Ordonnancement d'atelier de type multi-agents

Présenté par :

AZERINE Abdennour (**)

Faculté de Mathématiques

Département de Recherche Opérationnelle

Résumé : Les recherches menées dans cette thèse appartiennent au domaine de l'optimisation combinatoire. En particulier, nous nous intéressons au problème d'ordonnancement multi-agents. Plus précisément, nous concentrons notre attention sur le modèle avec des agents en compétition dans les environnements flow shop, job shop et open shop.

Le problème considéré peut être décrit comme suit : Étant donné un ensemble de n tâches $J = \{J_1, J_2, \ldots, J_n\}$ à traiter sur un ensemble M de m machines $M = \{M_1, M_2, \ldots, M_m\}$, si toutes les tâches ont les mêmes routes de traitement sur toutes les machines, le problème est appelé flow shop. Si chaque tâche a sa propre route de traitement, on parle de job shop, le problème avec des routes de tâches libres est appelé open shop. Les tâches peuvent appartenir à différents utilisateurs appelés agents. Chaque agent a un ensemble de tâches à traiter sur un ensemble de machines, la route de traitement est spécifiée à l'avance pour les problèmes de flow shop et de job shop. Chaque agent possède son ensemble de tâches et souhaite minimiser sa fonction de coût, qui ne dépend que des délais d'exécution de ses tâches. Lors de l'ordonnancement des tâches, différents critères doivent être satisfaits.

L'objectif de cette thèse est de développer des algorithmes pour résoudre différents ordonnancements multi-agents avec des machines dédiées, d'identifier les limites entre les cas polynomiaux et les cas $\mathcal{NP}$-difficile.

Mots clés : Multiagent ; ordonnancement ; scheduling ; flow shop ; job shop ; open shop ; makespan ; no-wait.

*Thèse de Doctorat LMD.

**Directeur de Thèse : BOUDHAR Mourad (Professeur, USTHB),

***Co-Directeur de Thèse : REBAINE Djamal (Professeur, UQAC, Canada).

Introduction Générale

L'ordonnancement (en anglais "Scheduling") s'intéresse à organiser la réalisation d'un ensemble de tâches planifié dans un temps en respectant des contraintes, pour optimiser un ou plusieurs critères et pour satisfaire au mieux les objectifs, elle détermine quand la tâche peut être réalisée, et qui doit la réaliser. Enfin, elle fait en sorte de fournir et de gérer l'ensemble de ces moyens, donc dans un problème d'ordonnancement on vise à répondre aux questions suivantes : Qui fait ? Quoi ? Et quand ? Et avec combien de ressource ?

Dans de nombreux problèmes de la vie réelle, les ressources peuvent être partagées entre les utilisateurs. Les utilisateurs sont appelés "agents",

Les études sur l'ordonnancement multi-agents concernent de nombreux domaines et industries. Plusieurs exemples et applications ont été publiés dans la littérature, comme : la gestion des chemins de fer, les réseaux de communication, les problèmes de réordonnancement, etc. Le problème d'ordonnancement multi-agents a été introduit par Agnetis et al. (2004). Ils ont prouvé que les problèmes de flow shop et d'open shop à deux machines avec deux agents concurrents et les critères de makespan pour chaque agent sont $\mathcal{NP}$-difficile pour l'approche ϵ-contrainte.

Dans cette thèse, nous considérons un problème d'ordonnancement multi-agents dans un environnement de flow shop, job shop ou open shop, dans lequel nous avons k agents. Chaque agent a un ensemble de tâches à traiter sur un ensemble de machines, l'itinéraire de traitement est donné à l'avance pour les les problèmes de flow shop et de job shop. Chaque agent a son propre ensemble de tâches et veut minimiser sa fonction objectif, qui dépend des temps d'achèvement (fin de traitement) de ses propres tâches uniquement. Lors de l'ordonnancement des tâches, différents critères doivent être satisfaits.

L'objectif de cette thèse est de développer des algorithmes pour résoudre différents problèmes d'ordonnancement à deux agents, d'identifier les limites entre les cas polynomiaux et les cas $\mathcal{NP}$-difficile. La thèse contient les chapitres suivants :

- Chapitre 1 : Dans ce chapitre nous citons les définitions et quelques notions nécessaires permettant la compréhension du problème.
- Chapitre 2 : Ce chapitre est consacré à l'état de l'art.
- Chapitre 3 : traitement du premier cas de flow shop avec deux objectifs differents, la complexité de quelque sous problèmes est étudié. Des modèles mathématiques, une méthode basé sur le branch-and-bound et des méthodes approchées (heuristiques et méta-heuristique hybride) sont présenté pour résoudre le problème.
- Chapitre 4 : présentation du deuxième problème avec no-wait constraint, la complexité de problème est étudié et des sous-cas sous-cas polynomiaux sont présenté. Des méthodes exactes et approchées sont proposés pour résoudre le problème.
- Chapitre 5 : Il est consacré pour étudier le job shop avec deux agents et makespan comme une fonction objectif pour les deux agents. Il contient la même structure que le chapitre précédent.
- Chapitre 6 : Une conclusion termine cette thèse avec un résumé des principaux résultats présentés. Nous proposons aussi des perspectives à ce travail.

Chapitre

1 Multi-agent et théorie de l'ordonnancement

1.1 Introduction

Au cours des quinze dernières années, les problèmes d'ordonnancement multi-agents ont fait l'objet d'une attention accrue dans la littérature. l'importance de l'ordonnancement dans l'amélioration de la performance des systèmes de fabrication et de la concurrence et de la compétitives dans les industries manufacturières.

Nous considérons le scénario où il y a plus d'un utilisateur (agent), chaque utilisateur (agent) a son sous-ensemble de travaux à traiter. Son sous-ensemble de travaux doit être traité sur plusieurs machines partagées configurées en tant que flow shop, job shop ou open shop. Au lieu d'une fonction objective unique pour tous les travaux, nous considérons que chaque agent a une fonction objective qui ne dépend que des temps d'achèvement de ses travaux uniquement. Lors de l'ordonnancement des tâches, différents critères doivent être satisfaits. l'exécution des tâches d'un agent minimisera cette fonction objective, ce qui empêchera les autres utilisateurs d'utiliser ces ressources. les autres utilisateurs ne peuvent pas utiliser ces ressources, ce qui peut augmenter la valeur des fonctions objectives pour les autres agents.

Avant de nous plonger dans l'analyse de divers scénarios de problèmes d'ordonnancement multi-agents, nous présentons d'abord, dans le reste de ce chapitre, les hypothèses, la notation, le schéma de classification des problèmes qui aide à organiser le matériel qui sera présenté dans les chapitres suivants, un concept de modèles d'agents en compétition, les avantages potentiels des problèmes multi-agents, ainsi qu'une brève perspective historique et certains domaines d'application de l'ordonnancement multi-agents. Pour plus de détails sur la théorie de l'ordonnancement, nous pouvons citer les livres suivants : [8, 26]. Pour le problème du flow shop, nous recommandons [9] et pour le problème multi-agents [2].

Dans notre thèses nous considérons les machines comme des ressource à utiliser pour exécuter les tâches. Une tâche peut appartenir à un ou plusieurs utilisateurs, que nous appelons agents. Nous considérons le problème de l'ordonnancement de n tâches sur m machines avec K agents. Chaque agent k dispose d'un ensemble de tâches J^k et d'une fonction objective ou d'un critère d'optimalité qui dépend généralement des dates de fin d'exécution de ses travaux uniquement. La valeur de la fonction objective f^k de la solution π est notée $f^k(\pi)$ pour $k = 1, \ldots, K$. Dans le cas de deux agents, nous désignons les agents par X, le premier agent par A et le deuxième par B.

Les machines spécialisées ou dédiées (dedicated machines)

Ces machines sont conçues pour effectuer des opérations spécifiques. On distingue trois systèmes ou modes de traitement dans le cas des machines spécialisées :

Ateliers à cheminement unique (flow shop) : dans un système flow shop, les tâches doivent être traitées par toutes les machines dans un même ordre. le nombre d'opérations de chaque tâche est égal au nombre de machines.

Ateliers à cheminement libre (open shop) : dans un systeme open shop, les tâches doivent être traitées par toutes les machines mais l'ordre du traitement n'est pas nécessairement le même pour toutes les tâches, l'ordre de traitement est quelconque.

