n_{24} = 0,\ n_1 \geq 1,\ n_3 \leq 1$

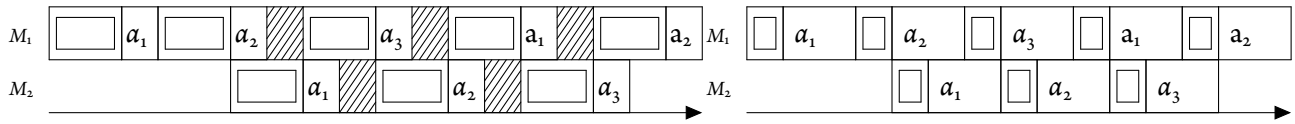
L'ordonnancement est illustré dans la Figure 4.2.1a pour le cas $s > p$ et dans la Figure 4.2.1b pour $s \leq p$ et le makespan est égal à :

$$C_{\max}(S) = 2s + p + \sum_{j=1}^{n}(s_{2j} + p_{2j}) + \sum_{j=3}^{n_{13}} \max\{0, s - p\}.$$

Cas 1.2 : $n_1 \geq 1, n_3 \geq 2$. La machine prioritaire est M_1 puisque $n_3 > n_4$. Il est évident que le makespan égal au temps de fin de traitement sur M_1. Plus précisément, il est égal au temps de traitement total sur

M_1 auquel on ajoute le temps mort possible sur cette machine causé par la préparation des tâches sur M_2. L'ordonnancement optimal est illustré dans les Figures 4.2.2a et 4.2.2b, avec un makespan égal à

$$C_{\max}(S) = \sum_{j=1}^{n}(s_{1j} + p_{1j}) + \sum_{j=1}^{n_1} \max\{0, s - p\}.$$



(a) Pour $s > p$ **(b)** Pour $s \leq p$

Figure 4.2.2 – Cas : $n_{24} = 0,\ n_1 \geq 1,\ n_3 \geq 2$

Cas 2 : $n_{24} = 1$, ce cas est divisé en plusieurs sous-cas qui sont détaillés dans ce qui suit.
Cas 2.1 : $n_1 \geq 1, n_3 \leq n_4$. Ce cas et le cas $n_2 \geq 1, n_{13} = 1, n_4 \leq n_3$ sont symétriques. Par conséquent, nous ne considérons qu'un seul des deux. Ces cas sont pris en charge dans l'Algorithme 4 par les deux premières conditions *"si"* et *"sinon si"*. Nous avons $n_{24} = 1$, par conséquent, nous obtenons les possibilités suivantes.

Cas 2.1.1 : Si $n_2 = 1$ alors $n_4 = 0$ et $n_3 = 0$. Même si nous avons le même nombre d'opérations sur les deux machines, si nous définissons M_2 comme machine prioritaire, la tâche β_1 sur M_2 commencera avant a_1 sur M_1. Ainsi, un temps d'inactivité se produit sur M_2 qui correspond au temps d'attente causé par chaque tâche de a. Nous concluons que la machine prioritaire doit être M_1. L'ordonnancement est illustré dans les Figures 4.2.3a et 4.2.3b ; le makespan est égal à

$$C_{\max}(S) = s + \sum_{j=1}^{n}(s_{2j} + p_{2j}) + \sum_{j=2}^{n_1+n_2} \max\{0, s - p\}.$$

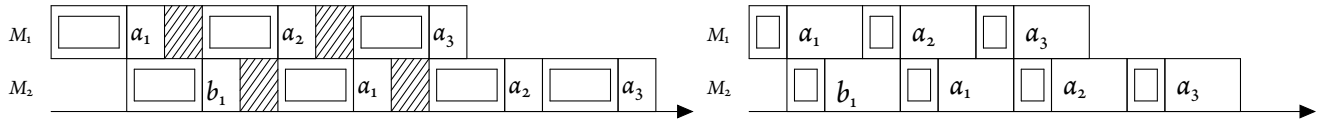
(a) Pour $s > p$ **(b)** Pour $s \leq p$

Figure 4.2.3 – Cas : $n_1 \geq 1,\ n_2 = 1,\ n_3 = n_4 = 0$

Cas 2.1.2 : Si $n_4 = 1$ alors soit $n_3 = 0$, soit $n_3 = 1$. Ce cas est similaire au précédent. La machine prioritaire est M_1 et ceci est dû aux mêmes raisons. L'ordonnancement est illustré dans les

Figures 4.2.4a et 4.2.4b. Le makespan est égal à

$$C_{\max}(S) = s + \sum_{j=1}^{n}(s_{2j} + p_{2j}) + \sum_{j=2}^{n_{13}} \max\{0, s - p\}.$$



(a) Pour $s > p$ **(b)** Pour $s \leq p$

Figure 4.2.4 – Cas : $n_1 \geq 1$, $n_2 = 0$, $n_3 = 0$, $n_4 = 1$

Cas 2.2 : $n_1 \geq 1, n_3 > n_4$. Cela englobe le cas $n_1 \geq 1, n_2 = 0, n_4 = 1, n_3 \geq 2$ et le cas $n_1 \geq 1, n_2 = 1, n_4 = 0, n_3 \geq 1$. La machine prioritaire dans ces cas est M_1 puisque $n_3 > n_4$, et le makespan de la solution optimale est égal à :

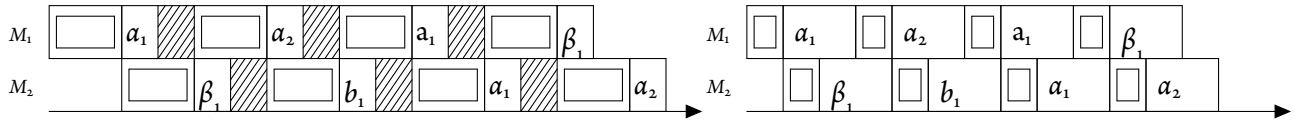
$$C_{\max}(S) = \sum_{j=1}^{n}(s_{1j} + p_{1j}) + \sum_{j=1}^{n_1+n_{24}} \max\{0, s - p\}.$$

Cas 3 : $n_{24} \geq 2$. Les sous-cas sont les suivants.

- $n_{13} = 0$ est le symétrique du Cas 1 avec $n_{24} = 0$.
- $n_{13} = 1$ est le symétrique du Cas 2 avec $n_{24} = 1$.
- $n_{13} \geq 2$ est le cas que nous examinerons ci-dessous ; et les sous-cas possibles sont les suivants.

Cas 3.1 : $n_{13} \geq 2, n_3 = n_4$: Nous pouvons définir soit M_1 soit M_2 comme machine prioritaire. L'ordonnancement est illustré dans la Figure 4.2.5a et la Figure 4.2.5b, avec un makespan égal à :

$$C_{\max}(S) = s + \sum_{j=1}^{n}(s_{2j} + p_{2j}) + \sum_{j=2}^{n_{13}+n_2} \max\{s - p, 0\}.$$



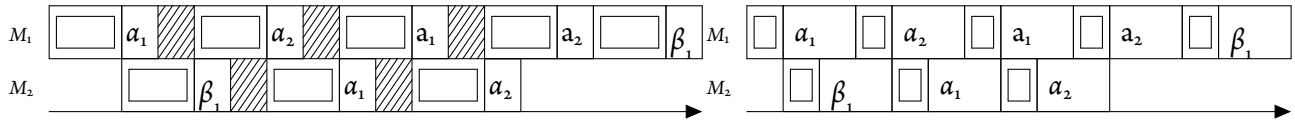
(a) Pour $s > p$ **(b)** Pour $s \leq p$

Figure 4.2.5 – Cas : $n_{13} \geq 2$, $n_3 = n_4$

Cas 3.2 : $n_{13} \geq 2, n_3 > n_4$: Dans ce cas, le temps de traitement total sur M_1 est plus grand que sur M_2. Par conséquent, la machine prioritaire est M_1. L'ordonnancement est illustré dans les Figures 4.2.6a et 4.2.6b,

de plus, le makespan est égal à :

$$C_{\max}(S) = \sum_{j=1}^{n}(s_{1j} + p_{1j}) + \sum_{j=1}^{n_{24}+n_1} \max\{s - p, 0\}.$$



(a) Pour $s > p$ **(b)** Pour $s \leq p$

Figure 4.2.6 – Cas : $n_{13} \geq 2,\ n_3 > n_4$

Nous avons montré en haut que l'ordre des tâches et le choix de la machine prioritaire de l'Algorithne 4 produisent l'ordonnancement optimal pour le problème $J2, S1|s_{ij} = s, p_{ij} = p|C_{\max}$. En ce qui concerne la complexité de l'algorithme, une analyse simple montre qu'il est en $O(n)$. □

4.2.2 Temps de traitement identiques et temps de préparation longs

On termine cette section avec le second cas particulier du job shop résoluble en temps polynomial et qui est le problème $J2, S1|p_{ij} = p, s_{ij} \geq p|C_{\max}$.

Théorème 4.4. *L'Algorithme 4 résout $J2, S1|p_{ij} = p, s_{ij} \geq p|C_{max}$ en $O(n)$ [14].*

Démonstration. Ce problème est similaire au $J2, S1|s_{ij} = s, p_{ij} = p|C_{max}$ avec $s \geq p$. Par conséquent, la preuve est similaire.

Nous définissons $n_{13} = n_1 + n_3$, $n_{24} = n_2 + n_4$, tandis que les tâches α, β, a et b désigneront les tâches de N_1, N_2, N_3 et N_4, respectivement. Les différents cas sont construits selon la valeur de n_{24}. De la même façon que dans la preuve précédente, les sous-cas sont construits par rapport à la forme de l'ordonnancement et la formule du makespan de la solution optimale.

Cas 1 : $n_{24} = 0$. Les sous-cas sont les suivants.

Cas 1.1 : $n_1 \geq 1, n_3 \leq 1$. Ce cas a les possibilités suivantes.

Cas 1.1.1 : $n_1 = 0, n_3 = 1$. Nous n'avons qu'une seule tâche ayant une seule opération. Donc, le makespan est égal à $C_{\max}(S) = \sum_{i=1}^{2}(s_{i1} + p_{i1}) = s_{11} + p$.

Cas 1.1.2 : $n_1 = 1, n_3 = 0$. Dans ce cas, nous n'avons qu'une seule tâche et celle-ci a deux opérations. Ainsi, le makespan est égal à $C_{\max}(S) = \sum_{i=1}^{2}(s_{i1} + p_{i1}) = s_{11} + s_{21} + 2p$.

Cas 1.1.3 : $n_1 = 1, n_3 = 1$. Nous n'avons trois opérations, dont deux sur M_1. Le makespan est égal à $C_{\max}(S) = s_{11} + s_{12} + p + \sum_{j=1}^{n}(s_{2j} + p_{2j}) = s_{11} + s_{12} + s_{21} + 2p$.

Cas 1.1.4 : $n_1 > 1, n_3 = 0$. La machine prioritaire est M_1 vue que $n_3 = n_4$ et $n_1 > n_2$. Par conséquent, à l'instant $t = s_{1a_1} + p$ nous ordonnançons d'abord a_2 sur M_1 avant a_1 sur M_2 sinon un temps d'attente sur M_2 se produira après la fin de chaque tâche de a sur M_1. L'ordonnancement est représenté dans la Figure 4.2.7 et le makespan est égal à : $C_{\max}(S) = \sum_{j=1}^{n}\sum_{i=1}^{2} s_{ij} + 3p$.

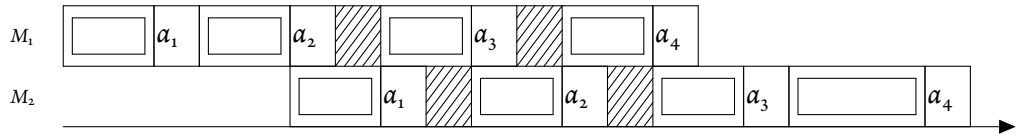


Figure 4.2.7 – Cas : $n_{24} = 0$, $n_1 > 1$, $n_3 = 0$

Cas 1.1.5 : $n_1 > 1, n_3 = 1$. La machine prioritaire est M_1 puisque $n_3 > n_4$, et la décision de la prioriser survient à l'instant $t = s_{1a_1} + p$. Si nous ordonnançons a_1 sur M_2 avant a_2 sur M_1 (dans le cas où M_2 est la machine prioritaire), alors M_1 sera inactif pendant s unités de temps. De plus, un temps d'attente se produira sur M_2 après la fin de chaque tâche sur M_1. Ainsi la machine prioritaire doit être M_1 et le makespan est égal à : $C_{\max}(S) = \sum_{j=1}^{n}\sum_{i=1}^{2} s_{ij} + 2p$.

Cas 1.2 : $n_1 \geq 1, n_3 \geq 2$. La machine prioritaire est M_1 puisque $n_3 > n_4$. Il est évident que le makespan égal au temps de fin de traitement est sur M_1. Plus précisément, il est égal au temps de traitement total sur M_1 auquel on ajoute le temps mort sur cette machine causé par la préparation des tâches sur M_2. L'ordonnancement optimal est similaire à l'ordonnancement illustré dans la Figure 4.2.2a. Le makespan est égal à :

$$C_{\max}(S) = \sum_{j=1}^{n}(s_{1j} + p_{1j}) + \sum_{j=1}^{n_1}(s_{2j} - p).$$

Cas 2 : $n_{24} = 1$, les différents sous-cas sont donnés dans ce qui suit.

Cas 2.1 : $n_1 \geq 1, n_3 \leq n_4$. Ce cas est symétrique avec $n_{13} = 1, n_2 \geq 1, n_4 \leq n_3$. Par conséquent, nous ne considérons qu'un seul des cas. Ces cas sont traités par l'Algorithme 4 dans les deux premières conditions *"si"*. Puisque $n_{24} = 1$ alors nous obtenons les possibilités suivantes.

Cas 2.1.1 : Si $n_2 = 1$ alors $n_4 = 0$ et $n_3 = 0$. Nous avons le même nombre d'opérations sur les deux machines. Ainsi, si nous définissons M_2 comme machine prioritaire, la tâche β_1 sur M_2 commencera avant a_1 sur M_1. Dans ce cas, M_2 sera inactive en attendant la fin de chaque tâche de a sur M_1. Nous concluons que la machine prioritaire doit être M_1. L'ordonnancement optimal est similair à celui représenté par la Figure 4.2.3a et le makespan est égal à :

$$C_{\max}(S) = \sum_{i=1}^{2}\sum_{j=1}^{n} s_{ij} + p.$$

Cas 2.1.2 : $n_4 = 1, n_3 = 1$ et $n_2 = 0$. Ce cas est similaire au cas précédent, nous avons le même nombre d'opérations sur les deux machines. Cependant, la machine prioritaire doit être M_1 pour la même raison mentionnée ci-dessus. Le makespan est également le même.

Cas 2.1.3 : $n_4 = 1, n_3 = 0$ et $n_2 = 0$. Ce cas est également similaire aux deux précédents. La machine prioritaire est M_1 et ceci est dû aux mêmes raisons. Cependant, l'ordonnancement est similaire à celui illustré dans la Figure 4.2.4a et le makesan est égal à :

$$C_{\max}(S) = \sum_{i=1}^{2} \sum_{j=1}^{n} s_{ij} + 2p.$$

Cas 2.2 : $n_1 \geq 1, n_3 > n_4$. Ceci englobe le cas $n_1 \geq 1, n_2 = 0, n_4 = 1, n_3 \geq 2$ et le cas $n_1 \geq 1, n_2 = 1, n_4 = 0, n_3 \geq 1$: La machine prioritaire, dans ces cas, est M_1 puisque $n_3 > n_4$, et le makespan de la solution optimale est

$$C_{\max}(S) = \sum_{j=1}^{n} (s_{1j} + p_{1j}) + \sum_{j=1}^{n_1+n_{24}} (s_{2j} - p).$$

Cas 3 : $n_{24} \geq 2$. Les sous-cas sont donnés dans ce qui suit.

- $n_{13} = 0$ est le symétrique du Cas 1 avec $n_{24} = 0$.
- $n_{13} = 1$ est le symétrique du Cas 2 avec $n_{24} = 1$.
- $n_{13} \geq 2$ est le cas que nous considérons ci-dessous, dont les sous-cas sont les suivants.