Ateliers à cheminement multiples (job shop) : dans un système job shop, le sous-ensemble de machines devant traiter une tâche et l'ordre du traitement sont arbitraires, mais doivent être spécifiés à l'avance.

1.2 Classification de l'ordonnancement

Afin de maintenir la clarté, la simplicité et la correction dans la description des problèmes, nous utilisons une notation à trois champs séparés par des barres $\alpha|\beta|\gamma$ qui a été introduite par [13].

Champs α Il décrit l'environnement de la machine, il spécifie le type et le nombre de machines, ainsi le que le type d'atelier.

Champ β spécifie les caractéristiques des tâches et les contraintes. On peut avoir plusieurs contraintes dans ce champ, $\beta = \beta_1\beta_2\beta_3\beta_4\beta_5\beta_6 \ldots$.

Champs γ nous entrons les critères à minimiser habituellement.

1.2.1 Classification des problèmes d'ordonnancement multi-agents

Afin d'étudier la complexité des problèmes d'ordonnancement multi-agents, nous devons classer les problèmes en fonction de l'intersection des jobs des agents, nous présentons cette classification pour le cas avec deux agents, nous avons quatre catégories comme suit :

Multi-critère Les agents partagent le même ensemble de tâches et de ressources, c'est-à-dire : $J = J_A = J_B$. Cependant, chaque agent a une fonction objective différente.

Compétition (CO) Les deux agents ne partagent que des ressources et sont en concurrence pour exécuter leur ensemble de tâches. Chaque tâche n'appartient qu'à un seul agent, ce qui signifie que les deux ensembles sont indépendants $J_A \cap J_B = \phi$ et $J_A \cup J_B = J$. Il a été introduit par [4] et [1] ce dernier a dénoté ce problème initialement par "multi-agents". On peut aussi trouvez des autre appellation comme "Interférants" dans la littérature.

Interférant (IN) Dans ce cas, l'ensemble des tâches d'un agent est inclus dans l'ensemble du deuxième agent. Ceux-ci : $J_B \subset J_A = J$. Pour le scénario avec K agents, nous avons la relation suivante : $J_K \subset \ldots J_2 \subset J_1 = J$. Les agents sont en concurrence même dans le cas où leurs critères sont de même nature.

Non-disjoint (ND) C'est le cas le plus général, chaque tâche appartient au moins à un agent, ce qui signifie qu'il peut y avoir des tâches communes entre les ensembles, si une tâche appartient à un agent alors elle a un poids et une date d'échéance uniques. Cependant, si un travail appartient à deux agents, il peut ou non avoir des poids et des dates d'échéance différents selon le propriétaire de cette tâche.

Le problème d'ordonnancement multi-agent est le problème le plus général des problèmes d'ordonnancement multi-critères, nous utiliserons donc le même contexte et les mêmes notions que la théorie d'ordonnancement multi-critères avec des notations supplémentaires dans certains

cas. Pour plus d'informations sur l'ordonnancement multi-critères, nous renvoyons les lecteurs aux travaux de [30].

Un ordonnancement consiste à déterminer la date de début et de fin des tâches, nous les appelons aussi jobs. Chaque tâche a besoin de ressources, de contraintes supplémentaires et de temps pour être exécutée, un exemple de contraintes supplémentaires est le temps d'arrivée, sans attente entre les opérations d'une même tâche.

De nombreuses approches ont été proposées dans la littérature pour résoudre les problèmes multi-agents, en particulier pour le cas multi-critères. Par conséquent, nous donnons par la suite quelques définitions, puis nous proposons quelques approches qui ont été proposées pour les problèmes d'ordonnancement multi-agents.

1.3 Conclusion

Dans ce chapitre, nous avons rappelé les concepts de base des problèmes d'ordonnancement. Nous avons présenté par la suite les notions de la théorie de complexité, et les méthode de résolution d'un problème d'ordonnancement.

La dernière partie de ce chapitre a été consacré au problèmes d'ordonnancement multi-agents, leur classification et les méthodes utilisées pour résoudre les problèmes d'optimisation multi-critères.

Chapitre

2 Ordonnancement multi-agent : état de l'art

2.1 Introduction

L'ordonnancement multi-agents fait référence aux problèmes dans lesquels des ressources communes sont partagées par deux ou plusieurs agents. L'objectif est de construire un ordonnancement, qui minimise la valeur de la fonction objectif de chaque agent. Dans les problèmes d'ordonnancement que nous abordons dans notre thèses, les tâches sont traités sans préemption, à partir du temps zéro. Nous supposons que chaque tâche est traitée par au plus une machine à la fois, et que chaque machine peut traiter une tâche à la fois.

Dans la plupart des problèmes d'ordonnancement classiques, on suppose implicitement que l'ensemble des tâches est traité par un seul agent, donc associé à un seul critère à optimiser. En pratique, cependant, cette hypothèse n'est pas toujours justifiée. En effet, la rareté des ressources a objectivement conduit les décideurs à partager ces ressources entre plusieurs utilisateurs, chacun étant associé à son propre ensemble de jobs, donc à son propre critère à optimiser.

Nous nous concentrons dans ce chapitre au revue de la littérature des problèmes d'ordonnancement multi-agent avec des agents en compétition. Plusieurs agents disjoints sont définis, chacun minimisant un critère relatif à son propre ensemble de tâches. Dans un premier lieu nous étudions la complexité des problèmes avec un seul agent pour les problèmes sur des machines dédiées. Pour bien comprendre et trouver la complexité des problèmes d'atelier, nous donnons un aperçu de la complexité des problèmes multi-agent avec une seule machine, nous rappelons brièvement la complexité des problèmes multi-agent sur une seule machine. Pour terminer ce chapitre,, nous présentons la revue de la littérature des problèmes d'ordonnancement d'atelier multi-agents et enfin nous fournissons quelques applications du monde réel.

D'autre part, il a été démontré que le problème du flow shop à deux machines avec une contrainte de non-attente est polynomialement solvable en utilisant un cas spécial du problème TSP par [11]. La complexité de calcul est améliorée pour être de $O(n \log n)$ par [31]. Les temps de traitement de chaque tâche sur chaque machine sont supposés être strictement positifs. Permettre des temps de traitement nuls sur chaque machine rend le problème $\mathcal{NP}$-difficile (voir [29]).

2.2 Ordonnancement avec machine dédiées et un seul agent

Dans cette section, nous présentons la complexité de base du problème de shop avec un seul agent. Il est connu que le flow shop à deux machines et le job shop à deux opérations, ont été prouvés être polynomialement solvables en $O(n \log n)$ respectivement avec le makespan comme fonction objectif, tandis que le open shop à deux machines peut être résolu en temps linéaire.

2.3 Problèmes d'ordonnancement multi-agent avec des machines dédiées

[1] ont prouvé que le problème de flow shop et open à deux machines avec des critères de makespan pour chaque agent est $\mathcal{NP}$-difficile pour l'approche ϵ-contrainte en utilisant une réduction à partir du problème de partition. Plus tard, le problème de flow shop a été montré comme étant $\mathcal{NP}$-difficile au sens faible par [19], ils ont présenté un algorithme en temps pseudo-polynomial fonctionnant en $O((n_A + n_B)P^4)$ avec P est la somme des temps de traitement de toutes les tâches sur les deux machines, et un algorithme d'approximation avec un rapport de $(1 + \epsilon)$ en relaxant la contrainte à $C^B_{max} \leq (1 + \epsilon)Q$ pour tout $\epsilon > 0$.

Divers autres objectifs ont été étudiés dans la littérature, [17] a considéré le problème du flow shop à deux machines pour minimiser la somme des dates de fin de traitement des tâches de premier agent avec zéro tâche en retard comme une borne supérieure pour le second agent. Ils ont développé un algorithme branch and bound qui peut donner une solution optimale pour la plupart des instances avec 20 tâches en un temps raisonnable et différents adaptation de l'algorithme de recuit simulé. Tandis que, [21] étendent certains problèmes d'une seule machine à deux agents à leur flowshop proportionnel correspondant. Ils ont considéré trois objectifs pour le premier agent : minimiser le coût maximum de toutes les tâches, la somme des dates de fin de traitement, et le nombre minimum de tâches en retard tout en bornant la fonction de coût du second agent par une valeur fixe Q.

2.4 Application

Les exemples suivants montrent à quel point l'ordonnancement multi-agents est fondamental :

Réseaux de communication [24] : a décrit un ensemble de problèmes qui se produisent dans les réseaux à commutation de paquets à services intégrés tels que l'ATM (Asynchronous Transfer Mode), les tâches sont généralement divisées en deux types ou plus, le premier est celui des tâches qui doivent être traitées et transformés complètement, nous fixons un objectif comme les temps d'achèvement totaux avec version et délais pondérés ou non pondérés. Le deuxième type pour les données qui prennent trop de temps et peuvent être perdues comme les photos, les vidéos et le service vocal, nous fixons l'objectif comme le nombre total de tâches en retard. Les données informatiques sont divisées en paquets, les paquets sont mis en file d'attente dans un tampon au point d'accès du réseau dans lequel ils sont ordonnés par un algorithme d'ordonnancement algorithme d'ordonnancement pour envoyer les paquets de la source à la destination.

Problèmes de ré-ordonnancement [25] : se sont concentrés sur les problèmes de ré-ordonnancement où certaines tâches ne peuvent pas être terminées dans une journée en raison de la production la demande dépasse la charge de travail du jour, ces tâches doivent être traitées le jour suivant avec une priorité plus élevée, ce qui peut être fait en fixant une date limite et une pondération. et pondérée, tandis que les autres nouvelles tâches peuvent avoir un objectif comme le makespan ou le temps d'écoulement total.

2.5 Conclusion

Dans ce chapitre, on a présenté l'état de l'art sur les problème d'ordonnancement avec des machines dédiées et pour les problèmes avec une seule machine multi-agent. Dans les chapitre suivants, nous allons étudier les problèmes avec flow shop, job shop et open shop avec deux agents et des contraintes différents.

Chapitre

3 Flow shop à deux machines et deux agents

3.1 Introduction

Dans cet article, nous étudions l'ordonnancement du flow shop à deux machines avec deux agents concurrents. La complexité des cas particuliers est étudiée en ce qui concerne le makespan et le temps d'achèvement total. La deuxième partie concerne la minimisation de la combinaison linéaire du retard total et du nombre de tâches en retard. Nous présentons trois formulations de programmation en nombres entiers mixtes et un algorithme de branch and bound ainsi qu'une heuristique et une méta-heuristique. Une analyse computationnelle est ensuite menée. Les résultats révèlent que l'algorithme de branch and bound surpasse les modèles de programmation en nombres entiers mixtes, tandis que les solutions approximatives de haute qualité sont produites par la méta-heuristique proposée.

3.2 Questions relatives à la complexité

Théorème 1. $F2|M_1 \mapsto M_2, M_2 \mapsto M_1, p_{ij} = p_j|C^A_{max} + C^B_{max}$ *est* $\mathcal{NP}$*-difficile.*

Le résultat ci-dessus peut être étendu au problème du job shop avec deux opérations.

Corollaire 1. $J2|p_{ij} = p_j|C^A_{max} + C^B_{max}$ *est* $\mathcal{NP}$*-difficile.*

Théorème 2. $F2|M_1 \mapsto M_2, M_2 \mapsto M_1, p_{ij} = p_j|C^A_{max} + \sum_{j \in J_B} C^B_j$ *est* $\mathcal{NP}$*-difficile.*

Théorème 3. $F2|M_1 \mapsto M_2, M_2, p^A_{ij} = p_j|C^A_{max} + C^B_{max}$ *est* $\mathcal{NP}$*-difficile.*

Corollaire 2. *Les deux problèmes* $F3|M_1 \mapsto M_2, M_2 \mapsto M_3|C^A_{max}+C^B_{max}$ *et* $F3|M_1 \mapsto M_2, M_3 \mapsto M_2|C^A_{max} + C^B_{max}$ *sont* $\mathcal{NP}$*-difficile même si* $p_{ij} = p_j$*.*

Théorème 4. $F2|M_1 \mapsto M_2, M_2, p^A_{ij} = p^A_j|\sum_j C^A_j + C^B_{max}$ *est* $\mathcal{NP}$*-difficile.*

Théorème 5. $F2|M_1 \longmapsto M_2, M_1|C^A_{max} + C^B_{max}$ *est* $\mathcal{NP}$*-difficile.*

Corollaire 3. $F3|M_1 \longmapsto M_2, M_1 \longmapsto M_3|C^A_{max} + C^B_{max}$ *et* $F3|M_1 \longmapsto M_2, M_3 \longmapsto M_1|C^A_{max} + C^B_{max}$ *sont* $\mathcal{NP}$*-difficile.*

Théorème 6. $F2|M_1 \longmapsto M_2, M_1|C^A_{max} + \sum_j C^B_j$ *est* $\mathcal{NP}$*-difficile.*

Dans les prochaines sections, nous aborderons le problème du flow shop à deux agents avec deux objectifs différents, ce problème peut être formulé comme suit : Deux agents A et B sont en compétition pour traiter leurs tâches sur deux machines partagées, en commençant par M_1 puis M_2. L'objectif du premier agent est de minimiser la somme des retards tandis que l'objectif du second agent est de minimiser le nombre de tâches en retard, nous utiliserons la combinaison

linéaire comme approche pour trouver une solution. Ce problème est donné par : $F2||\alpha \times \sum_{j \in J_A} T_j^A + \beta \times \sum_{j \in J_B} U_j^B$ où $0 \leq \alpha \leq 1$ et $\alpha + \beta = 1$ ou $F2||\lambda \times \sum_{j \in J_A} T_j^A + (1-\lambda) \times \sum_{j \in J_B} U_j^B$ où $0 \leq \lambda \leq 1$.

Afin de trouver une solution optimale à ce problème, nous proposons des modèles de programmation mixte en nombres entiers et un algorithme basé sur le branch-and-bound.

3.3 Modèles de programmation en nombres entiers

Dans cette section, nous proposons trois formulations mathématiques du problème étudié *v.z.* : une formulation indexée sur le temps et deux formulations basées sur la variable de précédence entre les tâches.

3.4 Algorithme de séparation et évaluation

Rappelons qu'un algorithme de branch-and-bound consiste à décomposer le problème étudié en sous-problèmes successivement plus petits, à calculer les limites de la fonction objectif associée à chaque sous-problème, et à les utiliser pour écarter certains de ces sous-problèmes de toute considération ultérieure. L'algorithme se termine lorsque chaque sous-problème a soit produit une solution réalisable, soit démontré qu'il ne contient pas de meilleure solution que celle déjà en main. La meilleure solution trouvée à la fin de l'algorithme est l'optimum global.

L'application d'un algorithme de branch-and-bound nécessite de spécifier des ingrédients tels que des bornes inférieures, des règles de dominance, des heuristiques de borne supérieure et des règles de branchement, comme décrit dans les sous-sections suivantes. Il convient de mentionner que l'efficacité des algorithmes de branch-and-bound repose non seulement sur la rigueur de ces ingrédients, mais aussi sur le temps d'exécution nécessaire à leur calcul.

3.5 Méta-heuristiques

Puisque le problème est fortement $\mathcal{NP}$-difficile, l'approche heuristique est bien justifiée. Dans cette section, nous proposons deux méta-heuristiques.

3.5.1 Recherche Tabou

La recherche Tabou (TS) est une méta-heuristique basée sur une structure de voisinage. TS commence avec une solution initiale. Dans notre cas, nous choisissons la meilleure solution trouvée par l'heuristique. À chaque itération, nous choisissons la meilleure solution du voisinage. Une solution de voisinage peut être générée par différentes méthodes : échange, insertion, etc. Pour plus de détails, voir par exemple [22].

3.5.2 Algorithme génétique

L'algorithme génétique est une méta-heuristique basée sur la population, inspirée de l'évolution naturelle. L'algorithme commence avec une population générée de manière aléatoire. À chaque itération, de nouveaux parents sont sélectionnés à l'aide d'un mécanisme de sélection pour produire de nouveaux enfants à l'aide d'opérateurs de croisement et de mutation. Ce processus se poursuit jusqu'à ce que certains critères prédéfinis soient atteints.

3.6 Expériences computationnelles

Afin d'évaluer les performances de l'algorithme proposé, nous avons réalisé des expériences numériques concernant l'efficacité des modèles mathématiques et de l'algorithme de branch-and-bound en termes de temps CPU. Les modèles mathématiques ont été codés à l'aide du package Cplex. Nous avons également étudié la qualité des solutions produites par les algorithmes méta-heuristiques. L'algorithme de branch-and-bound proposé et les méta-heuristiques ont été codés en C++. Les expériences de calcul ont été réalisées sur un PC avec Intel(R) Core(TM) i7-2670QM, CPU 2.20 GHZ et 8.00 GB RAM sur le système d'exploitation Windows 7.