Cas 3.1 : $n_{13} \geq 2, n_3 = n_4$: Nous pouvons définir soit M_1 soit M_2 comme machine prioritaire. L'ordonnancement est similaire à celui de la Figure 4.2.5a et le makespan est égal à :

$$C_{\max}(S) = \sum_{j=1}^{n} \sum_{i=1}^{2} s_{ij} + p.$$

Cas 3.2 : $n_{13} \geq 2, n_3 > n_4$: Dans ce cas, le temps de traitement total sur M_1 est plus grand que sur M_2. Par conséquent, la machine prioritaire est M_1. L'ordonnancement est similaire à celui illustré dans la Figure 4.2.6a. De plus, le makespan est égal à :

$$C_{\max}(S) = \sum_{j=1}^{n} (s_{1j} + p_{1j}) + \sum_{j=1}^{n_{24}+n_1} (s_{2j} - p).$$

Nous avons montre que l'Algorithme 4 résout optimalement le problème $J2, S1|p_{ij} = p, s_{ij} \geq p|C_{\max}$. Nous rappelons aussi que la complexité de l'algorithme est en $O(n)$.

□

Conclusion

Nous avons présenté, dans ce chapitre, les cas particuliers résolubles en temps polynomial des deux problèmes étudiés, *c.-à-d.* open shop et job shop à deux machines et un serveur. Dans le Tableau 4.2.1 sont résumés chacun des cas ainsi que la complexité de l'algorithme qui le résout.

Sous-problème	Complexité
$O2, S1 \mid s_{ij} = s, p_{ij} = p \mid C_{\max}$	$O(1)$ [9, 12]
$O2, S1 \mid s_{1j} = s_{2j} = a, p_{1j} = p_{2j} = b \mid C_{\max}$	$O(1)$ [13]
$J2, S1 \mid s_{ij} = s, p_{ij} = p \mid C_{\max}$	$O(n)$ [11, 14]
$J2, S1 \mid p_{ij} = p, s_{ij} \geq p \mid C_{\max}$	$O(n)$ [14]

Tableau 4.2.1 – Les cas résolubles en temps polynomial

5

Résolution du problème de l'open shop

Ce chapitre est consacré aux méthodes de résolution que nous avons élaborées afin de résoudre le problème $O2, S1|s_{ij}|C_{\max}$. D'abord, nous présenterons les méthodes approchées, qui consistent en sept heuristiques et une hyper-heuristique. Ensuite, on passe aux méthodes exactes, en présentant deux formulations du problème en programme mathématique linéaire en nombres entiers mixtes.

5.1 Méthodes approchées

Nous avons montré que le problème est $\mathcal{NP}$-difficile au sens fort, en prouvant que plusieurs cas particuliers du problème sont $\mathcal{NP}$-difficiles. De ce fait, l'approche heuristique est justifiée. Nous décrirons dans ce qui suit les heuristiques ainsi que l'hyper-heuristique qui combine les deux meilleures heuristiques.

5.1.1 Heuristiques

Nous décrirons, d'abord, les sept heuristiques que nous avons développées pour la résolution du problème. Ensuite, nous présentons l'étude expérimentale que nous avons faite pour tester leurs performances.

Les trois premières heuristiques sont résumées dans les heuristiques $H_k, k = 1, 2, 3$. Les deux premières dépendent de la procédure de tri. Un bon choix dans la méthode de tri fera la différence dans la performance de l'heuristique. On choisit de trier les tâches sur une machine par rapport aux temps de préparation, tandis que sur la deuxième machine l'ordre est par rapport aux temps de traitement. Le but est d'éviter d'avoir un temps mort dû à l'indisponibilité du serveur. Avec un tel ordre, une tâche ayant un long temps de préparation est ordonnancée en parallèle d'une tâche avec un long temps de traitement et *vice versa*. Cette méthode est testée via deux procédures. La Procédure $Proc_1$ trie les tâches dans un ordre croissant, alors que, la procédure $Proc_2$ trie les tâches dans un ordre décroissant.

Les trois heuristiques ordonnancent à chaque fois la première tâche de la liste J_i sur M_i ; $i = 1, 2$. Si cette tâche est en cours de traitement sur l'autre machine alors elles ordonnancent la seconde tâche de la liste. Cependant, l'heuristique H_3 construit une solution arbitraire, puisque les tâches ne sont pas triées dans les deux listes. Le traitement dans cette dernière commence sur la machine ayant le plus long temps de traitement total.

Algorithme 5 : Procédure $Proc_1$

début

trier les tâches dans J_1 dans l'ordre croissant des temps de traitement sur M_1 ;
trier les tâches dans J_2 dans l'ordre croissant des temps de préparation sur M_2 ;

Algorithme 6 : Procédure $Proc_2$

début

trier les tâches dans J_1 dans l'ordre décroissant des temps de traitement sur M_1 ;
trier les tâches dans J_2 dans l'ordre décroissant des temps de préparation sur M_2 ;

Algorithme 7 : Procédure $Proc_3$

début

soit $J_1 = (T_1, \ldots, T_n)$ la séquence des tâches de M_1 ;
soit $J_2 = (T_1, \ldots, T_n)$ la séquence des tâches de M_2 ;

La $4^{éme}$ et $5^{éme}$ heuristiques sont résumées dans les heuristiques $H_k, k = 4, 5$, et dépendent de la procédure du choix de la prochaine tâche à ordonnancer. La Procédure $Proc_4$ et la Procédure $Proc_5$ choisissent les

Algorithme 8 : Heuristique H_k, $k = 1, 2, 3$

début
soient J_1, J_2 les séquences des tâches sur M_1, M_2, respectivement, générées par la procédure $Proc_k$;
tan que $J_1 \neq \emptyset$ *et* $J_2 \neq \emptyset$ **faire**
soit M_i, $i = 1, 2$, la première machine libre, T_j la dernière tâche ordonnancée sur $M_{i'}$; $i' = 3 - i$;
si *T_j, la première tâche de la séquence J_i, est en traitement sur $M_{i'}$* **alors**
sélectionner la seconde tâche de J_i ;
sinon
sélectionner la première tâche de J_i ;
ordonnancer la tâche sélectionnée sur M_i et la supprimer de J_i ;
tan que $J_1 \neq \emptyset$ $(J_2 \neq \emptyset)$ **faire**
ordonnancer les tâches restantes selon l'ordre de J_1 (J_2), tout en les supprimant de la liste ;

tâches selon la règle GARD (voir Algorithme 3). La première considère seulement les temps de traitement, tandis que, la seconde considère les durées d'exécution. Ceci nous permet de voir si la stratégie qui résout le problème classique reste efficace pour la résolution de cette variante.

Algorithme 9 : Procédure $Proc_4$

début
si $|T| = n$ **alors**
sélectionner T_j telle que $p_{i'j} = \max_h\{p_{i'h} \,|T_h \in T, i' = 3 - i\}$;
sinon
sélectionner arbitrairement une tâche T_j de T ;
$T = T - \{T_j\}$;

Algorithme 10 : Procédure $Proc_5$

début
si $|T| = n$ **alors**
sélectionner T_j telle que $(p_{i'j} + s_{i'j}) = \max_h\{(s_{i'h} + p_{i'h}) \mid T_h \in T, i' = 3 - i\}$;
sinon
sélectionner arbitrairement une tâche T_j de T ;
$T = T - \{T_j\}$;

Dans l'heuristique H_6, l'accent est mis sur la réduction du temps d'arrêt du serveur, tout en réduisant le temps mort sur les machines. Lorsque les temps de préparation sont plus longs que les temps de traitement, la priorité serait de réduire le temps d'arrêt du serveur. Dans la solution construite par H_6, les préparations sont ordonnancées à des intervalles proches les uns des autres et la dernière tâche traitée sera celle avec le

Algorithme 11 : Heuristique H_k, $k = 4, 5$

début
- le traitement commence sur M_i avec $\sum_j (s_{ij} + p_{ij}) = \max_{\ell=1,2}\{\sum_j (s_{\ell j} + p_{\ell j})\}$;
- soit T l'ensemble de tâche à traiter et $J_1 = \emptyset, J_2 = \emptyset$;
- choisir la prochaine tâche à traiter sur M_i utilisant la Procédure $Proc_k$;
- ordonnancer la tâche sélectionnée et l'ajouter à la liste $J_{i'}$, $i = 1, 2$; $i' = 3 - i$;
- **tan que** $T \neq \emptyset$ **faire**
  - soit M_i la première machine libre ; sélectionner la prochaine tâche sur M_i en utilisant la Procédure $Proc_k$;
  - ordonnancer la tâche sélectionnée sur M_i, et ajouter la à $J_{i'}$, $i = 1, 2$; $i' = 3 - i$;
- **tan que** $J_1 \neq \emptyset$ *et* $J_2 \neq \emptyset$ **faire**
  - soit M_i, $i = 1, 2$, la première machine libre ; sélectionner la première tâche de J_i ;
  - ordonnancer la tâche sélectionnée sur M_i et supprimer la de J_i ;
- **tan que** $J_1 \neq \emptyset$ $(J_2 \neq \emptyset)$ **faire**
  - ordonnancer les tâches sur M_1 (M_2) selon l'ordre de J_1 (J_2), tout en les supprimant de la liste ;

Algorithme 12 : Heuristique H_6

début
- soit $J = \emptyset$;
- sélectionner la tâche T_j sur M_i tel que $s_{ij} = \min_{1 \leq \ell \leq n}\{s_{1\ell}, s_{2\ell}\}$ et ajouter T_j à J ;
- **tan que** $|J| < 2n$ **faire**
  - **si** *la dernière tâche sélectionnée est T_j sur M_i* **alors**
    - sélectionner la tâche $T_{j'} \notin J$ telle que $s_{ij} - p_{i'j'} = \min_{k \neq j}\{s_{ij} - p_{i'k}$ t.q. $s_{ij} \geq p_{i'k}\}$;
    - **si** *aucune tâche n'est sélectionnée* **alors**
      - sélectionner $T_{j'} \notin J$ tel que $p_{i'j'} = \min_{k \neq j}\{p_{i'k}\}$;
  - **si** *une tâche est sélectionnée* **alors**
    - ajouter cette tâche à J ;
  - **sinon**
    - (la tâche qui reste sur $M_{i'}$ est T_j) ;
    - permuter la dernière tâche avec l'avant dernière, puis ajouter T_j sur $M_{i'}$ à J ;
- ordonnancer les opérations selon l'ordre inverse donné par J.

plus court temps de traitement. Pour faire cela, nous allons construire dans un premier temps la séquence inverse de la séquence de traitement des tâches. On choisit d'abord la dernière tâche qui est celle avec le plus court temps de traitement. Par la suite, on choisit la tâche qui la précède et qui est sur l'autre machine. Le choix de cette tâche dépend de celle qui la succède dans l'ordonnancement. On choisit la tâche ayant un temps de traitement égal au temps de préparation de la dernière tâche sélectionnée. Si une telle tâche n'existe pas alors on choisit une tâche avec un temps de traitement inférieur au temps de préparation, mais qui soit le plus proche. Si aussi une telle tâche n'existe pas alors on sélectionne la tâche ayant le plus court

temps de traitement.

Dans l'heuristique H_7, on cherche à éviter le temps mort dû à l'indisponibilité du serveur. Nous définissons ℓ comme étant la longueur de l'intervalle de temps où seulement une des deux machines et le serveur sont libres. Ensuite, nous ordonnançons dans cet intervalle la tâche ayant un temps de préparation égal à ℓ. Si une telle tâche n'existe pas alors on cherche une tâche ayant un temps de préparation inférieur à ℓ, mais qui soit le plus proche. Sinon on ordonnance la tâche ayant le plus court temps de préparation.

Algorithme 13 : Heuristique H_7

début
- soient J_1 et J_2 l'ensemble des tâches sur M_1 et M_2, respectivement ;
- commencer sur M_i avec $\sum_j (s_{ij} + p_{ij}) = \max_{k=1,2}\{\sum_j (s_{kj} + p_{kj})\}$ le traitement de la tâche T_j tel que $s_{ij} = \min_k\{s_{ik}\}$;
- soient τ_1, τ_2 et τ les instants où M_1, M_2 et $S1$, deviennent libres, respectivement ;
- **tan que** $J_1 \neq \emptyset$ *et* $J_2 \neq \emptyset$ **faire**
  - **si** $M_i, i = 1, 2$, *est la première machine libre* **alors**
    - soit $\ell = \tau_{i'} - \max(\tau_i, \tau), i' = 3 - i$;
    - sélectionner une tâche de J_i avec $s_{ij} = \min_{T_k \in J_i}\{\ell - s_{ik}$ tel que $s_{ik} \leq \ell\}$;
    - **si** $\{T_k \in J_i$ *t.q.* $s_{ik} \leq \ell\} = \emptyset$ **alors**
      - sélectionner une tâche T_j tel que $s_{ij} = \min_{T_k \in J_i}\{s_{ik}\}$;
  - ordonnancer la tâche sélectionnée sur M_i et la supprimer de J_i ;
- **tan que** $J_1 \neq \emptyset$ $(J_2 \neq \emptyset)$ **faire**
  - ordonnancer les tâches de J_1 (J_2) dans un ordre arbitraire, tout en les supprimant de la liste ;

5.1.2 Expérimentation numérique

Nous avons codé les heuristiques précédentes en langage C++. Les tests ont été faits sur un ordinateur personnel avec un processeur Intel(R) Core(TM) i3-3217U 1.80GHz et une RAM de 4GB. Les expérimentations ont été faites sur un total de 3200 instances avec un nombre de tâches n qui varie dans $\{20, 100, 1000, 2000\}$. Les temps de préparation et de traitement sont générés de sort que le type d'instance varie avec un paramètre c dans $\{0.0, 0.1, 0.2, \ldots, 0.6, 0.7\}$. Pour $c < 0.3$ les temps de préparation sont plus courts que les temps de traitement, pour $c = 0.3$ les temps de préparation sont proches des temps de traitement et pour $c > 0.3$ les temps de préparation sont plus long que les temps de traitement. D'abord, nous gênerons la durée d'exécution $P_{ij} = p_{ij} + s_{ij}$ pour chaque opération d'une distribution uniforme dans $[10, 100]$. Ensuite, nous gênerons les temps de préparation de chaque opération O_{ij} aussi d'une distribution uniforme dans $[\lfloor c * P_{ij} \rfloor, \lfloor (0.3 + c) * P_{ij} \rfloor]$. Ainsi, à partir des deux valeurs précédentes on calcule $p_{ij} = P_{ij} - s_{ij}$, $i = 1, 2$, $j = 1, \ldots, n$. Pour chaque nombre de tâches n nous avons varié le paramètre c et généré 100 instances.

Nous avons suivi dans cette distribution la méthode utilisée dans [48]. Cependant, nous avons considéré plusieurs valeurs du paramètre c afin d'obtenir plus de variation dans les types d'instances. L'idée derrière cette approche est que si les temps de préparation dépendent de la tâche et de la machine alors une relation entre les temps de préparation et les temps de traitement peut probablement exister. Si c'est le cas, alors cette distribution exprime cette relation.

Nous avons considéré trois paramètres pour évaluer les heuristiques. Le premier paramètre est le nombre de fois que le makespan d'une solution donnée par une heuristique est égal au maximum des bornes inférieures mentionnées dans le Chapitre 2. Le second et le troisième paramètre sont la moyenne et le maximum de $f(H_k) = (C_{\max}(H_k) - LB)/LB$. Le premier paramètre est donné dans les Figures 5.1.1,5.1.2,5.1.3 et 5.1.4, pour un nombre de tâches égal à 20, 100, 1000 et 2000 respectivement. Les résultats montrent que deux heuristiques sont dominantes. L'heuristique H_7 est dominante lorsque les temps de traitement sont plus longs que les temps de préparation, alors que H_6 l'est pour les instances avec des temps de préparation plus longs.