3.6.1 Performances des méthodes exactes

Dans nos expériences de calcul, nous générons les instances de manière aléatoire. Des temps de traitement entiers ont été générés à partir d'une distribution uniforme $[1, 25]$ et $[25, 100]$, les dates d'échéus ont été générées à partir d'une autre distribution uniforme sur les entiers compris entre $T(1 - \tau - R/2)$ et $T(1 - \tau + R/2)$ où R est un paramètre appelé plage de dates d'échéu, et τ est appelé facteur de retard. La combinaison de (τ, R) a pris les valeurs de $(0, 25, 0, 25)$, $(0, 25, 0, 50)$, $(0, 25, 0, 75)$, $(0, 50, 0, 25)$, $(0, 50, 0, 50)$ et $(0, 50, 0, 75)$. La valeur T est la somme des temps de traitement sur la machine M_2 et du temps de traitement minimum de toutes les tâches sur la machine M_1.

3.6.2 Performance des méthodes approchées

Pour des instances de petite taille avec des tâches de 15, nous avons comparé expérimentalement l'efficacité des solutions générées par les procédures méta-heuristiques avec les solutions optimales produites par la procédure branch-and-bound, les données sont générées selon deux intervalles de temps de traitement différents ($[1, 25]$ et $[25, 100]$) et un intervalle de dates d'échéance (La combinaison de τ et R avec $\tau \in \{0.25, 0.5\}$ et $R \in \{0.25, 0.5, 0.75\}$).

3.7 Conclusion

Notre étude porte sur le flow shop à deux machines avec deux agents en compétition pour minimiser la combinaison linéaire de la somme des retards et du nombre de tâches en retard. Nous avons discuté de la complexité de calcul de divers cas particuliers. Pour le résoudre de manière optimale, nous avons proposé des modèles mathématiques ainsi qu'une procédure basée sur l'algorithme branch-and-bound qui utilise plusieurs propriétés de dominance et bornes. De plus, une heuristique basée sur différentes listes de priorités ainsi qu'un algorithme de recherche tabou qui utilise deux mécanismes de déplacement différents et un algorithme génétique avec trois types de méthodes de sélection sont proposés pour trouver une solution quasi-optimale. Enfin, des résultats de calcul pour des instances de problème générées aléatoirement sont présentés.

Chapitre

4 Flow shop sans attente à deux agents avec le critère makespan

4.1 Introduction

Dans ce chapitre, nous étudions le problème d'ordonnancement de type "flow shop" sans attente à deux machines avec deux agents en compétition. L'objectif est de minimiser le temps d'achèvement global d'un agent sous réserve d'une limite supérieure sur le makespan du second agent. Nous montrons la dureté $\mathcal{NP}$ pour trois cas particuliers : (i) chaque agent a exactement deux opérations. (ii) les tâches d'un agent ne nécessitent un traitement que sur une seule machine, (iii) la contrainte de sans-attente n'est requise que pour les tâches d'un agent. Nous présentons des algorithmes en temps polynomial pour d'autres cas restreints. Nous proposons également un modèle de programmation mathématique et un schéma de branche et limite comme approches de résolution pour les problèmes à petite taille. Pour les grandes instances, nous présentons un algorithme méta-heuristique de recherche tabou. Une étude expérimentale intensive est menée pour illustrer l'efficacité des algorithmes exacts et d'approximation proposés.

4.2 Questions relatives à la complexité

Dans cette section, nous discutons la complexité du problème de flow shop sans attente à deux machines avec deux agents en compétition sous différents scénarios. L'objectif est de minimiser le temps maximum d'achèvement de l'agent A, avec la restriction que le makespan de l'agent B ne peut pas dépasser une borne supérieure donnée Q, en respectant différentes contraintes. Les résultats de dureté $\mathcal{NP}$ faible et forte sont dérivés en utilisant une réduction de Partition et 3-Partition, respectivement.

En outre, nous utilisons le résultat suivant.

Théorème 7. *[28] $F2|no-wait|L_{\max}$ est $\mathcal{NP}$-difficile au sens fort même si seulement deux dates d'échéance différentes sont autorisées.*

Le statut de complexité des problèmes suivants est discuté dans le reste de cette section.

$F2|no-wait|C^A_{\max} : C^B_{\max} \leq Q$ **:** chaque tâche a exactement deux opérations, et les temps de traitement sont supposés être strictement positifs.

$F2|M_1 \rightarrow M_2, M_2, no-wait|C^A_{\max} : C^B_{\max} \leq Q$ **:** l'agent A traite tous ses tâches sur M_1 puis sur M_2, tandis que l'agent B traite ses tâches sur M_2 seulement.

$F2|M_1 \rightarrow M_2, M_1, no-wait, p_{ij} = p_j|C^A_{\max} : C^B_{\max} \leq Q$ **:** l'agent B traite ses jobs sur M_1 uniquement, et les temps de traitement sont proportionnels.

$F2|no-wait^A|C^A_{\max} : C^B_{\max} \leq Q$ **:** la contrainte de non-attente est appliquée aux jobs de l'agent A uniquement, tandis que les jobs de l'agent B sont réguliers (*i.e.* aucune restriction de non-attente en cours de traitement ne s'applique aux jobs de l'agent B).

L'ajout des contraintes $p_{ij}^A = p^A$, $p_{ij}^A = p_j^A$, $p_{ij}^A = p_i^A$ et $p_{ij} = p_j$ précise que les temps de traitement sont spéciaux comme mentionné dans la section précédente.

Théorème 8. *$F2|no-wait|C_{\max}^A : C_{\max}^B \leq Q$ est fortement $\mathcal{NP}$ difficile.*

Théorème 9. *$F2|M_1 \to M_2, M_2, p_{ij}^A = p^A, no-wait|C_{\max}^A : C_{\max}^B \leq Q$ est $\mathcal{NP}$ difficile, même si le nombre de jobs associés à l'agent A est réduit à un.*

Corollaire 4. *$F2|M_1 \to M_2, M_2, p_{ij}^A = p_j^A, no-wait|C_{\max}^A : C_{\max}^B \leq Q$ et $F2|M_1 \to M_2, M_2, p_{ij}^A = p_i^A, no-wait|C_{\max}^A : C_{\max}^B \leq Q$ sont $\mathcal{NP}$ difficiles même si l'agent A a un emploi.*

2

Théorème 10. *$F2|M_1 \to M_2, M_2, no-wait|C_{\max}^A : C_{\max}^B \leq Q$ est $\mathcal{NP}$-difficle au sens fort.*

Théorème 11. *$F2|M_1 \to M_2, M_1, no-wait, p_{ij} = p_j|C_{\max}^A : C_{\max}^B \leq Q$ est $\mathcal{NP}$-difficile au sens fort.*

Théorème 12. *$F2|no-wait^A|C_{\max}^A : C_{\max}^B \leq Q$ est $\mathcal{NP}$-difficile au sens fort même si $p_{1j} = p$ ou $p_{ij}^A = p_i^A$.*

4.3 Cas polynomiaux solvables

Rappelons que la recherche d'un algorithme polynomial, même pour des cas particuliers, est importante car elle trace une frontière entre les cas faciles et difficiles du problème.

Bien que plusieurs cas particuliers du flow shop soient $\mathcal{NP}$-difficile, il existe d'autres cas particuliers avec la restriction sans attente et différentes contraintes qui sont encore polynomialement solvables, comme il a été mentionné avant. Dans cette section, nous présentons d'autres cas qui sont également solvables en temps polynomial.

Théorème 13. *$F2|no-wait^X, p_{ij} = p_j|C_{\max}^A : C_{\max}^B \leq Q$ est solvable en temps linéaire pour $X \in \{A, B, \emptyset\}$.*

Le théorème ci-dessus peut être étendu au problème avec m machines

Corollaire 5. *$Fm|no-wait^X, p_{ij} = p_j|C_{\max}^A : C_{\max}^B \leq Q$ est solvable en $O(mn)$ en ordonnançant tous les travaux d'un agent avant les tâches du second agent .*

Théorème 14. *Si $\min\limits_{1\leq h\leq n}\{p_{2h}\} \geq \max\limits_{1\leq h\leq n}\{p_{1h}\}$ alors le problème $F2|no-wait^X, M_1 < M_2|C_{\max}^A : C_{\max}^B \leq Q$ est solvable en $O(n)$.*

Théorème 15. *Si $\min\limits_{1\leq h\leq n}\{p_{1h}\} \geq \max\limits_{1\leq h\leq n}\{p_{2h}\}$ alors le problème $F2|no-wait^X, M_1$ $N > M_2|C_{\max}^A : C_{\max}^B \leq Q$ est solvable en $O(n)$.*

Le reste de cette étude est consacré à la résolution du problème du flow shop sans attente à deux machines avec deux agents en compétition. Nous supposons que tous les temps de traitement sont strictement positifs. Le but est de trouver une permutation, σ, des tâches qui minimise le makespan de l'agent A, en respectant la contrainte que le makespan associé à l'agent B ne dépasse pas une borne supérieure donnée Q. Ce problème est noté $F2|no-wait|C_{\max}^A : C_{\max}^B \leq Q$. Nous fournissons des méthodes exactes et heuristiques pour résoudre le problème correspondant.

4.4 Modèles mathématiques

Dans cette section, nous présentons un modèle de programmation mixte en nombres entiers (MILP) pour résoudre le problème $Fm|no-wait|C^A_{\max} : C^B_{\max} \leq Q$. Nous utilisons le modèle disjonctif, qui a été introduit pour la première fois dans [20]. Ce modèle s'appuie sur les variables de précédence.

4.5 Algorithme de séparation et évaluation

Dans cette section, nous abordons l'algorithme branch-and-bound ($B\&B$) comme approche de résolution. Nous commençons par développer quelques résultats, avant de décrire l'algorithme. Nous rappelons qu'un algorithme de branch-and-bound consiste à décomposer les solutions réalisables d'un problème en sous-solutions successivement plus petites, à calculer les bornes inférieures de la valeur de la fonction objectif sur chaque sous-solution, et à les utiliser pour éliminer certaines sous-solutions de toute considération ultérieure. La procédure s'achève lorsque chaque sous-solution a abouti soit à une solution réalisable, soit à la démonstration qu'elle ne contient pas de meilleure solution que celle déjà en main. La meilleure solution trouvée à la fin de la procédure est un optimum global. Un diagramme représentant ce processus est appelé un arbre de recherche, dans lequel chaque nœud représente une solution partielle, et la racine une solution vide. L'application d'un algorithme de branch-and-bound nécessite de spécifier au moins l'un des ingrédients suivants : des règles de dominance, une règle de branchement, des règles de bornes inférieures et supérieures. Pour plus de détails, voir par exemple Balas et Toth [5].

Notre algorithme utilise une nouvelle règle de dominance pour réduire l'espace de recherche et nous permettre d'appliquer le schéma de branchement aux tâches d'un seul agent . Les positions des tâches de l'autre agent sont obtenues à l'aide de l'algorithme de Gilmore-Gomory, et ne dépendent pas de la durée des temps de traitement des tâches.

4.5.1 Borne supérieure

Pour développer une borne supérieure globale sur la valeur du makespan, nous proposons une heuristique, qui est basée sur des listes de priorités. Nous utilisons la même représentation d'une solution que dans l'algorithme branch-and-bound (S_1, S_2). L'algorithme commence par une solution réalisable, σ, où les tâches de l'agent B sont ordonnancées dans S_1, tandis que les tâches de l'agent A sont ordonnancées dans S_2. De plus, les tâches de chaque ensemble sont séquencées selon la méthode GGA. À chaque itération, nous essayons de déplacer les tâches de l'agent A vers S_1 afin d'améliorer la qualité de la solution tout en respectant la contrainte que $C^B_{\max}(\sigma) \leq Q$.

4.6 Approche méta-heuristique

La recherche tabou (TS) est une technique de recherche par voisinage qui commence par une solution initiale réalisable et se déplace vers une solution voisine. Elle impose des restrictions sous la forme d'une liste tabou sur certains déplacements afin d'éviter l'inversion de ces actions. Cependant, de temps en temps, nous autorisons l'utilisation de mouvements interdits s'ils mènent à une nouvelle meilleure solution qui n'a pas été visitée. Par conséquent, dans de telles situations, nous utilisons une fonction d'aspiration pour supprimer l'attribut interdit des mouvements. TS s'exécute jusqu'à ce qu'une condition d'arrêt prédéfinie soit remplie, en essayant de trouver des solutions réalisables tout en améliorant le makespan de l'agent A. À la fin, la meilleure solution trouvée au cours de la recherche est renvoyée.

4.7 Étude expérimentale

Pour étudier les performances des différentes méthodes proposées développées ci-dessus, une étude expérimentale est menée. Les instances du problème sont construites avec des données générées aléatoirement. Le nombre de tâches est considéré $\in \{50, 100, 200\}$ et les temps de traitement sont générés à partir de $[1, 20]$ et $[20, 100]$ pour les petits et grands temps de traitement, respectivement.

4.8 Conclusion

Dans cet article, nous avons considéré le problème d'ordonnancement de flow shop sans attente à deux machines avec deux agents en compétition. L'objectif est de minimiser le makespan du premier agent tout en maintenant le makespan du second agent sous une valeur fixe. Nous avons présenté des résultats de complexité par rapport à différents paramètres. Nous avons présenté des algorithmes en temps polynomial pour des versions restreintes de ce problème. Nous avons également développé un modèle de programmation mathématique, un algorithme de branch and bound et un algorithme de recherche tabou comme approches de résolution. En outre, nous avons mené une expérience informatique pour évaluer les performances des techniques développées.

Le modèle mathématique était capable de résoudre des instances jusqu'à 20 de tâches. Les résultats ont indiqué que l'algorithme branch and bound pouvait résoudre des instances avec plus de 40 tâches dans un temps raisonnable, et la méthode méta-heuristique semble plutôt performante en ce qui concerne la qualité de la solution qu'elle fournit.

Chapitre

5 Job shop à deux agents avec le critère makespan

5.1 Introduction

Dans ce chapitre, nous étudions le problème d'ordonnancement dans un atelier à deux machines avec deux agents en compétition. Le but est de minimiser une fonction objective donnée pour un agent, sous réserve d'une limite supérieure donnée d'une autre fonction objective associée au second agent. Nous prouvons que ce problème est $\mathcal{NP}$-difficile en fonction de divers critères et paramètres. En ce qui concerne le makespan pour les deux agents, nous avons d'abord exposé des algorithmes en temps polynomial pour plusieurs cas particuliers. Nous avons ensuite développé des modèles mathématiques, un algorithme branch and bound et un algorithme génétique, suivis d'une simulation expérimentale intensive pour étudier leurs performances.

5.2 Questions relatives à la complexité

Nous commençons cette section en rappelant que la partition est connue pour être $\mathcal{NP}$-dure au sens faible [10]. Nous avons utilisé ce problème dans le processus de réduction pour prouver la dureté $\mathcal{NP}$ des problèmes que nous considérons dans cette section.

Théorème 16. *Pour* $Fm|prmu, p_{ij} = p_j|C_{\max}$ *le makespan est égal à* $C_{\max} = \sum_{j=1}^{n} p_j + (m-1)\max\{p_1, \ldots, p_n\}$. *et est indépendant de l'horaire.*

Théorème 17. $J2|M_1 \mapsto M_2, M_2 \mapsto M_1, p_{ij} = p_i|C^A_{\max} : C^B_{\max} \leq Q$ *est* $\mathcal{NP}$*-difficile.*

Corollaire 6. $J2|p_{ij} = p_i|C^A_{\max} : C^B_{\max} \leq Q$ *est* $\mathcal{NP}$*-difficile.*

Corollaire 7. $J2|M_1 \mapsto M_2, M_2 \mapsto M_1|C^A_{\max} : C^B_{\max} \leq Q$ *est* $\mathcal{NP}$*-difficile.*

Corollaire 8. $J2||C^A_{\max} : C^B_{\max} \leq Q$ *est* $\mathcal{NP}$*-difficile.*

Corollaire 9. $J2|M_1 \mapsto M_2, M_2 \mapsto M_1, p_{ij} = p_j|C^A_{\max} : \sum_j C^B_j \leq Q$ *est* $\mathcal{NP}$*-difficile même si le nombre de tâches de l'agent* B *est réduit à un.*

Corollaire 10. $J2|p_{ij} = p_j|C^A_{\max} : \sum_j C^B_j \leq Q$ *est* $\mathcal{NP}$*-difficile même si le nombre de jobs pour l'agent* B *est réduit à un.*

Théorème 18. $J2|M_1 \mapsto M_2, M_1|C^A_{\max} : C^B_{\max} \leq Q$ *est* $\mathcal{NP}$*-difficile.*

Théorème 19. $J2|M_1 \mapsto M_2, M_1|\sum_i C^A_i : C^B_{\max} \leq Q$ *est* $\mathcal{NP}$*-difficile au sens fort.*

Théorème 20. $J2|M_1 \mapsto M_2, M_2, p^A_{ij} = p^A_i|C^A_{\max} : C^B_{\max} \leq Q$ *est* $\mathcal{NP}$*-difficile.*

Théorème 21. $J2|M_1 \mapsto M_2, M_2, p^A_{ij} = p^A_i|\sum_i C^A_i : C^B_{\max} \leq Q$ *est* $\mathcal{NP}$*-difficile même si le nombre de tâches de l'agent* A *est réduit à un.*

5.3 Cas polynomiaux solvables

Nous présentons dans cette section les cas restreints qui peuvent être résolus en temps polynomial.