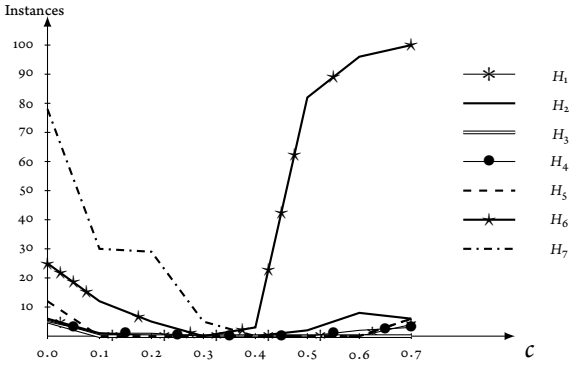


Figure 5.1.1 – $C_{\max}(H_k) = LB$; $n = 20$

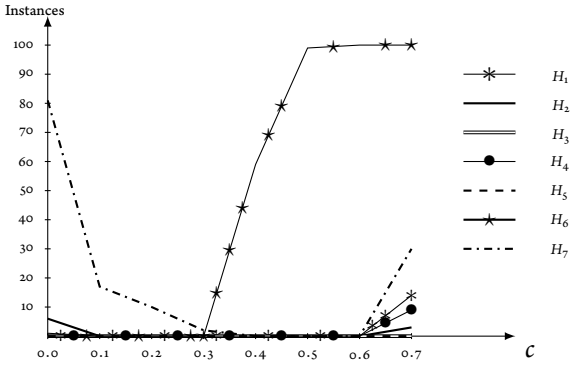


Figure 5.1.2 – $C_{\max}(H_k) = LB$; $n = 100$

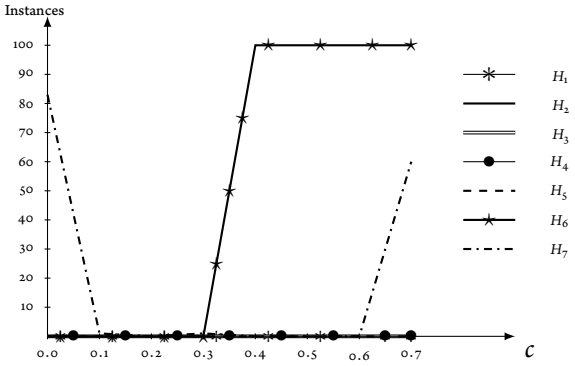


Figure 5.1.3 – $C_{\max}(H_k) = LB$; $n = 1000$

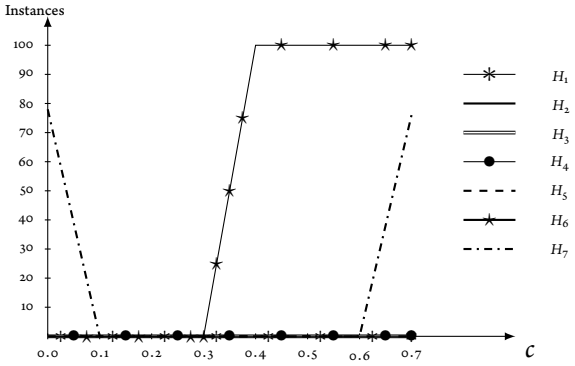


Figure 5.1.4 – $C_{\max}(H_k) = LB$; $n = 2000$

Dans les Figures 5.1.5, 5.1.7, 5.1.9 et 5.1.11 est résumée la moyenne de $f(H_k) = (C_{\max}(H_k) - LB)/LB$, pour un nombre de tâches égal à 20, 100, 1000 et 2000, respectivement. Les Figure 5.1.6, 5.1.8, 5.1.10 et

5.1.12 représentent le maximum de $f(H_k) = (C_{\max}(H_k) - LB)/LB$, pour des instances de tailles suivantes, 20, 100, 1000 et 2000 tâches, respectivement.

Nous pouvons voir que la qualité des solutions produites par les algorithmes basés sur le tri est supérieure lorsqu'il est utilisé avec l'ordre décroissant (heuristique H_2). De plus, la règle GARD est meilleure lorsque l'on considère les durées d'exécution, car l'heuristique H_5 produit des solutions de meilleure qualité que celle de l'heuristique H_4. De plus, les résultats montrent que l'heuristique H_2 est plus efficace que H_5 pour les instances avec 20 tâches. Cependant, H_5 devient plus efficace pour les autres cas. Néanmoins, la qualité des solutions produites par H_2 et H_5 n'est pas aussi bonne que celle de H_6 et H_7. Comme on pouvait s'y attendre, la règle GARD n'est pas la meilleure approche pour résoudre le problème lorsque les temps de préparation sont pris en compte.

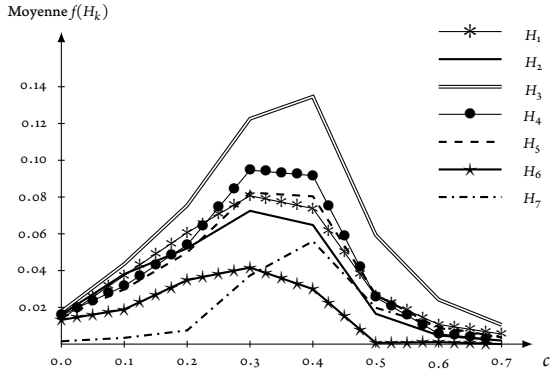


Figure 5.1.5 – Moyenne de $f(H_k)$, $n = 20$

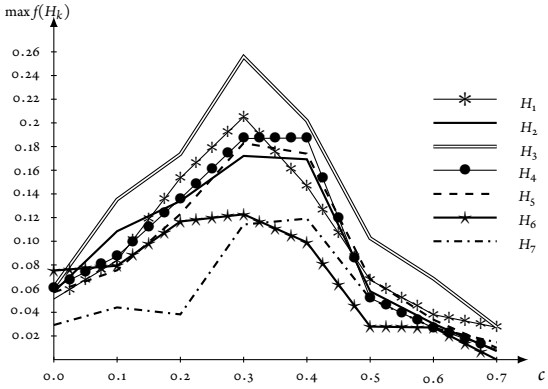


Figure 5.1.6 – max $f(H_k)$, $n = 20$

Nous avons observé que la plus grande valeur de la moyenne ainsi que le maximum de $f(H_k)$, de toutes les heuristiques, est atteinte pour $c = 0.3$ et $c = 0.4$. Ceci peut être dû au fait qu'on est face aux instances difficiles ou que les bornes inférieures proposées ne sont pas efficaces pour ces types d'instance.

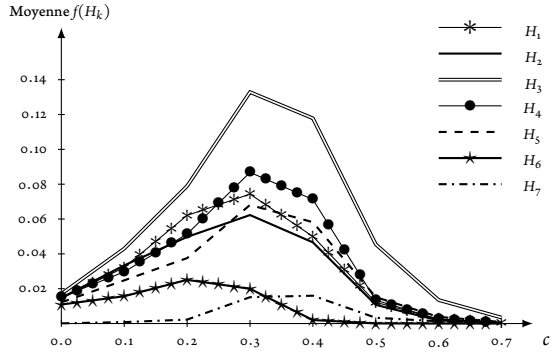


Figure 5.1.7 – Moyenne de $f(H_k)$, $n = 100$

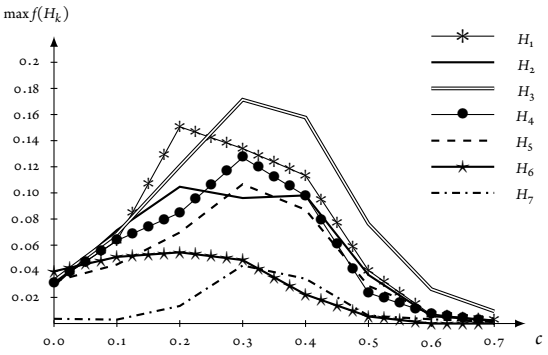


Figure 5.1.8 – max $f(H_k)$, $n = 100$

Les résultats précédents montrent clairement que H_7 et H_6 sont les meilleures heuristiques pour $0.0 \leq c \leq 0.3$ et $c \geq 0.4$, respectivement. Ceci fait que le choix de la meilleure méthode dépend du type d'instance

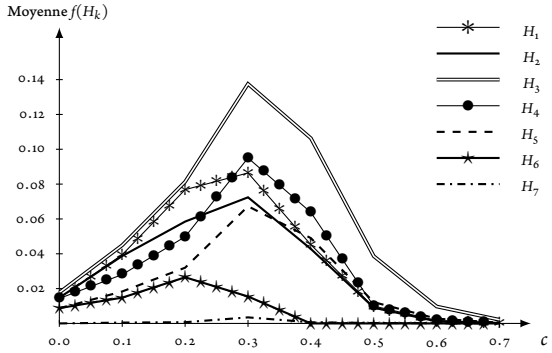


Figure 5.1.9 – Moyenne de $f(H_k)$, $n = 1000$

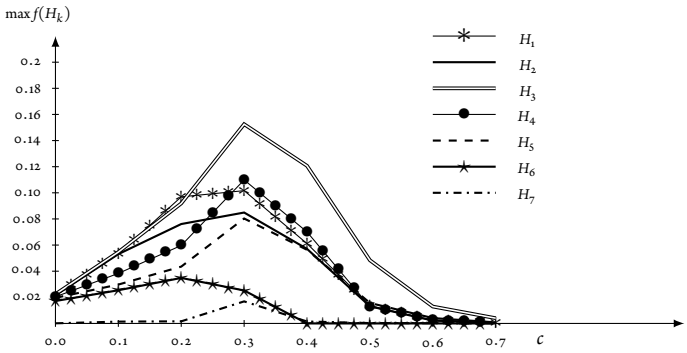


Figure 5.1.10 – max $f(H_k)$, $n = 1000$

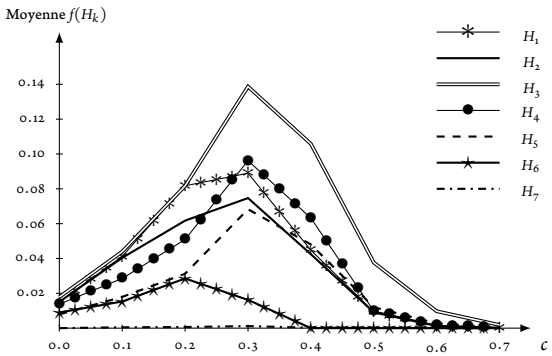


Figure 5.1.11 – Moyenne de $f(H_k)$, $n = 2000$

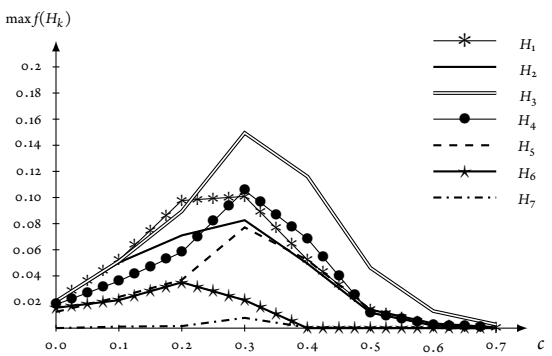


Figure 5.1.12 – max $f(H_k)$, $n = 2000$

et qui à son tour dépend de si les temps de préparation sont plus grand que les temps de traitement. Il s'en suit que le choix de la bonne heuristique à utiliser dépend de l'instance qu'on a à portée de main.

5.1.3 Hyper-heuristique

Nous avons combiné les deux heuristiques dominantes, à savoir H_6 et H_7, pour obtenir une hyper-heuristique capable de choisir la bonne heuristique selon l'instance actuelle. Pour déduire la bonne méthode qui nous permit de choisir l'heuristique à utiliser pour résoudre une instance arbitraire, d'abord, on a comparé les deux premières bornes inférieures, LB_1 et LB_2, présentées précédemment. 100 instances sont générées pour chaque type d'instance.

Un type d'instance est défini par la valeur du paramètre c et du nombre de tâches n. Pour chaque valeur de c et pour $n = 20, 100, 1000, 2000$, le nombre total de fois où $LB_1 < LB_2$ et où $LB_1 > LB_2$ sont calculés. Les résultats obtenus sont résumés dans le Tableau 5.1.1.

La même distinction de type d'instance par rapport à c apparait pour les deux bornes comme c'était le cas

	c	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
$n = 20$	$LB_1 > LB_2$	100	100	100	100	13	0	0	0
	$LB_1 < LB_2$	0	0	0	0	87	100	100	100
$n = 100$	$LB_1 > LB_2$	100	100	100	100	0	0	0	0
	$LB_1 < LB_2$	0	0	0	0	100	100	100	100
$n = 1000$	$LB_1 > LB_2$	100	100	100	100	0	0	0	0
	$LB_1 < LB_2$	0	0	0	0	100	100	100	100
$n = 2000$	$LB_1 > LB_2$	100	100	100	100	0	0	0	0
	$LB_1 < LB_2$	0	0	0	0	100	100	100	100

Tableau 5.1.1 – Comparaison entre LB_1 et LB_2

pour les deux heuristiques H_6 et H_7. Nous pouvons, par conséquent, obtenir la classification d'une instance en comparant ces bornes inférieures. Ce qui nous permet de savoir quelle heuristique utiliser. Une hyper-heuristique est ainsi dérivée comme le résume l'Algorithme 14.

Algorithme 14 : Hyper-heuristique H

début
  calculer la valeur de LB_1 et LB_2 ;
  si $LB_1 > LB_2$ **alors**
    utiliser **heuristique** H_7 ;
  sinon
    utiliser **heuristique** H_6 ;

La Figure 5.1.13 illustre la moyenne de $(C_{\max}(H) - LB)/LB)$ de l'hyper-heuristique H pour $n = 20$, 100, 1000 et 2000. Nous pouvons voir qu'elle combine efficacement les deux heuristiques.

5.2 Méthodes exactes

Pour une résolution exacte du problème étudié, dès lors qu'il est $\mathcal{NP}$-difficile au sens fort, nous proposons dans cette section deux formulations en programme mathématique linéaire en nombres entiers. Les deux modèles sont ensuite testés via une batterie de tests que nous avons effectués.

5.2.1 Modèles mathématiques

Nous présenterons, dans un premier temps, les deux modèles, puis nous passerons à l'étude de leur performance. Le premier modèle est basé sur une définition directe des temps de début de traitement de chaque

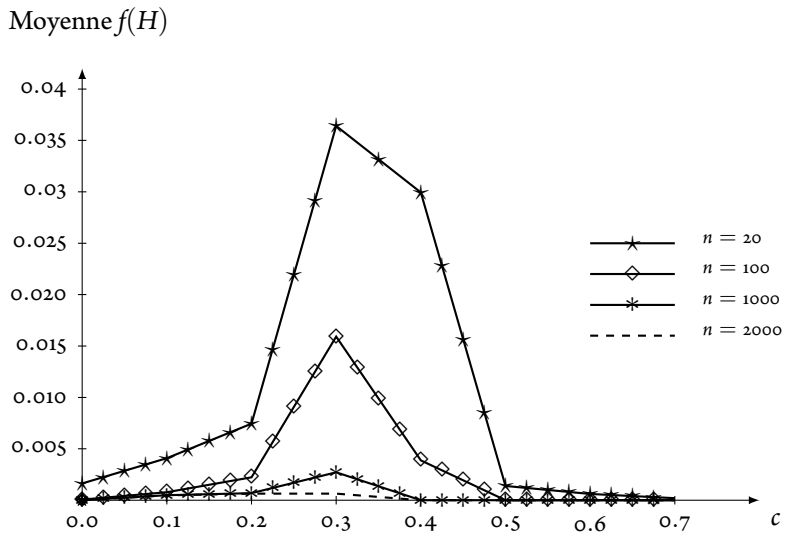


Figure 5.1.13 – Moyenne de $f(H)$

tâche, ainsi que, les relations de précédence entre les opérations. La plupart des contraintes sont basées sur la relation de précédence entre les opérations. Ainsi donc, les variables de décision du modèle sont,

t_{ij} : la date de début de traitement de l'opération i ; $i = 1, 2$ de la tâche T_j ; $j = 1, \ldots, n$.

Les variables binaires qui expriment les relations de précédence entre les opérations et qui sont définies comme suit :

$$x_{iji'j'} = \begin{cases} 1 & \text{si } O_{ij} \text{ commence avant } O_{i'j'} \text{ ; } i, i' = 1, 2; j, j' = 1, \ldots, n, \\ 0 & \text{sinon.} \end{cases}$$

Nous définissons aussi, pour $i = 1, 2$ et $j = 1, \ldots, n$, ce qui suit :

$$a_{ij} = \begin{cases} 1 & \text{si } s_{ji} > 0, \\ 0 & \text{sinon.} \end{cases}$$

M : une borne supérieure du makespan.

L'objectif est de minimiser le temps de fin de traitement final, y. Le modèle complet est le suivant :

$$
\begin{array}{|l}
\min y\,; \\
t.q. \quad a_{i'j'} \times (t_{ij} + s_{ij}) \leq t_{i'j'} + M \times x_{iji'j'}; \qquad j, j' = 1, \ldots, n\,; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i, i' = 1, 2\,; (i, j) \neq (i', j'). \quad (1) \\
\qquad a_{ij} \times (t_{i'j'} + s_{i'j'}) \leq t_{ij} + M \times (1 - x_{iji'j'}); \quad j, j' = 1, \ldots, n\,; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i, i\prime = 1, 2\,; (i, j) \neq (i', j'). \quad (2) \\
\qquad t_{ij} + s_{ij} + p_{ij} \leq t_{ij'} + M \times x_{ijij'} \qquad j, j' = 1, \ldots, n\,; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i = 1, 2\,; j \neq j'. \quad (3) \\
\qquad t_{ij'} + s_{ij'} + p_{ij'} \leq t_{ij} + M \times (1 - x_{ijij'}); \quad j, j' = 1, \ldots, n\,; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i = 1, 2\,; j \neq j'. \quad (4) \\
\qquad t_{ij} + s_{ij} + p_{ij} \leq t_{i'j} + M \times x_{iji'j}; \qquad j = 1, \ldots, n\,; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i, i' = 1, 2\,; i \neq i'. \quad (5) \\
\qquad t_{ij} + s_{ij} + p_{ij} \leq y\,; \qquad\qquad\qquad\qquad\qquad\qquad j = 1, \ldots, n\,; i = 1, 2. \quad (6)
\end{array}
\tag{P1}
$$

(1) et (2) assurent que le serveur prend en charge une préparation à la fois. Lorsque le serveur est occupé, les seules tâches qui peuvent être lancées sont celles avec un temps de préparation nul.

(3) et (4) assurent qu'une machine ne traite qu'une tâche à la fois.

(5) font que les opérations de la même tâche ne soient pas ordonnancées en même temps.

(6) indiquent que le temps de fin de traitement de chaque tâche est inférieur ou égal à y.

Le second modèle détermine la séquence des préparations des tâches faites par le serveur, en affectant chaque opération à une position. Puisque les préparations ne peuvent pas se chevaucher alors un ordre de préparation est défini pour les opérations. Ainsi, nous définissons les variables de décision suivantes pour $j = 1, \ldots, n\,; i = 1, 2\,; k = 1, \ldots, 2n$:

$$
x_{ijk} = \begin{cases} 1 & \text{si l'opération } O_{ij} \text{ est en position } k, \\ 0 & \text{sinon.} \end{cases}
$$

t_k : la date de début de traitement de l'opération qui est à k^{me} position, $k = 1, \ldots, 2n$.
M : une borne supérieure.

$$
\begin{array}{|l}
\min y; \\
\text{s.t.} \quad \sum_{j=1}^{n}\sum_{i=1}^{2} x_{ijk} = 1; \qquad k = 1,\ldots,2n. \qquad (1) \\
\qquad \sum_{k=1}^{2n} x_{ijk} = 1; \qquad i = 1,2; \qquad (2) \\
\qquad\qquad\qquad\qquad\qquad\quad j = 1,\ldots,n. \\
\qquad t_k + \sum_{j=1}^{n}\sum_{i=1}^{2} x_{ijk} \times s_{ij} \leq t_{k+1}; \qquad k = 1,\ldots,2n-1. \qquad (3) \\
\qquad t_k + \sum_{j=1}^{n} x_{ijk} \times (s_{ij} + p_{ij}) - t_{k'} \leq M \times (1 - \sum_{j=1}^{n} x_{ijk'}); \quad i = 1,2; \qquad (4) \\
\qquad\qquad\qquad\qquad\qquad\quad k = 1,\ldots,2n-1; \\
\qquad\qquad\qquad\qquad\qquad\quad k' = k+1,\ldots,2n. \\
\qquad t_k + (s_{ij} + p_{ij}) - t_{k'} \leq M \times (2 - x_{ijk} - x_{i'jk'}); \qquad j = 1,\ldots,n; \qquad (5) \\
\qquad\qquad\qquad\qquad\qquad\quad i \neq i' = 1,2; \\
\qquad\qquad\qquad\qquad\qquad\quad k = 1,\ldots,2n-1; \\
\qquad\qquad\qquad\qquad\qquad\quad k' = k+1,\ldots,2n. \\
\qquad t_k + \sum_{j=1}^{n}\sum_{i=1}^{2} x_{ijk} \times (s_{ij} + p_{ij}) \leq y; \qquad k = 1,\ldots,2n. \qquad (6)
\end{array}
\qquad (P2)
$$

(1) assurent qu'il n'y ait qu'une seule opération par position.

(2) font en sorte qu'une opération soit affectée à une seule position.

(3) assurent que deux préparations qui se succèdent ne se chevauchent pas.

(4) font qu'une machine traite une tâche à la fois.

(5) assurent que les opérations de la même tâche ne soient pas ordonnancées simultanément.

(6) assurent que le temps de fin de traitement de toutes les tâches soit inférieur ou égal à y.

5.2.2 Comparaison des modèles

D'abord, nous comparons les composantes des deux modèles : le nombre de variables et le nombre de contraintes. La taille des deux modèles est résumée dans le Tableau 5.2.1. Comme on peut voir, on a plus de variables binaires dans le second modèle, tandis que le nombre de variables entières est le même pour les deux modèles. La deuxième différence réside dans le nombre de contraintes où le second modèle a plus de contraintes que le premier.

Passons, maintenant, à la comparaison des performances des deux modèles. Nous avons conduit une étude de performance en utilisant IBM ILOG CPLEX Optimization Studio 12.7, sur un ordinateur personnel avec un processeur Intel (R) Core (TM) i5-5300U, 2.30 GHz et une RAM de 8 GB. Le nombre de tâches varie dans $\{5, 10, 15, 20 \ldots, 70\}$. Les temps de préparation et de traitement sont générés en utilisant la même

Modèle	Variables binaires	Variables entières	Contraintes
(P1)	$2n^2$	$2n+1$	$8n^2-n$
(P2)	$4n^2$	$2n+1$	$12n^3-12n^2+11n-1$

Tableau 5.2.1 – La taille des modèles mathématiques

méthode que celle utilisée pour les heuristiques. Nous gênerons, donc, la durée d'exécution $P_{ij} = p_{ij} + s_{ij}$ pour chaque opération d'une distribution uniforme dans $[10, 100]$. Par la suite, nous gênerons les temps de préparation de chaque opération O_{ij}, aussi d'une distribution uniforme dans $[\lfloor c * P_{ij} \rfloor, \lfloor (0.3+c) * P_{ij} \rfloor]$. On calcule ensuite le temps de traitement de chaque opération $p_{ij} = P_{ij} - s_{ij}$, $i = 1, 2$, $j = 1, \ldots, n$. Pour une valeur des deux paramètres (n, c), nous avons généré 10 instances. Un temps d'exécution limite de 3600 secondes est imposé pour chaque instance. Dans ce qui suit, nous comparons les modèles en comparant la moyenne de leurs temps d'exécution en secondes pour chaque type d'instance.

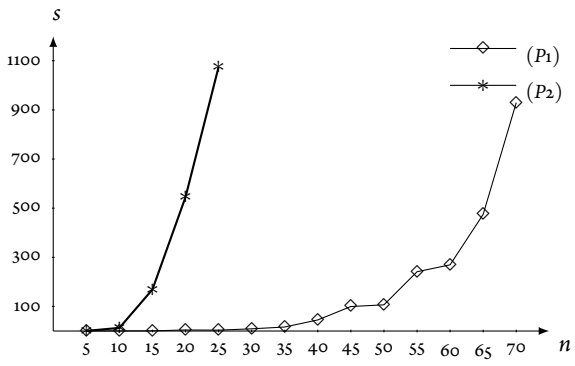


Figure 5.2.1 – $c = 0.0$

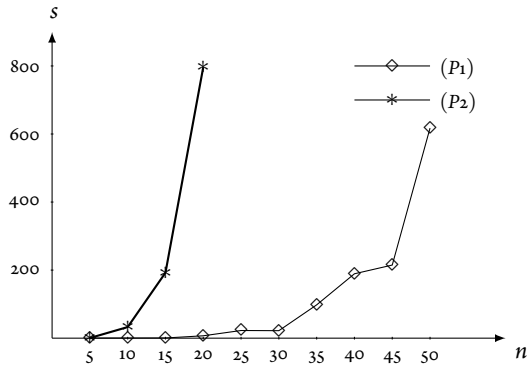


Figure 5.2.2 – $c = 0.1$

Pour le premier type d'instances, avec $c = 0.0$, comme nous pouvons le voir dans la Figure 5.2.1, le modèle (P1) surpasse le modèle (P2) et il peut résoudre des instances de jusqu'à 70 tâches. Tandis que le temps d'exécution de (P2) augmente rapidement au point qu'il ne peut résoudre que des instances d'au plus 25 tâches.

Concernant le deuxième type d'instances, avec $c = 0.1$, le modèle (P1) est toujours le meilleur modèle, mais il est moins efficace que pour le premier type. Il est capable de résoudre des instances de jusqu'à 50 tâches. Cependant (P2) ne peut résoudre que des instances d'au plus 20 tâches, comme illustré dans la Figure 5.2.2.

Pour le troisième type d'instances, avec $c = 0.2$, comme le montre la Figure 5.2.3, le modèle (P1) est capable de résoudre des instances de 30 tâches, alors que le modèle (P2) résout des instances avec 20 tâches. En ce qui concerne le quatrième type, $c = 0.3$, nous pouvons clairement voir, dans la Figure 5.2.4, qu'il

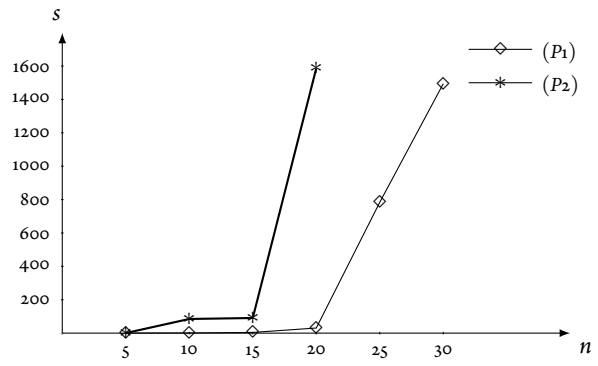


Figure 5.2.3 – $c = 0.2$

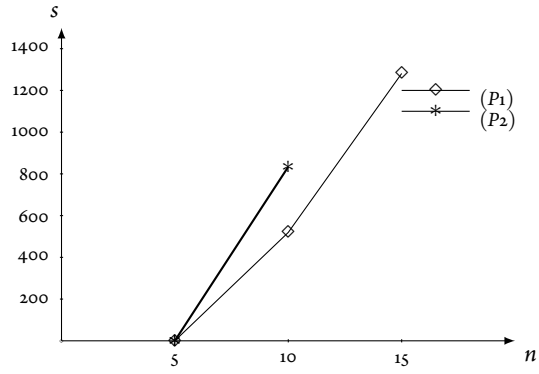


Figure 5.2.4 – $c = 0.3$

s'agit de l'un des types d'instances les plus difficiles. Le modèle (P2) peut résoudre certaines instances, mais uniquement celles avec 10 tâches. Le modèle (P1) est capable de résoudre des instances avec $n = 15$.

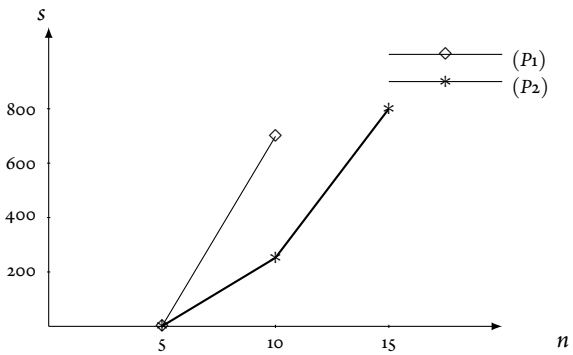


Figure 5.2.5 – $c = 0.4$

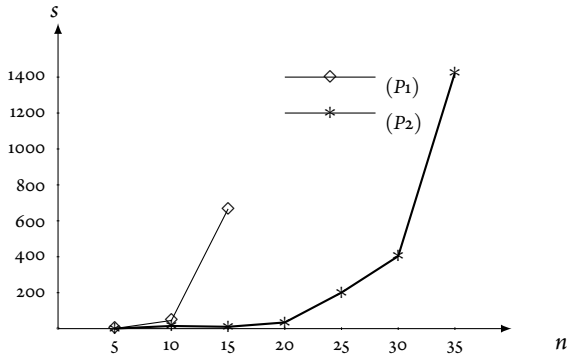


Figure 5.2.6 – $c = 0.5$

Le deuxième type d'instances difficiles correspond au type avec $c = 0.4$. Nous pouvons voir, dans la Figure 5.2.5, que le modèle (P2) est meilleur que le modèle (P1) et il est capable de résoudre des instances de 15 tâches. Alors que le modèle (P1) a résolu des instances de seulement 10 tâches.

En ce qui concerne les instances avec $c = 0.5$, les performances de (P2) sont meilleures, car des instances de jusqu'à 35 tâches sont résolues, alors que (P1) est beaucoup plus lent et n'a pu résoudre que des instances d'au plus 15 tâches. Les performances des modèles pour ce type sont résumées dans la Figure 5.2.6.

Pour $c = 0.6$, le modèle (P2) résout des instances avec jusqu'à 40 tâches, tandis que le modèle (P1) résout des instances avec 15 tâches, comme le montre la Figure 5.2.7. Quant au dernier type d'instances, avec $c = 0.7$, nous pouvons voir sur la Figure 5.2.8 que le modèle (P2) résout des instances avec 55 tâches. Cependant, le modèle (P1) n'a pu résoudre que des instances avec 20 tâches.

Il est assez facile de distinguer les types d'instances pour lesquelles un modèle surpasse l'autre. En effet, le

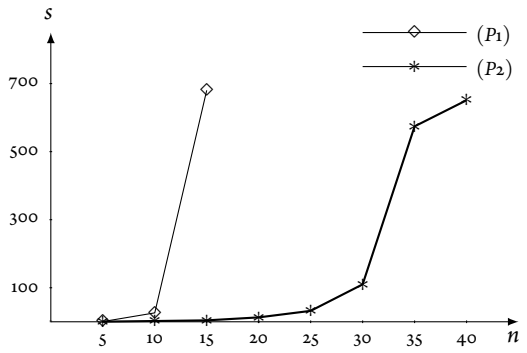


Figure 5.2.7 – $c = 0.6$

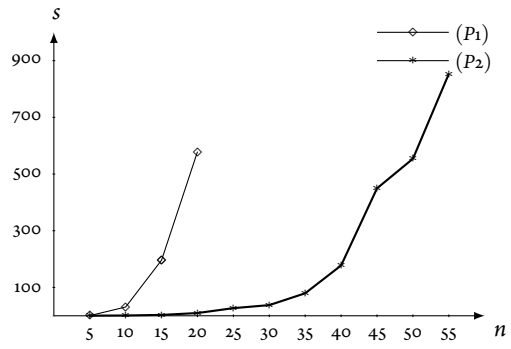


Figure 5.2.8 – $c = 0.7$

modèle (P1) a une meilleure performance pour le type d'instances avec $c \leq 0.3$. Cependant, pour $c > 0.3$, le modèle (P2) devient le modèle le plus performant. Néanmoins, à travers les résultats que nous avons obtenus de cette étude expérimentale nous pouvons conclure que les deux modèles mathématiques sont efficaces.

Conclusion

Nous avons développé des heuristiques pour résoudre le problème général. Ensuite, nous avons mené des tests pour étudier leurs performances. Les résultats montrent que deux heuristiques, H_6 et H_7, sont dominantes, mais chacune pour un type distinct d'instances. Nous avons donc proposé une classification des instances afin de pouvoir choisir l'heuristique adéquate pour résoudre le problème.