Théorème 22. *Si* $Q \geq \sum_{j \in J_A} p_{1j}^A + \sum_{j \in J_B} p_{1j}^B$ *alors* $J2|M_1 \mapsto M_2, M_1|C_{\max}^A : C_{\max}^B \leq Q$ *est polynomialement solvable en ordonnançant d'abord* J_A *selon la règle de Johnson suivi de* J_B*.*

Théorème 23. *Si* $Q < \min_{j \in J_A}\{p_{1j}^A\} + \sum_{j \in J_B} p_{1j}^B$ *alors* $J2|M_1 \mapsto M_2, M_1|C_{\max}^A : C_{\max}^B \leq Q$ *est polynomialement solvable en ordonnançant d'abord* J_B *puis* J_A *selon la règle de Johnson.*

Théorème 24. *Le problème* $J2|M_1 \longmapsto M_2, M_1, p_{ij}^A = p_j|C_{\max}^A : C_{\max}^B \leq Q$ *est solvable en* $O(n \log n)$ *par l'algorithme* **??**.

Le théorème ci-dessus termine cette section sur les résultats de complexité. Le reste de cet article est consacré à la résolution du problème de job shop à deux machines avec deux agents et au plus deux opérations par job *i.e.* $J2||C_{\max}^A : C_{\max}^B \leq Q$. Nous présentons deux algorithmes d'énumération implicite *c'est-à-dire* les approches de programmation mathématique et de branch-and-bound, puis l'approche méta-heuristique. Rappelons que puisque ce problème est $\mathcal{NP}$-difficile, ces approches sont bien justifiées.

5.4 Modèles mathématiques

Dans cette section, nous présentons trois modèles de programmation mixte en nombres entiers (MIP) pour résoudre le problème d'ordonnancement job shop, avec le critère de makespan pour les deux agents en compétition. Rappelons que l'efficacité d'un modèle MIP dépend en partie du nombre de variables de décision et de contraintes qu'il comporte.

5.5 Algorithme de séparation et évaluation

Dans cette section, nous abordons l'algorithme branch and bound comme approche de résolution. Nous commençons par développer quelques résultats, avant de décrire l'algorithme. Nous rappelons brièvement qu'un algorithme de branch-and-bound consiste à décomposer les solutions réalisables d'un problème en sous-solutions successivement plus petites, à calculer les limites inférieures de la valeur de la fonction objectif sur chaque sous-solution et à les utiliser pour éliminer certaines sous-solutions de toute considération ultérieure. La procédure se termine lorsque chaque sous-solution a soit produit une solution réalisable, soit démontré qu'elle ne contient pas de meilleure solution que celle déjà trouvée. La meilleure solution trouvée à la fin de la procédure est un optimum global. Un diagramme représentant ce processus est appelé un arbre de recherche, dans lequel chaque nœud représente une solution partielle, et la racine une solution vide. L'application d'un algorithme de branch-and-bound nécessite de spécifier au moins l'un des ingrédients suivants : les bornes supérieure et inférieure, les règles de dominance et une règle de branchement. Pour plus de détails sur cette approche, voir par exemple Balas et Toth [5].

5.5.1 Question de mise en oeuvre

Nous supposons dans ce qui suit que $Q < C_{\max}^J(\sigma_J)$, le cas $Q \geq C_{\max}^J(\sigma_J)$ étant symétrique. Nous avons des opérations de l'agent B dans S_1. Les opérations de l'agent A sont réparties dans les ensembles S_1 et S_2. Chaque job J_i^A a deux opérations. Ainsi, pour chaque job, nous avons trois cas possibles : (i) les deux opérations de J_i^A sont dans S_1 ; (ii) une seule opération de J_i^A

est traitée dans S_1, et la seconde dans S_2 ; (iii) les deux opérations de J_i^A sont dans S_2. Les deux ensembles, S_1 et S_2, sont traités selon la règle de Jackson. Ainsi, le problème se révèle être un problème de partitionnement des opérations de l'agent A en deux sous-ensembles tels que le makespan de l'agent A soit minimisé, sous la contrainte que le makespan de l'agent B ne dépasse pas la valeur Q.

En ce qui concerne l'implémentation, nous procédons comme suit. Une solution est représentée par un vecteur de n_A éléments (Nous utilisons la même représentation de n_B éléments pour l'agent B). Le i-ème élément fait référence à l'état de la i-ème tâche de l'agent A, pour $i = 1, \ldots, n_A$, qui prend l'une des trois valeurs suivantes :

0 : Si les deux opérations du job i appartiennent à S_2.

1 : Si la première opération du travail i appartient à S_1 et la deuxième opération à S_2.

2 : Si les deux opérations de la tâche i appartiennent à S_1.

La solution générée peut être représentée par la Figure 5.1, où :

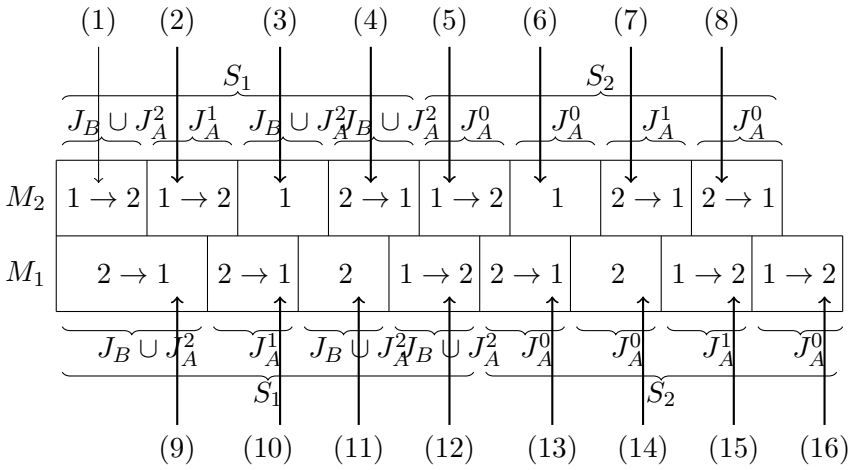

Figure 5.1 – *Ordonnancement des opérations de chaque agent pour le problème* $J2||C_{max}^A : C_{max}^B$.

5.6 Approche méta-heuristique

L'algorithme génétique est un algorithme basé sur la population, qui utilise les principes de la génétique et de la sélection naturelle. L'algorithme commence avec un ensemble d'individus (chromosomes), qui explorent l'espace de recherche. De nouveaux individus sont reproduits en recombinant les individus de la population actuelle à l'aide d'un opérateur de croisement. Pour ce faire, un sous-ensemble d'individus est choisi comme parents selon une méthode de sélection prédéfinie. Une nouvelle génération (enfants) est créée avec une probabilité P_c, qui hérite des gènes de ses parents. Pour éviter de se retrouver piégé dans des optima locaux, un opérateur de mutation est appliqué à certains enfants, avec P_m, une faible probabilité, ce qui nous permet d'éviter une convergence prématurée de l'algorithme.

5.7 Expériences computationnelles

Nous présentons dans cette section une étude expérimentale que nous avons menée, concernant l'efficacité des trois modèles mathématiques et de l'algorithme de branch-and-bound , en termes de temps CPU. Nous avons utilisé le package Cplex pour coder les modèles mathématiques. Nous avons également étudié la qualité des solutions produites par l'algorithme génétique.