Nous utilisons pour cela les deux bornes inférieures afin de trouver une classification d'instance. En comparant les bornes inférieures LB_1 et LB_2, nous avons remarqué la même distinction dans les types d'instances que celle observée dans la performance de H_6 et H_7 durant l'étude expérimentale. Nous avons donc proposé une hyper-heuristique qui combine efficacement les deux heuristiques dominantes.

De plus, nous avons proposé deux formulations de programmation linéaire en nombres entiers mixtes pour résoudre le problème. Les deux modèles sont testés à travers une étude expérimentale. Cette dernière montre que les deux modèles sont efficaces, dans le sens où l'un est meilleur que l'autre pour différents types d'instances. Les deux modèles sont complémentaires, car l'un est efficace lorsque l'autre ne l'est pas et vice versa.

6

Résolution du problème du job shop

Nous présentons, dans ce qui suit, les méthodes approchées que nous avons élaboré pour résoudre le problème $J2, S1|s_{ij}|C_{\max}$. Dans un premier temps, nous décrirons les heuristiques que nous avons utilisé pour obtenir les solutions initiales des méta-heuristiques. Nous décrirons, par la suite, les méta-heuristiques que nous avons développées pour résoudre le problème. Nous conclurons par l'études expérimentale que nous avons mené pour tester ces méthodes.

6.1 Heuristiques

Nous commençons par la présentation des heuristiques ensuite nous entamons l'étude expérimentale que nous avons menée pour les tester.

6.1.1 Description des heuristiques

Dans un premier temps, nous décrirons les heuristiques. Principalement, nous avons trois heuristiques, néanmoins, la variation dans les procédures. Les procédures qui construisent les sous-ensembles de tâches ou celles qui choisissent la prochaine tâche à traiter dans l'ordonnancement, nous donne un total de 10 heuristiques.

Les heuristiques sont résumées dans les Algorithmes 15, 16 et 17. Nous avons testé ces heuristiques lors d'une étude expérimentale que présentons par la suite.

Algorithme 15 : Heuristique $H_h, h = 1, \ldots, 4$

début
- soient t_1, t_2 et t l'instant auquel M_1, M_2 et $S1$ deviennent libre, respectivement ;
- initialiser $t_1 = t_2 = t = 0$ et définir $m_1 = \sum_j (s_{1j} + p_{1j})$,$m_2 = \sum_j (s_{2j} + p_{2j})$ et $Q'_1 = Q'_2 = \emptyset$;
- construire les listes : Q_1, Q_2, Q_3, Q_4, Q'_1 et Q'_2 en utilisant $Proc_{C_h}$;
- **tan que** $Q_1 \cup Q_2 \cup Q_3 \cup Q_4 \cup Q'_1 \cup Q'_2 \neq \emptyset$ **faire**
  - **si** $t_1 < t_2$ *ou* $(t_1 = t_2$ *et* $m_1 > m_2)$ **alors**
    - ordonnancer T_j la première tâche de la liste Q_1, Q_3, Q'_2 sur M_1 et l'enlever de sa liste ;
  - **sinon**
    - ordonnancer T_j la première tâche de la liste Q_2, Q_4, Q'_1 sur M_2 et l'enlever de sa liste ;
  - **si** T_j *était de* Q_1 *(resp.* Q_2*)* **alors**
    - ajouter T_j à Q'_1 (resp. Q'_2) comme étant la dernière tâche de la liste ;

Nous avons quatre procédures de construction. Elles sont toutes basées sur la méthode de l'algorithme de Jackson. La première construit une séquence arbitraire. La seconde considère seulement les temps de préparation. La troisième considère les durées d'exécution dans la première phase puis seulement les temps de traitement dans la deuxième phase. La dernière considère les durées d'exécution dans les deux phases. Les procédures de construction sont nommées $Proc_{C_k}, k = 1, \ldots, 4$ et sont résumées dans Algorithme 18.

En ce qui concerne les procédures qui choisissent la prochaine tâche à ordonnancer, appelées procédure de choix, nous avons trois procédures. La première et la seconde sont basées sur la réduction des temps morts causés par l'indisponibilité du serveur. Pour cela, on définit ℓ comme étant la longueur de l'intervalle de temps où le serveur et seulement une des deux machines sont libres. Dans cet intervalle, on ordonnance une tâche avec un temps de traitement égal ou proche de ℓ. Ceci est utilisé dans procédures, l'une minimise la valeur absolue de la différence entre ℓ et le temps de préparation, l'autre favorise les temps de préparation plus courts que ℓ pour éviter le temps mort. La dernière procédure se concentre sur la réduction des temps d'arrêt du serveur. Par conséquent, elle ordonnance les préparations à des intervalles rapprochés, en ordon-

Algorithme 16 : Heuristique $H_{1.k}$, $k, \ldots, 3$

début

soient t_1, t_2 et t l'instant auquel M_1, M_2 et $S1$ deviennent libre, respectivement ;
initialiser $t_1 = t_2 = t = 0$ et définir $m_1 = \sum_j (s_{1j} + p_{1j})$, $m_2 = \sum_j (s_{2j} + p_{2j})$ et $N'_1 = N'_2 = \emptyset$;
tan que $N_1 \cup N_2 \cup N_3 \cup N_4 \cup N'_1 \cup N'_2 \neq \emptyset$ **faire**
  si $t_1 < t_2$ *ou* $(t_1 = t_2$ *et* $m_1 > m_2)$*(resp.*$t_2 < t_1$ *ou* $(t_1 = t_2$ *et* $m_2 \leq m_1)$ *)* **alors**
    si $N_1 \cup N_3 \cup N'_2 \neq \emptyset$ **alors**
      soit N le premier sous-ensemble non-vide parmi les sous-ensembles ordonés comme suit : N_1, N_3, N'_2 (resp. N_2, N_4, N'_1) ;
      $Proc_{Ch_k}(M_1, N)$ (resp. $Proc_{Ch_k}(M_2, N)$) ;
    sinon
      soit N le premier sous-ensemble non-vide parmi les sous-ensembles ordonés comme suit : N_2, N_4, N'_1 (resp. N_1, N_3, N'_2) ;
      $Proc_{Ch_k}(M_2, N)$ (resp. $Proc_{Ch_k}(M_1, N)$) ;
  ordonnancer T_j, la tâche sélectionnée, sur la machine associée, et l'enlever du sous-ensemble ;
  si T_j *était de* N_1 *(resp.* N_2*)* **alors**
    ajouter T_j à N'_1 (resp. N'_2) ;

Algorithme 17 : Heuristique $H_{2.k}$, $k = 1, \ldots, 3$

début

soient t_1, t_2 et t l'instant auquel M_1, M_2 et $S1$ deviennent libre, respectivement ;
initialiser $t_1 = t_2 = t = 0$ et définir $m_1 = \sum_j (s_{1j} + p_{1j})$, $m_2 = \sum_j (s_{2j} + p_{2j})$ et $N'_1 = N'_2 = \emptyset$;
tan que $(N_1 \cup N_2 \cup N_3 \cup N_4 \cup N'_1 \cup N'_2 \neq \emptyset)$ **faire**
  si $(t_1 < t_2)$ ***ou*** $((t_1 = t_2)$ ***et*** $(m_1 > m_2))$ *(resp.* $(t_2 > t_1)$ ***ou*** $((t_1 = t_2)$ ***et*** $(m_1 \leq m_2)))$ **alors**
    si $N_1 \cup N_3 \cup N'_2 \neq \emptyset$ *(resp.* $N_2 \cup N_4 \cup N'_1 \neq \emptyset$*)* **alors**
      $Proc_{Ch_k}(M_1, N_1 \cup N_3 \cup N'_2)$ (resp. $Proc_{Ch_k}(M_2, N_2 \cup N_4 \cup N'_1)$) ;
    sinon si $N_2 \cup N_4 \cup N'_1 \neq \emptyset$ *(resp.* $N_1 \cup N_3 \cup N'_2 \neq \emptyset$*)* **alors**
      $Proc_{Ch_k}(M_2, N_2 \cup N_4 \cup N'_1)$ (resp. $Proc_{Ch_k}(M_1, N_1 \cup N_3 \cup N'_2)$) ;
    ordonnancer T_j, la tâche sélectionnée, sur M_1 (resp. M_2), et l'enlever du sous-ensemble ;
    si T_j *était de* N_1 *(resp.* N_2*)* **alors**
      ajouter T_j à N'_1 (resp. N'_2) ;

nançant les phases de traitement en parallèle des phases de préparations les plus longues. Cette approche est plus intéressante lorsqu'on a des temps de préparation largement plus grands que les temps de traitement. Les procédures de choix sont nommées $Proc_{Ch_k}$, $k = 1, \ldots, 3$ et sont résumées dans Algorithme 19.

Nous avons combiné les procédures de choix et de construction et obtenu un total de 10 heuristiques. Heuristiques H_1, H_2, H_3 et H_4 utilisent, respectivement, $Proc_{C_1}$, $Proc_{C_2}$, $Proc_{C_3}$ et $Proc_{C_4}$ comme procédure

Algorithme 18 : $Proc_{C_h}, h = 1, \ldots, 4$

début

Définir : $Q_1 = \{T_j, M_1 \rightarrow M_2\}$, $Q_2 = \{T_j, M_2 \rightarrow M_1\}$, $Q_3 = \{T_j, M_1\}$, $Q_4 = \{T_j, M_2\}$, $Q'_1 = \emptyset$ et $Q'_2 = \emptyset$;

si $h = 2$ **alors**

définir $Q^1_1 = \{T_j \in Q_1 \text{ t.q. } p_{2j} > p_{1j}\}$, $Q^2_1 = \{T_j \in Q_1 \text{ t.q. } p_{2j} \leq p_{1j}\}$;

définir $Q^1_2 = \{T_j \in Q_2 \text{ t.q. } p_{1j} > p_{2j}\}$, $Q^2_2 = \{T_j \in Q_2 \text{ t.q. } p_{1j} \leq p_{2j}\}$;

sinon si $h = 3$ *ou* $h = 4$ **alors**

définir $Q^1_1 = \{T_j \in Q_1 \text{ t.q. } s_{2j} + p_{2j} > s_{1j} + p_{1j}\}$, $Q^2_1 = \{T_j \in Q_1 \text{ t.q. } s_{2j} + p_{2j} \leq s_{1j} + p_{1j}\}$;

définir $Q^1_2 = \{T_j \in Q_2 \text{ t.q. } s_{1j} + p_{1j} > s_{2j} + p_{2j}\}$, $Q^2_2 = \{T_j \in Q_2 \text{ t.q. } s_{1j} + p_{1j} \leq s_{2j} + p_{2j}\}$;

si $h = 2$ *ou* $h = 3$ $(h = 4)$ **alors**

trier les tâches de Q^1_1 et Q^1_2 dans l'ordre croissant des temps de traitement (temps d'exécution) sur M_1 etM_2 respectivement ;

trier les tâches de Q^2_1 and Q^2_2 dans l'ordre décroissant des temps de traitement (temps d'exécution) sur M_2 et M_1 respectivement ;

définir les séquences des tâches de Q_1 et Q_2 comme étant : Q^1_1, Q^2_1 et Q^1_2, Q^2_2 respectivement ;

Algorithme 19 : $Proc_{Ch_k}, k = 1, \ldots, 3$

début

supposer que M_i, $i = 1, 2$ est la machine actuelle et N le sous-ensemble sélectionné

définir $\ell = t_{i'} - \max(t_i, t)$, et p le temps de traitement de la dernière tâche ordonnancée sur M'_i, $i' = 3 - i$

si $k = 1$ **alors**

sélectionner T_j telle que $|\ell - s_{ij}| = \min_{T_h \in N}\{|\ell - s_{ih}|\}$;

sinon si $k = 2$ **alors**

sélectionner T_j telle que $\ell - s_{ij} = \min_{T_h \in N}\{\ell - s_{ih} \text{ t.q. } \ell \geq s_{ij}\}$

si *aucune tâche n'est sélectionnée* **alors**

sélectionner T_j telle que $s_{ij} = \min_{T_h \in N}\{s_{ih}\}$;

sinon si $k = 3$ **alors**

sélectionner $T_j : p_{ij} = \max_{T_h \in N}\{p_{ih} \text{ t.q. } s_{ij} = p\}$; **si** *aucune tâche n'est sélectionnée* **alors**

sélectionner T_j telle que $p_{ij} = \max_{T_h \in N}\{p_{ih} \text{ t.q. } s_{ij} > p\}$;

si *aucune tâche n'est sélectionnée* **alors**

sélectionner T_j telle que $s_{ij} = \max_{T_h \in N}\{s_{ih}\}$;

de construction de séquences. Tandis que, heuristiques $H_{1.1}$, $H_{1.2}$ et $H_{1.3}$ utilise, respectivement, $Proc_{Ch_1}$, $Proc_{Ch_2}$ et $Proc_{Ch_3}$ comme procédure de choix. Finalement, heuristiques $H_{2.1}$, $H_{2.2}$ et $H_{2.3}$ utilise aussi, respectivement, $Proc_{Ch_1}$, $Proc_{Ch_2}$ et $Proc_{Ch_3}$ comme procédures de choix.

Heuristiques H_1, H_2, H_3 et H_4 sont de différentes variations de l'algorithme de Jackson qui résout $J2||C_{\max}$.

La différence entre ces versions est la considération des temps de préparation dans les procédures de construction des séquences. Tandis que, la différence entre $H_{1.1}$, $H_{1.2}$, $H_{1.3}$ et $H_{2.1}$, $H_{2.2}$, $H_{2.3}$ est que les premières sélectionnent la prochaine tâche à traiter sur M_1 (resp. M_2) dans l'ordonnancement à partir de N_1 (resp. N_2) et ce n'est que si cet ensemble est vide qu'elles la sélectionnent de N_3 (resp. N_4), et ainsi de suite. Par contre les autres heuristiques sélectionnent la prochaine tâche sur M_1 (resp. M_2) à partir de $N_1 \cup N_3 \cup N'_2$ (resp. $N_2 \cup N_4 \cup N'_1$).

6.1.2 Étude expérimentale

Nous avons codé les heuristiques ci-dessus en langage C++. Les tests ont été faits sur un ordinateur personnel équipé d'un processeur Intel (R) Core (TM) i5-5300U à 2.30 GHz et une RAM de 8 GB. Les expériences sont réalisées sur un total de 3200 instances avec le nombre n de tâches dans $\{20, 100, 200, 1000\}$. Les temps de préparation et les temps de traitement sont générés de manière à ce que le type d'instance varie avec le paramètre c dans $\{0.0, 0.1, 0.2, \ldots, 0.6, 0.7\}$. Nous avons d'abord généré la durées d'exécution $P_{ij} = p_{ij} + s_{ij}$ pour chaque opération à partir d'une distribution uniforme dans $[10, 100]$. Ensuite, nous avons généré le temps de préparation de chaque opération O_{ij} également à partir d'une distribution uniforme dans $[\lfloor c * P_{ij} \rfloor, \lfloor (0.3 + c) * P_{ij} \rfloor]$. Ainsi, à partir des deux valeurs précédentes on calcule le temps de traitement $p_{ij} = P_{ij} - s_{ij}$, $i = 1, 2$, $j = 1, \ldots, n$. De plus, les cheminements des tâches sont générés comme suit. Pour chaque tâche $T_j, j = 1, \ldots, n$ nous générons R'_j à partir d'une distribution uniforme dans $\{1, 2, 3, 4\}$. Si $R'_j = 1$ alors $T_j \in N_1$ et son cheminement $R_j = M_1 \rightarrow M_2$. De la même manière, $R'_j = 2$, $R'_j = 3$ et $R'_j = 4$ correspondent respectivement à $T_j \in N_2$, $T_j \in N_3$ et $T_j \in N_4$. Conséquemment, nous posons $s_{2j} = p_{2j} = 0$ si $R'_j = 3$ et $s_{1j} = p_{1j} = 0$ lorsque $R'_j = 4$. Pour conclure la génération des instances se fait en variant le paramètre c pour chaque nombre n de tâches et générant 100 instances.

Le paramètre que nous avons utilisé pour évaluer les heuristiques est la moyenne de $f(H_k) = (\frac{C_{\max}(H_k) - LB}{LB})$. Les Figures 6.1.1, 6.1.2, 6.1.3 et 6.1.4 représentent la moyenne de $f(H_k)$, pour différentes tailles d'instances soit 20, 100, 200 et 1000 tâches, respectivement.

Les résultats montrent que les heuristiques H_1, H_2, H_3 et H_4 sont les heuristiques les moins efficaces. De plus, la différence entre la qualité des solutions produites par ces heuristiques est légèrement perceptible. Les meilleures solutions sont données par les heuristiques $H_{1.1}$ et $H_{1.2}$, ce qui signifie que considérer la différence entre ℓ et les temps de préparation est la procédure de choix la plus efficace. Cependant, la différence entre la qualité des solutions données par $H_{1.1}$, $H_{1.2}$ et $H_{2.1}$, $H_{2.2}$, respectivement, montre que donner la priorité au sous-ensemble N_1 (resp N_2) sur N_3 (resp N_4), et N_3 sur N'_2 (resp N'_1) est mieux que de sélectionner directement la prochaine tâche de l'ensemble $N_1 \cup N_3 \cup N'_2$ (resp $N_2 \cup N_4 \cup N'_1$). De plus, nous pouvons voir que l'heuristique $H_{1.1}$ est généralement meilleure que $H_{1.2}$, ce qui en fait la meilleure heuristique globale.

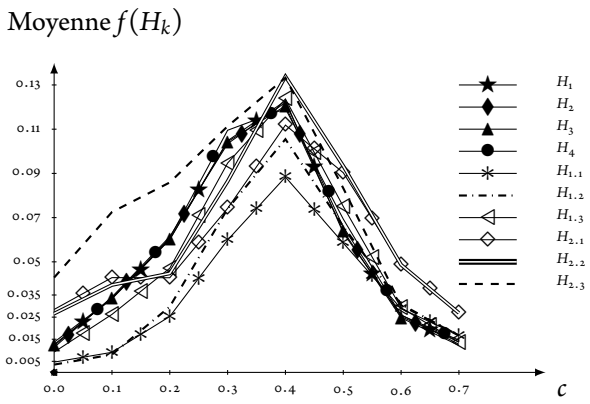


Figure 6.1.1 – Moyenne $f(H_k)$, $n = 20$

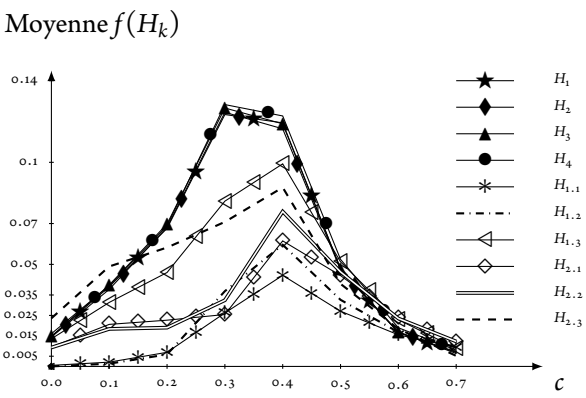


Figure 6.1.2 – Moyenne $f(H_k)$, $n = 100$

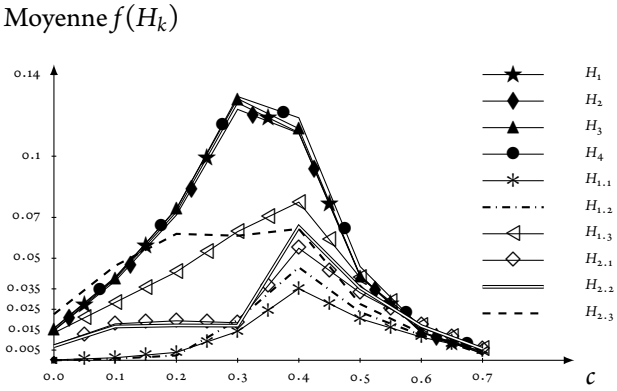


Figure 6.1.3 – Moyenne $f(H_k)$, $n = 200$

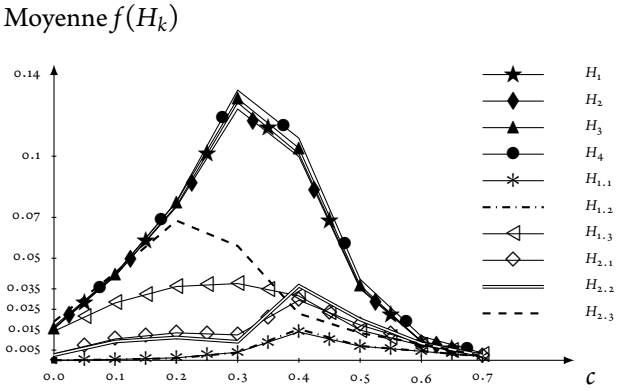


Figure 6.1.4 – Moyenne $f(H_k)$, $n = 1000$

6.2 Méta-heuristiques

Les heuristiques précédentes sont les algorithmes qui construisent les solutions initiales des méta-heuristiques. Nous avons développé un recuit simulé et un algorithme génétique. Dans un premier temps, nous allons décrire les méta-heuristiques. Ensuite on passe à l'étude expérimentale établie pour tester leur performance.

6.2.1 Recuit Simulé

Le recuit simulé que nous avons développé peut être décrit comme suit. La solution initiale est obtenue à partir des 10 heuristiques et qui correspond à la meilleure solution trouvée parmi celles-ci. De plus, l'algorithme de recuit simulé global est résumé dans Algorithme 20.

Algorithme 20 : Recuit Simulé

début
construire un ensemble de solutions réalisables
sélectionner la solution actuelle S qui est la meilleure solution de l'ensemble
initialiser la température initiale τ, la température finale τ_f, a et $k = 1$
tan que $\tau > \tau_f$ *ou* $C_{max}(S) > L$ **faire**
  tan que $k < Nb$ *ou* $C_{max}(S) > L$ **faire**
    sélectionner une solution S' en utilisant Recherche Local$_1$
    calculer $\Delta C_{max} = C_{max}(S') - C_{max}(S)$
    si $\Delta C_{max} < 0$ **alors**
      S' devient la solution actuelle ;
    sinon
      sélectionner aléatoirement une probabilité $p : 0 < p < 1$; **si** $p < \exp^{-\Delta C_{max}/\tau}$ **alors**
        S' devient la solution actuelle ;
      sinon
        aller à 4
    $k = k + 1$
  $\tau = \tau \times a$

Représentation d'une solution : un ordonnancement est représenté par un vecteur qui contient l'ordre de traitement des opérations. C'est une séquence des opérations où chacune est représentée par le numéro de la tâche. Le traitement de la tâche T_k sur M_1 est représenté par k. Tandis que sur M_2 il est représenté par $k + n$. Ainsi, la solution illustrée dans la Figure 2.4.1 est représentée par : $S = (3, 6, 1, 5, 2, 7, 4)$.

La structure du voisinage Dans certaines structures de voisinage, l'ordonnancement résultant peut ne pas satisfaire toutes les contraintes, par conséquent d'autres changements doivent être apportés pour rendre la solution réalisable. Cependant, la structure de voisinage que nous proposons satisfait toutes les contraintes, plus précisément les contraintes de cheminement des tâches.

Une solution voisine est donnée par les deux procédures de recherche locale qui sont présentées dans Recherche Locale$_1$ et Recherche Locale$_2$.

Recherche Locale$_1$ peut être décrite comme suit. D'abord une machine est sélectionnée. Ensuite, un sous-ensemble est sélectionné parmi N_1, N_2 et N_3 ou N_4 selon la machine sélectionnée. Le sous-ensemble sélectionné doit contenir au moins deux tâches, puisque deux tâches vont être aléatoirement sélectionnées de ce sous-ensemble. Si les deux tâches sont de N_1 ou N_2 alors leur ordre va être permuté sur les deux machines dans la solution actuelle. Si ce n'est pas le cas alors leur ordre va être permuté sur la machine

associée. La procédure est résumée dans Algorithme 21.

Algorithme 21 : Recherche Locale$_1$

début
  sélectionner aléatoirement un sous-ensemble parmi N_1, N_2, N_3 (resp. N_1, N_2, N_4) qui contient au moins deux tâches ;
  sélectionner aléatoirement deux tâches T_{j1} et T_{j2} du sous-ensemble sélectionné ;
  si $(T_{j1}, T_{j2} \in N_1)$ ***ou*** $(T_{j1}, T_{j2} \in N_2)$ **alors**
    permuter la position des deux tâches sur les deux machines ;
  sinon
    permuter la position des deux tâches sur leur machine ;

Recherche Locale$_2$ est résumée dans Algorithme 22 et a le fonctionnement suivant. Dans un premier temps, une machine est aléatoirement sélectionnée, ainsi qu'une tâche de $N_1 \cup N_2 \cup N_3$ ou $N_1 \cup N_2 \cup N_4$ selon la machine sélectionnée. Par la suite, on construit le sous-ensemble de tâches avec lesquelles la tâche sélectionnée est permutable. On dit que deux tâches sont permutables si la permutation de leur ordre dans la solution actuelle produit une nouvelle solution réalisable qui respecte le cheminement de toutes les tâches. Finalement, une tâche est aléatoirement sélectionnée du sous-ensemble construit. On permute l'ordre des tâches sur les deux machines si elles sont toutes les deux de N_1 ou de N_2. Sinon leur ordre sera permuté sur la machine associée.

Algorithme 22 : Recherche Locale$_2$

début
  sélectionner une machine, soit M_1 (resp. M_2) et sélectionner une tâche T_{j_1} dans $N_1 \cup N_2 \cup N_3$ (resp. $N_1 \cup N_2 \cup N_4$) ;
  définir $N = \emptyset$ et définir la permutabilité de deux tâches, selon Règle 1,2,3 et 4 ;
  pour $T_j, j = 1, \ldots, n$ **faire**
    si $(T_{j_1} \neq T_j)$ ***et*** $(T_{j_1}$ *et* T_j *sont permutables)* **alors**
      $N = N + \{T_j\}$;
  choisir aléatoirement une tâche T_{j_2} de N ;
  si $T_{j_1}, T_{j_2} \in N_1)$ ***ou*** $(T_{j_1}, T_{j_2} \in N_2$ **alors**
    permuter la position des deux tâches sur les deux machines ;
  sinon
    permuter la position des deux tâches sur leur machine ;

La permutabilité de deux tâches, dans la solution actuelle, est définie par les règles suivantes.

Règle 1 : Une tâche est permutable avec toute autre tâche du même sous-ensemble. Puisque les tâches de N_1 et N_2 sont permutées sur les deux machines alors leur cheminement reste toujours respecté et

il n'y a aucune condition quant à la permutation des tâches de N_3 et N_4.

Règle 2 : Une tâche de N_1 est permutable avec une tâche de N_3 (resp. N_4), lorsque la machine sélectionnée est M_1 (resp. M_2), si la tâche de N_3 (resp. N_4) est ordonnancée sur M_1 (resp. M_2), dans la solution actuelle, avant (resp. après) le traitement de la tâche de N_1 sur M_2 (resp. M_1).

Règle 3 : Une tâche de N_2 est permutable avec une tâche de N_3 (resp. N_4), lorsque la machine sélectionnée est M_1 (resp. M_2), si la tâche de N_3 (resp. N_4) est ordonnancée sur M_1 (resp. M_2), dans la solution actuelle, après (resp. avant) le traitement de la tâche de N_2 sur M_2 (resp. M_1).

Règle 4 : si la machine sélectionnée est M_1 (respM_2) alors une tâche de N_1 est permutable avec une tâche de N_2 sous deux conditions. La première est que le traitement de la tâche de N_1 sur M_1 est après (resp. avant) le traitement de la tâche de N_2 sur M_2. La seconde condition est que le traitement de la tâche de N_2 sur M_1 est avant (resp. après) le traitement de la tâche de N_1 sur M_2, dans la solution actuelle.

6.2.2 Algorithme Génétique

Passons, maintenant, à la description de l'algorithme génétique que nous avons développé. La population initiale de cet algorithme est composée des solutions produites par les heuristiques. Nous avons gardé la même représentation de la solution telle qu'énoncée dans la description du recuit simulé. Les procédures élémentaires de l'algorithme sont présentées, dans ce qui suit. On commence par la mutation des individus, ensuite le croisement, et enfin la sélection.

La mutation : un individu est d'abord sélectionné ensuite muté en utilisant l'une des procédures de recherche locale, résumées dans Algorithme 21 et Algorithme 22.

Le croisement : d'abord deux individus sont sélectionnés. Ensuite, à partir de chaque individu on obtient deux listes de priorité de traitement des tâches, l'une est sur M_1 et l'autre est sur M_2. On construit par la suite deux nouvelles solutions en utilisant sur une machine la liste d'un individu et sur l'autre machine la liste de l'autre individu et vice versa. La procédure est résumée dans Algorithme 23.

La sélection : Une fois que les individus sont mutés et croisés, on sélectionne de la population intermédiaire les 10 meilleures solutions pour être les individus de la nouvelle population.

Ces étapes sont répétées un certain nombre de fois qui sera défini dans la condition d'arrêt. De plus, la procédure sera interrompue si le makespan de la meilleure solution égale au maximum des bornes inférieures

Algorithme 23 : Croisement

début
- soit Sol_1 Sol_2 les deux individus sélectionnés
- construire les listes L_h^1 et L_h^2 qui sont les séquences des tâches sur M_1 et M_2 respectivement par rapport à l'individu Sol_h, $h = 1, 2$;
- soit $i' = 3 - i$ avec $i = 1, 2$ et $h' = 3 - h$;
- **pour** *h=1 à 2* **faire**
  - **tan que** $L_h^i \neq \emptyset$ *ou* $L_{h'}^{i'} \neq \emptyset$ **faire**
    - **si** *les deux machines sont libres* **alors**
      - ordonnancer la tâche avec le plus court temps de préparation parmi premières tâches disponible sur L_h^i et $L_{h'}^{i'}$;
    - **sinon**
      - soit M_i la premiére machine libre, ordonnancer la premiére tâche libre de L_h^i ;
    - enlever la tâche ordonnancée de la liste ;

Algorithme 24 : Algorithme Génétique

début
- générer la population initial en utilisant les heuristiques ;
- définir $S^{\#}$ comme étant la meilleure solution trouvée et initialiser $g = 0$;
- **tan que** $g < NbG$ *ou* $C_{\max}(S^{\#}) > LB$ **faire**
  - sélectionner la population actuelle ;
  - sélectionner quatre individu à partir de la population actuelle ;
  - muter les quatre individus en utilisant Recherche Locale$_1$ et/ou Recherche Locale$_2$;
  - appliquer le croisement sur les deux premier et deux dernier individus sélectionnés ;
  - calculer le fitness de chaque individu de la population intermédiaire ;
  - mettre à jour $S^{\#}$ et $g = g + 1$;

présentées dans la Sous-section 2.4.3 du Chapitre 2. L'algorithme génétique général est résumé dans Algorithme 24.

6.2.3 Étude expérimentale

Nous présentons dans ce qui suit l'étude expérimentale que nous avons menée afin d'analyser les performances des méthodes précédentes. Nous avons décrit, précédemment, de façon générale, les deux métaheuristiques. Cependant, nous les avons appliquées de différentes manières afin d'obtenir une meilleure combinaison des précédentes procédures.

Le recuit simulé est appliqué de trois façons différentes, nommées *SA*1, *SA*2 et *SA*12. Dans la première et

deuxième version nous avons seulement utilisé Recherche Locale$_1$ et Recherche Locale$_2$ respectivement. Dans la troisième, nous avons utilisé les deux en même temps. D'abord, on obtient la solution voisine de la solution actuelle en utilisant Recherche Locale$_1$. Ensuite, on applique Recherche Locale$_2$ à la solution actuelle mise à jour. En ce qui concerne l'algorithme génétique, nous avons quatre versions. La différence entre ces versions est la taille de la population intermédiaire et la méthode de sélection des individus à muter et à croiser. Les quatre versions sont nommées $GA1$, $GA2$, $GA3$ et $GA4$. Les algorithmes génétiques choisissent les individus à muter et à croiser de manière différente. $GA1$ sélectionne les quatre meilleures solutions et les mute en utilisant Recherche Locale$_1$ puis Recherche Locale$_2$. Ensuite, les deux premiers et les deux derniers individus sont croisés entre eux. $GA2$ et $GA3$ suivent le même schéma cependant ils utilisent seulement Recherche Locale$_1$ et Recherche Locale$_2$, respectivement, pour la mutation. $GA4$ suit le même schéma que $GA1$ sauf que les quatre individus sont sélectionnés aléatoirement.