Nous avons utilisé C++ pour coder l'algorithme de branch-and-bound proposé et l'algorithme génétique. Les problèmes testés sont exécutés sur un PC avec Intel(R) Core(TM) i7-2670QM, CPU 2.20 GHZ et 8.00 GB RAM sur le système d'exploitation Windows 7.

5.7.1 Performances des méthodes exactes

Cette section concerne l'étude computationnelle que nous avons menée sur les méthodes exactes *viz.* les trois approches de programmation linéaire en nombres entiers et de branch-and-bound que nous avons présentées, respectivement. Les instances sont générées aléatoirement et les exécutions se terminent après 3600s. Nous avons calculé le nombre de fois où les méthodes proposées produisent une solution optimale.

Nous avons testé les instances générées aléatoirement avec plus de 5000 tâches, pour les deux heuristiques et l'algorithme branch and bound, avec différentes intervalles de temps de traitement ($p_{ij} \in [1,20], [1,50], [1,100], [20,50]$, $[20,100]$), la valeur de la borne supérieure Q (entre $C^{J_B}_{\max}$ et $C^{A,B}_{\max}$), et les routes de traitement (pour chaque tâche, l'une des routes de traitement est générée aléatoirement *viz.* $M_1 \to M_2$, $M_2 \to M_1$, M_1, et M_2). Les résultats obtenus indiquent que ces instances sont faciles à résoudre, et dans la plupart des cas, les deux heuristiques ont pu générer des solutions optimales. Notre algorithme de branch and bound était capable de résoudre ces instances presque instantanément. Par conséquent, nous avons décidé pour le reste de cette étude expérimentale de générer des instances de type partition.

5.7.2 Performances de l'algorithme génétique

Dans cette section, nous analysons les performances de l'algorithme génétique que nous avons présenté. L'étude est principalement basée sur les résultats générés par cet algorithme par rapport aux bornes inférieures. Nous avons défini $n_B = 2 \times n_A$, et fixé la valeur de la borne supérieure à $Q = C^{J_B}_{\max}(\sigma_B) + (C^{J}_{\max}(\sigma_J) - C^{J_B}_{\max}(\sigma_B))/2$.

Nous avons généré aléatoirement les temps de traitement à partir de différents intervalles ($(p_{i1}, p_{i2}) \in [1,20] \times [1,20]$, $[1,50] \times [1,50]$, $[1,20] \times [1,50]$, $[1,20] \times [20,50]$, $[20,50] \times [20,50]$), et considéré différentes tailles de problème $n_A \in \{50, 100, 200, 500\}$. Pour chaque taille de problème, 100 instances ont été testées en fonction de l'intervalle des temps de traitement des jobs. Nous avons testé au total 2000 instances. Comme mentionné ci-dessus, l'algorithme branch and bound a été capable de résoudre toutes ces instances en peu de temps, même avec des jobs de 5000. En ce qui concerne l'algorithme méta-heuristique, les résultats produits sont 100% optimaux.

5.8 Conclusion

Dans cet article, nous avons étudié le problème du job shop à deux machines avec deux agents en compétition. Nous avons présenté des preuves de complexité pour plusieurs fonctions objectives. Pour certains cas restreints, nous avons développé des algorithmes en temps polynomial par rapport au makespan pour les deux agents. Pour le problème général, nous avons présenté trois formulations mathématiques pour minimiser le makespan pour un agent sous la contrainte de respecter une borne supérieure sur la valeur du makespan pour l'autre agent. Nous avons également conçu un algorithme de branch and bound et un algorithme génétique par rapport au makespan pour les deux agents.

Chapitre

6 Open shop à deux agents avec le critère makespan

6.1 Introduction

Cet article traite du problème de l'atelier ouvert à deux machines avec deux agents en concurrence. Le makespan est considéré comme une fonction objective pour les deux agents. La complexité du problème est étudiée sous différents paramètres. Trois modèles de programmation mathématique et un algorithme de branchement et de liaison sont proposés pour résoudre le problème de manière optimale. Nous présentons également une procédure heuristique qui est utilisée comme limite supérieure. Toutes les méthodes sont testées sur des instances générées aléatoirement. Les résultats de calcul montrent que l'heuristique est performante dans tous les tests et que l'algorithme de branchement et de liaison est capable de résoudre des instances allant jusqu'à 5000 tâches en très peu de temps.

6.2 Questions relatives à la complexité

Cette section est consacrée à l'étude de la complexité de l'open shop avec deux agents en compétition sous différents paramètres. Considérons d'abord l'open shop proportionnel, où les temps de traitement de chaque tâche sont égaux sur les deux machines, c'est-à-dire, $p_{ij} = p_i$.

Théorème 25. *$O2|p_{ij} = p_j|C^A_{max} : C^B_{max} \leq Q$ est $\mathcal{NP}$-difficile.*

Corollaire 11. *$O2|p_{ij} = p_j|C^A_{max} : \sum_{i \in J_B} C^B_i \leq Q$ est $\mathcal{NP}$-difficile, même si l'agent B a une seule tâche.*

6.3 Cas polynomiaux solvables

Théorème 26. *$O2|p_{ij} = p_i|C^A_{max} : C^B_{max}$ est solvable en $O(n)$.*

Dans ce qui suit, nous considérons le modèle réalisable dans un environnement open shop, nous supposons qu'il y a K agents en compétition. Notre objectif est de trouver une solution réalisable σ qui satisfait toutes les contraintes.

Théorème 27. *$O2|p_{ij} = p_i, C^k_{max} \leq Q_k, n_k \geq 2|-$ est solvable en $O(n + k \log k)$.*

6.4 Modèles mathématiques

L'objectif principal de cette section est l'exposition de trois modèles mathématiques de programmation mixte en nombres entiers (MIP) pour résoudre l'open shop à deux machines, avec des critères de makespan pour les deux agents en compétition.

6.5 Algorithme de séparation et évaluation

Dans cette section, nous développons un algorithme basé sur le schéma de branch and bound. Nous commençons par introduire quelques propriétés, qui nous permettent de réduire la zone de recherche, qui sera utilisée pour développer un algorithme de branch and bound. Nous présentons le concept de base du schéma de branchement ainsi que les bornes inférieures et supérieures. Dans ce qui suit, nous désignons par Y l'agent qui termine de traiter toutes ses tâches en premier, et par $\bar{Y}$ l'agent qui termine en second.

Propriété 1. *Il existe une solution optimale σ, dans laquelle un agent Y finit de traiter toutes ses tâches avant $\bar{Y}$, définissant deux ensembles S_1 et S_2, où S_1 contient toutes les deux opérations de toutes les tâches de l'agent Y et certaines opérations des $\bar{Y}$-jobs, tandis que S_2 contient le reste des opérations de $J_{\bar{Y}}$ qui n'ont pas encore été effectuées.*

Dans un ordonnancement optimal, nous pouvons diviser $J_{\bar{Y}}$ en quatre types :
R_1 : Ensemble de tâches dont une opération est exécutée sur M_1 dans S_1, et la seconde opération sur M_2 dans S_2.
R_2 : Ensemble de tâches avec une opération exécutée sur M_2 dans S_1, et la seconde opération sur M_1 dans S_2.
L : Ensemble des tâches dont les deux opérations appartiennent à S_2.
Z : Ensemble des tâches dont les deux opérations appartiennent à S_1.
Dans ce qui suit, nous allons utiliser la propriété suivante pour ordonnancer les tâches de S_2.

Propriété 2. *L'open shop à deux machines avec des critères de makespan et exactement une machine n'est pas disponible au temps zéro est solvable en $O(n)$.*

6.6 Expériences computationnelles

Dans un premier temps, nous avons comparé les performances des modèles de programmation mathématique proposés et de l'algorithme branch and bound. Un ensemble de problèmes est généré de manière aléatoire. Deux types d'intervalles de longueur (petit et long) sont utilisés pour produire les temps de traitement entiers, indépendamment d'une distribution uniforme pour toutes les opérations. Le premier type pour les petits temps de traitement sur l'intervalle $[1, 25]$. Le second pour les longs temps de traitement $[25, 100]$.

6.7 Conclusion

Dans cet article, nous avons étudié l'open shop multi-agent à deux machines. Chaque agent a son ensemble de tâches indépendantes et veut exécuter ses tâches dans un ensemble de machines partagées afin de minimiser les temps d'achèvement maximum de ses tâches. Des résultats de complexité ont été donnés pour ce problème sous une variété d'hypothèses.