Nous avons codé les algorithmes précédents en langage C++. Les tests sont faits sur un ordinateur personnel avec un processeur Intel(R) Core(TM) i5-5300U 2.30GHz et une RAM de 8GB. L'expérimentation est menée sur un total de 320 instances avec un nombre de tâches n variant dans $\{20, 100, 200, 1000\}$. Les temps de préparation et de traitement sont générés de la même façon que dans les tests des heuristiques. Pour un nombre de tâches n nous avons varié le paramètre c et généré 10 instances.

Les paramètres des méta-heuristiques sont résumés dans le Tableau 6.2.1.

n	20	100	200	1000
NbG	200000	150000	80000	10000
Nb	1000000	150000	100000	10000
RS paramètres	$\tau = 1.25, a = 0.005, \tau_f = 0.01$			

Tableau 6.2.1 – Paramètres des Méta-heuristiques

Nous avons utilisé dans l'évaluation des méta-heuristiques deux paramètres, qui sont la moyenne ainsi que le maximum de $f(MH) = (\frac{C_{\max}(MH)-LB}{LB})$.

Les résultats de l'étude expérimentale sont résumés dans les figures suivantes. Les Figure 6.2.1, 6.2.3, 6.2.5 et 6.2.7 représentent le maximum de $f(MH)$ des méta-heuristiques pour un nombre de tâches égal à 20, 100, 200 et 1000, respectivement. Dans les Figures 6.2.2, 6.2.4, 6.2.6 et 6.2.8 est donnée la moyenne de $f(MH)$ pour un nombre de tâches égal à 20, 100, 200 et 1000, respectivement. Nous pouvons voir que toutes les méta-heuristiques ont amélioré la qualité de la solution, cependant le taux d'amélioration est différent. Parmi les méthodes, nous distinguons trois groupes en fonction du taux d'amélioration, les méta-heuristiques d'un même groupe ont globalement les mêmes performances. Le premier groupe contient $GA1$, $GA3$ et $GA4$. Tandis que $SA2$ et $SA12$ sont dans le deuxième groupe et enfin le troisième groupe contient $GA2$ et $SA1$.

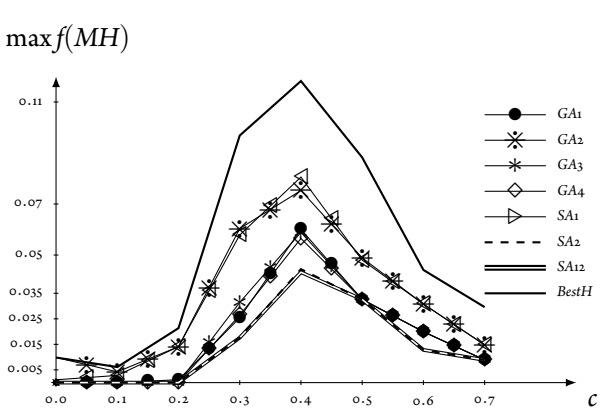


Figure 6.2.1 – max $f(MH)$, $n = 20$

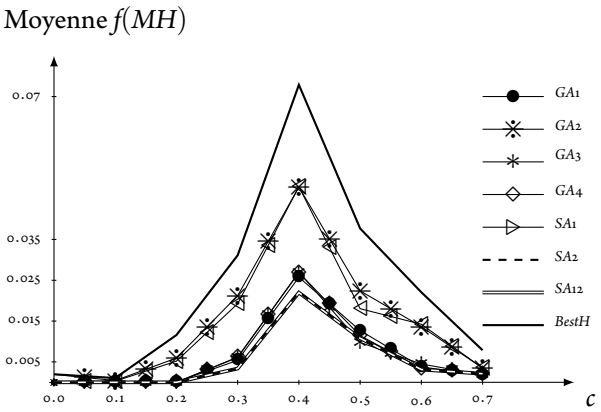


Figure 6.2.2 – Moyenne $f(MH)$, $n = 20$

Pour $n = 20$, les résultats montrent que les méthodes qui ont le plus amélioré la qualité de la solution sont $SA2$ et $SA12$. Le taux d'amélioration de $GA1$, $GA3$ et $GA4$ est légèrement différent de $SA2$ et $SA12$. Les méthodes qui ont le moins amélioré la qualité de la solution sont $GA2$ et $SA1$.

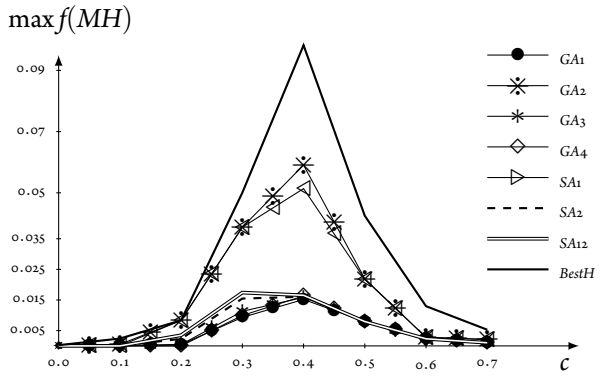


Figure 6.2.3 – max $f(MH)$, $n = 100$

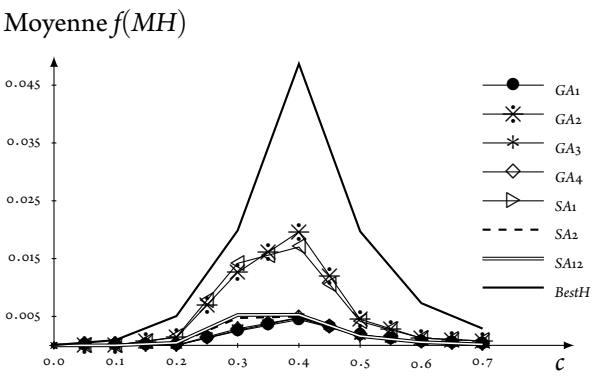


Figure 6.2.4 – Moyenne $f(MH)$, $n = 100$

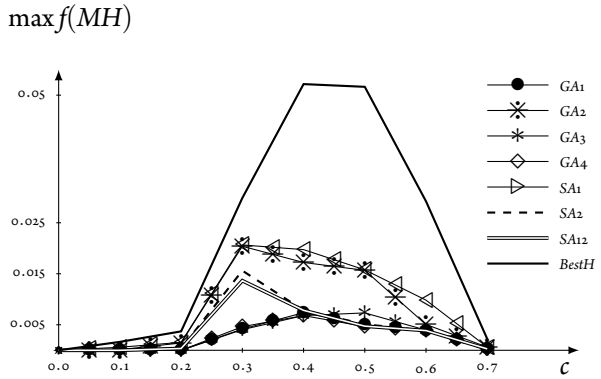


Figure 6.2.5 – max $f(MH)$, $n = 200$

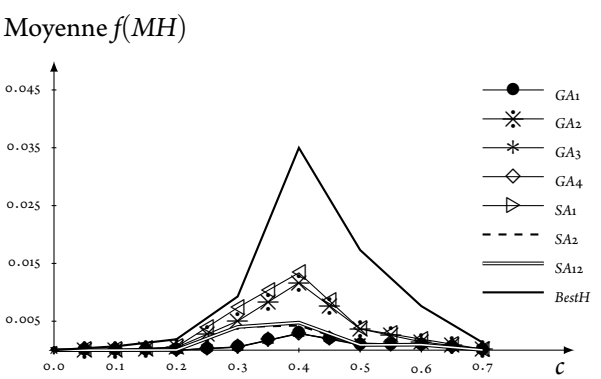


Figure 6.2.6 – Moyenne $f(MH)$, $n = 200$

Pour un nombre de tâches égal à 100 ou 200, on remarque que les méthodes qui produisent les meilleures

solutions sont *GA*1, *GA*3 et *GA*4. Par ailleurs, les solutions obtenues à partir de *SA*2 et *SA*12 sont meilleures que celles de *SA*1 et *GA*2.

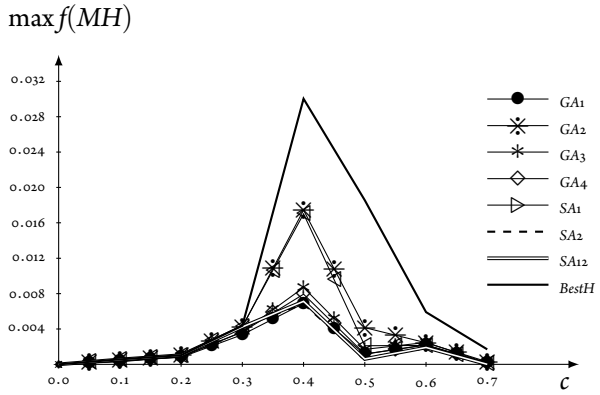


Figure 6.2.7 – max $f(MH)$, $n = 1000$

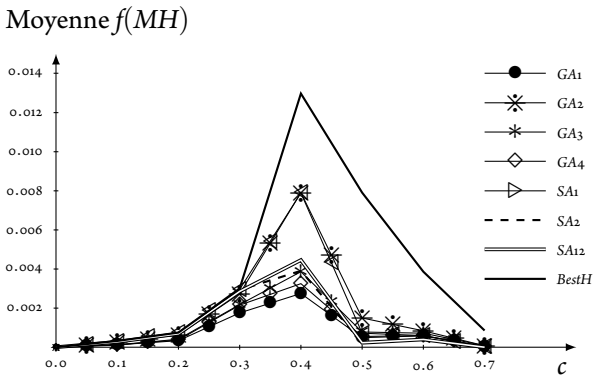


Figure 6.2.8 – Moyenne $f(MH)$, $n = 1000$

Lorsqu'on a affaire à des instances de grande taille, $n = 1000$, on remarque que l'ordre des méthodes en fonction de leur efficacité est le suivant : *GA*1, *GA*4, *GA*3, *SA*2, *SA*12 , enfin *GA*2 et *SA*1. Ainsi, pour ces instances, nous pouvons distinguer plus facilement la meilleure méthode globale, qui est *GA*1. Même si *SA*2 et *SA*12 sont meilleurs pour $c = 0.5$ et 0.6, nous remarquons que la différence entre la qualité de la solution n'est pas aussi flagrante que lorsque *GA*1 est la méthode la plus performante, *i.e.* pour $c = 0.3$ et 0.4. Par conséquent, en vue de ces observations et aussi des résultats pour $n = 100$ et 200, nous pouvons dire que globalement *GA*1 est la méta-heuristique la plus performante pour les instances de grande taille. Tandis que *SA*2 et *SA*12 sont les meilleures méthodes pour les instances avec $n = 20$.

Concernant la vitesse des méta-heuristiques précédentes, nous avons résumé dans le Tableau 6.2.2 le temps d'exécution moyen enregistré pour les instances générées. Il est évident que les versions de recuit simulé sont plus rapides que les algorithmes génétiques, ce qui est attendu. Néanmoins, *GA*1 a pu améliorer la qualité de la solution pour tous les types d'instance et pour toutes les tailles d'instance. Alors que *SA*2 et *SA*12 étaient les méthodes les plus performantes dans certains types et incapables d'améliorer la solution pour d'autres, comme nous pouvons le voir dans $n = 1000$ et $c = 0.1, 0.2, 0.3$. Nous concluons que si nous recherchons une méthode rapide et globalement efficace, nous optons pour *SA*2 ou *SA*12. Cependant, si nous recherchons une méthode qui garantit l'amélioration de la solution, la meilleure méthode est *GA*1.

Pour résumer les résultats ci-dessus, on peut conclure que les algorithmes génétiques qui utilisent les deux procédures de recherche locale sont meilleures que celles qui n'utilisent qu'une seule d'entre elles. De plus, le recuit simulé fonctionne mieux pour les instances de petite taille et aucune différence n'est observée entre la version qui utilise les deux procédures de recherche locale et celle qui utilise la deuxième procédure de recherche locale. L'algorithme génétique est meilleur pour les instances de grande taille. De plus, *GA*1 qui

sélectionne les meilleures solutions pour le croisement et la mutation est la meilleure approche globale pour les instances de grande taille.

Conclusion

Nous avons proposé plusieurs heuristiques pour la résolution du problème. Ces heuristiques sont testées via une batterie de tests. Une seule heuristique, $H_{1.1}$, s'est révélée plus efficace que les autres. Nous avons proposé un algorithme génétique (AG) et un recuit simulé (RS). Tout d'abord, nous générons la population initiale de l'AG, en utilisant les heuristiques que nous avons développés. De plus, la meilleure solution trouvée après l'application de l'heuristique est la solution initiale pour la RS. Nous avons conçu deux procédures de recherche de local qui garantissent la faisabilité de la solution produite. Ces procédures sont utilisées en tant qu'étape de mutation dans l'AG et en tant que procédures de recherche locale de RS. Le croisement dans l'AG est conçu pour combiner la séquence de tâches des deux solutions parentes en extrayant les deux listes de priorité. Ces listes sont ensuite utilisées pour construire les deux solutions enfants dont la faisabilité est garantie. Ainsi, la réparation des solutions n'est pas nécessaire.

Nous avons testé les méta-heuristiques sous plusieurs versions. Le RS est meilleur pour les instances de petite taille. De plus, il est plus efficace lorsqu'il utilise la procédure Recherche Locale$_2$. Quant à l'AG, nous avons observé qu'il est meilleur lorsque les deux procédures de recherche locale sont utilisées dans la mutation. Par ailleurs la version qui sélectionne les meilleures solutions pour la mutation et le croisement donne est plus efficace que celle qui fait cette sélection aléatoirement. Elle aussi la meilleure méthode pour les grandes instances.

c	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
n=20								
GA1	0.04	0.04	13.71	27.18	56.46	70.91	39.54	42.45
GA2	6.09	2.83	22.51	37.43	46.29	63.29	45.69	34.18
GA3	0.08	0.04	3.35	26.17	46.87	59.56	34.06	32.23
GA4	0.08	0.04	1.06	28.44	50.95	64.54	32.77	35.37
SA1	6.28	3.04	19.45	35.84	45.36	47.16	37.36	31.87
SA2	0.07	0.04	0.12	23.19	49.54	56.32	33.19	36.71
SA12	0.07	0.04	0.12	35.58	76.43	86.20	54.55	53.38
n=100								
GA1	0.18	1.06	30.58	103.13	210.17	207.09	144.15	64.24
GA2	2.62	65.06	139.67	195.16	232.13	300.14	192.56	142.46
GA3	0.23	1.29	30.55	104.98	195.50	179.91	129.39	60.41
GA4	0.24	1.43	18.09	123.11	208.77	204.78	150.12	62.07
SA1	0.15	1.67	50.78	117.98	113.36	152.51	104.74	83.69
SA2	0.11	0.41	30.21	107.05	135.21	130.12	87.87	42.99
SA12	0.11	0.49	47.39	138.68	212.10	250.86	137.01	66.15
n=200								
GA1	0.20	21.07	27.74	493.74	814.89	651.10	531.52	304.56
GA2	0.20	294.53	685.85	752.98	1061.34	763.16	784.96	505.41
GA3	0.20	32.80	37.33	428.93	767.40	703.21	497.69	317.98
GA4	0.20	23.10	32.53	482.86	861.92	637.99	521.98	265.25
SA1	0.20	41.53	184.50	198.03	186.16	189.93	159.78	77.34
SA2	0.20	7.25	34.35	214.97	225.52	211.76	159.85	49.60
SA12	0.20	6.53	70.86	343.54	351.88	329.41	237.68	75.56
n=1000								
GA1	0.84	2940.83	3091.58	2675.01	2780.30	2407.18	2060.88	1575.42
GA2	0.84	3160.96	3137.88	2690.82	2893.36	2460.82	2116.23	2075.24
GA3	0.84	3322.55	3110.66	2672.71	2756.50	2325.80	2017.09	1993.01
GA4	0.84	3126.35	3237.94	2711.20	2808.97	2369.88	2031.26	1674.61
SA1	0.84	437.23	414.99	420.51	412.76	384.44	328.86	143.19
SA2	0.84	559.64	535.52	540.17	524.90	490.30	413.92	161.59
SA12	0.84	840.43	817.91	862.50	808.76	830.70	634.46	127.48

Tableau 6.2.2 – Moyenne des temps d'exécution des méta-heuristiques (secondes)

Conclusion générale

Nous avons considéré, dans cette thèse, les problèmes d'ordonnancement d'atelier open shop et job shop à deux machines et un seul serveur. Le serveur prépare les tâches avant leur traitement sur les machines. Cependant, il est seulement nécessaire durant la phase de préparation. De plus, les préparations sont non-anticipatoires et leur durée indépendante de la séquence. Par ailleurs, le critère que nous avons considéré est la minimisation de temps fin de traitement final, noté $C_{\max}$.