Des modèles mathématiques et un algorithme de branch and bound ont été proposés afin de trouver une solution optimale, et une solution heuristique pour fournir une solution quasi-optimale. Les résultats de calcul ont montré que l'algorithme branch and bound est très rapide et peut résoudre des instances avec plus de 5000 de tâches. La déviation de l'heuristique par rapport à l'optimum est assez faible.

Chapitre

7 Conclusion générale

Dans ce travail, nous avons étudié le problème d'ordonnancement d'ateliers multi-agents. Le but du problème multi-agent est soit de trouver un ordonnancement qui minimise une combinaison des fonctions objectives des agents, soit de trouver un ordonnancement qui minimise l'objectif d'un agent tout en satisfaisant les exigences des autres agents pour leur propre fonctions objectifs.

Les problèmes d'ordonnancement multi-agents apparaissent dans les applications du monde réel. Dans un environnement de services multimédias, par exemple, des ressources radio limitées doivent être affectées à divers types de demandes de données. Chaque type de demande est considéré comme un agent et il peut avoir un critère différent des autres. Il est également intéressant d'introduire la notion de multi-agent au problème d'ordonnancement de la maintenance préventive en supposant que les tâches appartiennent à un agent et que l'exécution de la maintenance appartient à l'autre.

Nous avons commencé le première chapitre par une introduction générale.

Dans le deuxième chapitre, on a donné quelques définitions et notation de base, suivi par la classification des problèmes d'ordonnancement, la théorie de complexité et finalement une introduction sur la théorie d'ordonnancement multi-agents. La troisième chapitre a été consacré à l'étude de l'état de l'art. Dans les restes des chapitre, nous avons étudié quatre problème différents avec deux machines dans les environnements flow shop, job shop et open shop. Dans le chapitre 4, on a étudié le problème de flow shop avec deux machines et deux agents en compétition, l'objectif de ce chapitre est de minimiser la combinaison linaire de la somme des retards des jobs de premier agent et le nombre de tâches en retards du deuxième agent. Quelques résultats de complexité ont été établit pour des cas strict. Trois modèles mathématiques et une méthode de séparation et évaluation pour trouver des solutions optimales pour le problèmes, suivi par deux méta-heuristiques (Recherche Tabu et algorithme génétique) pour trouver des solutions approchées.

Dans le chapitre 5, nous étudions le problème du flow shop sans attente à deux machines pour minimiser le makespan des deux agents. ϵ-contrainte a été considérée comme approche de solution. Nous avons d'abord prouvé qu'elle est $\mathcal{NP}$-difficile au sens fort, ainsi que plusieurs cas stricts. Nous avons également proposé des algorithmes polynomiaux pour des cas particuliers. Un modèle de programmation mathématique ainsi qu'un algorithme de branch and bound ont été proposés pour résoudre le problème de manière optimale pour les instants de petite taille. Un algorithme de recherche tabou a été adapté pour générer des solutions approchées pour les instances de grande taille.

Nous avons abordé dans les chapitres 6 et 7 les problèmes de job shop et d'open shop à deux machines avec le critère de makespan pour les deux agents. Nous avons établi des résultats de type $\mathcal{NP}$-difficulté. On a aussi proposé des algorithmes en des temps polynomiaux pour résoudre de nombreux sous-problèmes faciles. Des modèles mathématiques et des méthodes de séparation et évaluation basé sur des propriétés, des bornes inférieure et supérieure sont proposé pour résoudre les deux problèmes.

Pour conclure chaque chapitre on a présenté les résultats des différentes expérimentations réalisées sur un ensemble des instances générer aléatoirement.

Bibliographie

[1] A. Agnetis, P. B. Mirchandani, D. Pacciarelli, and A. Pacifici. Scheduling problems with two competing agents. *Operations research*, 52(2) :229–242, 2004.

[2] S. G. D. P. A. S. Alessandro Agnetis, Jean-Charles Billaut. *Multiagent Scheduling : Models and Algorithms*. Springer-Verlag Berlin Heidelberg, 1 edition, 2014. ISBN 978-3-642-41879-2,978-3-642-41880-8.

[3] S. Azem. *Ordonnancement des systèmes flexibles de production sous contraintes de disponibilité des ressources*. PhD thesis, École Nationale des Mines de Saint-étienne, France., 2010.

[4] K. R. Baker and J. C. Smith. A multiple-criterion model for machine scheduling. *Journal of scheduling*, 6(1) :7–16, 2003.

[5] E. Balas and P. Toth. Branch and bound methods. chapter 10 in : The traveling salesman problem. el lawler, jk lenstra, ahg rinnooy kan, db shmoys, 1985.

[6] M. Benttaleb, F. Hnaien, and F. Yalaoui. Two-machine job shop problem under availability constraints on one machine : Makespan minimization. *Computers & Industrial Engineering*, 117 :138–151, 2018.

[7] P. J. Brewer and C. R. Plott. A binary conflict ascending price (bicap) mechanism for the decentralized allocation of the right to use railroad tracks. *International Journal of Industrial Organization*, 14(6) :857–886, 1996.

[8] P. Brucker. *Scheduling Algorithms*. Springer, 5th ed edition, 2007. ISBN 354069515X,9783540695158,9783540695165.

[9] H. Emmons and G. Vairaktarakis. *Flow shop scheduling : theoretical results, algorithms, and applications*, volume 182. Springer Science & Business Media, 2012.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. WH Freeman & Co, 1979.

[11] P. C. Gilmore and R. E. Gomory. Sequencing a one state-variable machine : A solvable case of the traveling salesman problem. *Operations research*, 12(5) :655–679, 1964.

[12] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM (JACM)*, 23(4) :665–679, 1976.

[13] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[14] J. R. Jackson et al. An extension of johnson's results on job idt scheduling. *Naval Research Logistics Quarterly*, 3(3) :201–203, 1956.

[15] S. M. Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1) :61–68, 1954.

[16] M. A. Kubzin and V. A. Strusevich. Two-machine flow shop no-wait scheduling with a nonavailability interval. *Naval Research Logistics (NRL)*, 51(4) :613–631, 2004.

[17] W.-C. Lee, S.-K. Chen, C.-W. Chen, and C.-C. Wu. A two-machine flowshop problem with two agents. *Computers & Operations Research*, 38(1) :98–104, 2011.

[18] L. Lu and M. E. Posner. An np-hard open shop scheduling problem with polynomial average time complexity. *Mathematics of operations research*, 18(1) :12–38, 1993.

[19] W. Luo, L. Chen, and G. Zhang. Approximation schemes for two-machine flow shop scheduling with two agents. *Journal of Combinatorial Optimization*, 24(3) :229–239, 2012.

[20] A. S. Manne. On the job-shop scheduling problem. *Operations Research*, 8(2) :219–223, 1960.

[21] B. Mor and G. Mosheiov. Polynomial time solutions for scheduling problems on a proportionate flowshop with two competing agents. *Journal of the Operational Research Society*, 65(1) :151–157, 2014.

[22] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1) :160–175, 1996.

[23] P. S. Ow. Focused scheduling in proportionate flowshops. *Management Science*, 31(7) : 852–869, 1985.

[24] J. M. Peha. Heterogeneous-criteria scheduling : minimizing weighted number of tardy jobs and weighted completion time. *Computers & operations research*, 22(10) :1089–1100, 1995.

[25] C. Pessan, J.-L. Bouquard, and E. Neron. An unrelated parallel machines model for an industrial production resetting problem. *European Journal of Industrial Engineering*, 2(2) : 153–171, 2008.

[26] M. L. Pinedo. *Scheduling : theory, algorithms, and systems*. Springer-Verlag, New York, 2012.

[27] R. M. V. Rachamadugu, A. Vepsalainen, and T. E. Morton. Scheduling in proportionate flowshops. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1982.

[28] H. Röck. Some new results in flow shop scheduling. *Zeitschrift für Operations Research*, 28 (1) :1–16, 1984.

[29] S. Sahni and Y. Cho. Complexity of scheduling shops with no wait in process. *Mathematics of Operations Research*, 4(4) :448–457, 1979.

[30] V. T'kindt and J.-C. Billaut. *Multicriteria scheduling : theory, models and algorithms*. Springer Science & Business Media, 2006.

[31] G. L. Vairaktarakis. Simple algorithms for gilmore–gomory's traveling salesman and related problems. *Journal of Scheduling*, 6(6) :499–520, 2003.