Nous avons prouvé que plusieurs cas particuliers de l'open shop à deux machines et un serveur, $O2, S1|s_{ij}|C_{\max}$, sont $\mathcal{NP}$-difficiles au sens fort. Les deux premiers sont le cas avec des temps de préparation identiques et ainsi celui avec des temps de traitement identiques. Le troisième est le cas avec des temps de préparation et de traitement proportionnels. Nous avons aussi prouvé qu'un cas particulier de ce dernier, dans lequel les durées d'exécution sur une machine sont identiques, est $\mathcal{NP}$-difficile au sens ordinaire.

Nous avons également présenté des algorithmes en temps polynomial pour deux cas particuliers du problème. Le premier est le cas avec des temps de préparation identiques et des temps de traitement identiques. Le second est celui où les durées d'exécution sont identiques sur chaque machine.

Outre les résultats ci-dessus, nous avons aussi développé des heuristiques pour résoudre le problème général. Nous avons mené des tests expérimentaux pour étudier leurs performances en les comparant aux bornes inférieures que nous avons proposées. Deux heuristiques, H_6 et H_7, se sont révélées dominantes. Néaumoins, chacune l'était pour un type distinct d'instances. Nous avons donc proposé une classification des instances pour savoir quelle heuristique choisir pour résoudre le problème. Pour cela, nous avons utilisé les deux bornes inférieures, car en comparant LB_1 et LB_2 on remarque la même distinction dans les types d'instance que celle observée dans la performance de H_6 et H_7. Conséquemment, nous avons proposé une hyper-heuristique qui combine efficacement ces deux heuristiques.

Par ailleurs, nous avons proposé deux formulations de programmation linéaire mixte pour résoudre le problème $O2, S1|s_{ij}|C_{\max}$. Les deux modèles se sont avérés, à travers une vaste étude numérique, efficaces dans

le sens où l'un est meilleur que l'autre, pour différents types d'instances. Les deux modèles sont complémentaires, car l'un est efficace lorsque l'autre ne l'est pas et *vice versa*.

Concernant le job shop à deux machines avec un serveur, noté $J2, S1|s_{ij}|C_{\max}$, nous avons montré que deux cas particuliers du problème sont $\mathcal{NP}$-difficiles au sens fort. Il s'agit du cas avec des temps de préparation identiques, et du cas avec des temps de traitement identiques. De plus, nous avons proposé un algorithme polynomial pour résoudre deux cas particuliers. Le premier cas est celui avec des temps de préparation identiques et de traitement identiques. Le second est le cas avec des temps de traitement identiques et des temps de préparation longs. Nous avons également proposé plusieurs bornes inférieures pour le problème général.

Afin de résoudre $J2, S1|s_{ij}|C_{\max}$, nous avons proposé un recuit simulé et un algorithme génétique. Tout d'abord, nous avons généré la population initiale de l'algorithme génétique en utilisant des heuristiques que nous avons développées. De plus, la meilleure solution trouvée est aussi la solution initiale du recuit simulé. Nous avons conçu deux procédures de recherche locale qui garantissent la réalisabilité de la solution produite. Ces procédures sont mises en œuvre en tant qu'étape de mutation dans l'algorithme génétique et en tant que procédures de recherche locale du recuit simulé. Le croisement dans l'algorithme génétique est conçu pour combiner la séquence de tâches de deux solutions parentes en extrayant les deux listes de priorité. Ces listes sont, ensuite, utilisées pour construire les deux solutions enfants dont la réalisabilité est garantie. Ainsi, la réparation des solutions n'est pas requise.

Plusieurs versions des méta-heuristiques sont testées. Dans les versions du recuit simulé et de la mutation dans l'algorithme génétique, nous avons utilisé soit une, soit les deux procédures de recherche locale. Les meilleures versions sont celles qui utilisent les deux procédures dans les deux méta-heuristiques. De plus, le choix des individus pour la mutation et le croisement est différent dans chaque version de l'algorithme génétique. Nous avons observé que le recuit simulé fonctionne mieux pour les instances de petite taille. Cependant, l'algorithme globalement performant est l'algorithme génétique. En outre, la version de l'algorithme génétique qui sélectionne les meilleurs individus pour la mutation et le croisement s'avère plus performante par rapport à l'algorithme génétique à sélection aléatoire.

Pour de futures recherches, il serait intéressant de :

— Considérer une application modélisant un service hospitalier avec plusieurs machines et plusieurs serveurs, ainsi que des préparations spécifiques à chaque serveur.
— Étudier les problèmes d'ordonnancement modélisant d'autres types de préparation, à savoir les préparations anticipatoires et les temps de préparation indépendants de la séquence.
— Considérer le problème avec des ressources supplémentaires humaines nécessitant des pauses.

Bibliographie

[1] Abdekhodaee A.H., Wirth A., Gan H.S (2006) : Scheduling two parallel machines with a single server : The general case. *Computers and Operations Research*, 33, pp. 994-1009.

[2] Abdekhodaee A.H., Wirth A., Gan H.S (2004) : Equal processing and equal setup time cases of scheduling parallel machines with a single server.*Computers and Operations Research*, 31, pp. 1867-1889.

[3] Abdekhodaee A.H., Wirth A (2002) : Scheduling parallel machines with a single server : Some solvable cases and heuristics. *Computers and Operations Research*, 29, pp. 295-315.

[4] Allahverdi A (2015) : The third comprehensive survey on scheduling problems with setup times/costs, *European Journal of Operational Research*, 246, pp. 345-378.

[5] Allahverdi A., Soroush H.M (2008) : The significance of reducing setup times/setup costs, *European Journal of Operational Research*, 187, pp. 978-984.

[6] Allahverdi A., Ng C.T., Cheng T.C.E., Kovalyov M.Y (2008) : A survey of scheduling problems with setup times or costs, *European Journal of Operational Research*, 187, pp. 985-1032.

[7] Allahverdi A., Gupta J.N.D., Aldowaisan T (1999) : A review of scheduling research involving setup considerations, *Omega*, 27, pp. 219-239.

[8] Averbakh I., Berman O., Chernykh I (2005) : A 6/5 approximation algorithm for the two-machine routing open-shop problem on a two-node network, *European Journal of Operational Research*, 166, pp. 3-24.

[9] Babou N., Boudhar M., Rebaine D (2018) : Scheduling with Set-ups in an Open Shop Environment, *The 6th IFIP International Conference on Computational Intelligence and Its Applications, IFIP CIIA'2018*.

[10] Babou N., Boudhar M., Rebaine D (2019a) : MIP models for a two-machine open shop problem and a server with set-up times, *International Conference on Artificial Intelligence and Information Technology, ICA2IT'19*.

[11] Babou N., Boudhar M., Rebaine D (2019b) : Job shop problem with two machines and a single server for setting the jobs, *International conference Discrete Mathematics and Computer Science, DIMACOS'19*.

[12] Babou N., Rebaine D., Boudhar M (2021a) : Two-machine open shop problem with a single server and set-up time considerations. *Theoretical Computer Science*, 867, pp. 13-29.

[13] Babou N., Rebaine D., Boudhar M (2021b) : Solving the two-machine open shop problem with a single server, Manuscript submitted for publication.

[14] Babou N., Boudhar M., Rebaine D (2021c) : Two-machine job shop problem with a single server and sequence-independent non-anticipatory set-up times.

[15] Baki M.F., Vickson R.G (2004) : One-operator, two-machine open shop and flow shop problems with setup times for machines and weighted number of tardy jobs objective, *Optimization Methods and Software*, 19, pp. 165-178.

[16] Baki M.F., Vickson R.G (2003) : One-operator, two-machine open shop and flow shop scheduling with setup times for machines and maximum lateness objective, *INFOR : Information Systems and Operational Research*, 41, pp. 301-319.

[17] Bellman R (1957) : Dynamic Programming. *Princeton University Press*.

[18] Blazewicz J.K., Ecker E., Pech G., Schmidt G., Weglarz J (2001) Scheduling computer and manufacturing processes, *Springer*.

[19] Brucker P., Knust S., Wang G (2005) : Complexity results for flow-shop problems with a single server. *European Journal of Operational Research*, 165, pp. 398-407.

[20] Brucker P., Dhaenens-Flipo C., Knust S., Kravchenko S.A., Werner F (2002) : Complexity results for parallel machine problems with a single server. *Journal of Scheduling*, 5, pp. 429-457.

[21] Cheng T.C.E., Wang G., Sriskandarajah C (1999) : One operator - two-machine flowshop scheduling with setup and dismounting times. *Computers and Operations Research*, 26, pp. 715-730.

[22] Cheng T.C.E., Kovalyov M.Y (2003) : Scheduling a Single Server in a Two-machine Flow Shop. *Computing*, 70, pp. 167-180.

[23] Gan H. S., Wirth A., Abdekhodaee A (2012) : A Branch-and-price Algorithm for the General Case of Scheduling Parallel Machines with a Single Server, *Computers and Operations Research*, 39, pp. 2242-2247.

[24] Garey M.R., Johnson D.S (1979) : *Computers and Intractability : A Guide to the Theory of NP-Completeness*, A Series of Books in the Mathematical Sciences. San Francisco, California : W.H. Freeman and Co.

[25] Gharbi A., Ladhari T., Msakni M.K., Serairi M (2013) : The two machine flowshop problem with sequence-independent setup times : New lower bounding strategies, *European Journal of Operational Research*, 231, pp. 69-78.

[26] Glass C.A., Shafransky Y.M., Strusevich V.A (2000) : Scheduling for Parallel Dedicated Machines with a Single Server, *Naval Research Logistics*, 47, pp. 304-328.

[27] Gonzalez T., Sahni S (1976) : Open shop scheduling to minimize finish time, *Journal of the ACM*, 23, pp. 665-679.

[28] Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G (1979) : Optimization and approximation in Deterministic Sequencing : a Survey, *Proceeding of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of Discrete Optimization Symposium. Elsevier*, 5, pp.287-326.

[29] Hall N.G., Potts C.N., Sriskandarajah C (2000) : Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102, pp. 223-243.

[30] Hasani K., Kravchenko S.A., Werner F (2016) : Minimizing the makespan for the two-machine scheduling problem with a single server : Two algorithms for very large instances, *Engineering Optimization*, 48(1), pp. 173-183.

[31] Hasani K., Kravchenko S.A., Werner F (2014) : Simulated annealing and genetic algorithms for the two-machine scheduling problem with a single server, *International Journal of Production Research*, 52(13), pp. 3778-3792.

[32] Hasani K., Kravchenko S.A., Werner F (2013) : Block Models for Scheduling Jobs on Two Parallel Machines with a Single Server, *Computers and Operations Research*, 41, pp. 94-97.

[33] Jackson J.R (1956) : An extension of Johnson's results on job lot scheduling, *Naval Research Logistics Quarterly*, 3, pp. 201-203.

[34] Johnson S.M (1954) : Optimal Two-and Three-Stage Production Schedules with Setup Times, *Naval Research Logistics Quarterly*, 1, pp. 61-66.

[35] Kim M.Y., Lee Y.H (2012) : MIP models and hybrid algorithm for minimizing the makespan of parallel machines scheduling problem with a single server.*Computers and Operations Research*, 39, pp. 2457-2468.

[36] Kravchenko S.A., Werner F (2001) : A heuristic algorithm for minimizing mean flow time with unit setups. *Information Processing Letters*, 79, pp. 291-296.

[37] Kravchenko S.A., Werner F (1998) : Scheduling on parallel machines with a single and multiple servers. *Otto-von-Guericke-Universitat Magdeburg, Preprint*, 30/98, pp. 1-18.

[38] Kravchenko S.A., Werner F (1997) : Parallel machine scheduling problems with a single server, *Mathematical and Computer Modelling*, 26, pp. 1-11.

[39] Koulamas C.P (1996) : Scheduling two parallel semiautomatic machines to minimize machine interference, *Computers and Operations Research*, 23, pp. 945-956.

[40] Labbi W., Boudhar M., Oulamara A (2014) : Scheduling two identical parallel machines with preparation constraints, *International Journal of Production Research*, 55(6), pp. 1531-1548.

[41] Lim A., Rodrigues B., Wang C (2006) : Two-machine flow shop problems with a single server, *Journal of Scheduling*, 9, pp. 515-543.

[42] Ling S., Xue-guang C (2011) : On a Two-machine Flow-shop Scheduling Problem with a Single Server and Unit Processing Times, *Journal of Applied Mathematics and Bioinformatics*, 1(2), pp. 33-38.

[43] Ling S., Xue-guang C (2012) : The Two-machine Flow-shop Scheduling Problem with a Single Server and Unit Server Times, *Journal of Informatics and Mathematical Sciences*, 4(1), pp. 123-127.

[44] Low C., Yeh Y (2009) : Genetic algorithm based heuristics for an open shop scheduling problem with setup, processing, and removal times separated, *Robotics and Computer-Integrated Manufacturing*, 25(2), pp. 314-322.

[45] Mosheiov G., Oron D (2008) : Open-shop batch scheduling with identical jobs, *European Journal of Operational Research*, 187, pp. 1282-1292.

[46] Naboureh K., Safari E (2016) : Integrating the sequence dependent setup time open shop problem and preventive maintenance policies, *Decision Science Letters*, 5, pp. 535-550.

[47] Naderi B., Ghomi S.M.T., Fatemi Aminnayeri M., Zandieh M (2011) : Modeling and scheduling open shops with sequence-dependent setup times to minimize total completion time, *International Journal of Advanced Manufacturing Technology*, 53(5-8), pp. 751-760.

[48] Oulamara A., Rebaine D., Serairi M (2013) : Scheduling the two-machine open shop problem under resource constraints for setting the jobs, *Annals of Operations Research*, 356, pp. 211-333.

[49] Pinedo M.L (2008) : *Scheduling : Theory, Algorithms, and Systems*, Springer Publishing Company, 3rd Edition.

[50] Pinedo M.L., Schrage L (1982) : Stochastic shop scheduling : A survey, in Dempster MA. (Ed.) : Deterministic and stochastic scheduling,*Proceedings of the NATO Advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems*, Durham, England, pp. 81-96.

[51] Roshanaei V., Seyyed Esfehani M.M., Zandieh M (2010) : Integrating non-preemptive open shops scheduling with sequence-dependent setup times using advanced metaheuristics, *Expert Systems with Applications*, 37(1), pp. 259-266.

[52] Strusevich V.A (2000) : Group technology approach to the open shop scheduling problem with batch setup times, *Operations Research Letters*, 26, pp. 181-192.

[53] Su L.H., Lee Y.Y (2008) : The two-machine flowshop no-wait scheduling problem with a single server to minimize the total completion time, *Computers and Operations Research*, 35, pp. 2952-2963.

[54] Wagelmans A.P.M., Gerodimos A.E (2000) : Improved dynamic programs for some batching problems involving maximum lateness criterion, *Operations Research Letters*, 27, pp. 109-118.

[55] Wang G., Cheng T.C.E (2001) : An approximation algorithm for parallel machine scheduling with a common server, *Journal of the Operational Research Society*, 52(2), pp. 234-237.

[56] Werner F., Kravchenko S. A (2010) : Scheduling with Multiple Servers, *Automation and Remote Control*, Vol. 71(10), pp. 2109-2121.