N° d'ordre : 09/2017-D/MT

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene

Faculté de Mathématiques

THESE DE DOCTORAT EN SCIENCES

En vue de l'obtention du Grade du Docteur
EN : **Mathématiques**
Spécialité : **Recherche opérationnelle "Génie mathématiques"**

Présentée par : CHIKHI Nacira

Sujet

Flow shop sous Contraintes de Transport Et Approche Multicritère

Soutenue publiquement le 11/05/2017 **à 11h30, devant le jury composé de :**

M. AIDER Méziane, Prof./à l'Université USTHB	Président
M. ABBAS Moncef, Prof./à l'Université USTHB	Directeur de thèse
M. HANAFI Saïd, Prof./à l'Universié de Valenciennes, France	Co-directeur de thèse
M. AIDENE Mohamed, Prof./à l'Université de Tizi-Ouzou	Examinateur
M. MOULAI Mustapha, Prof./à l'Université USTHB	Examinateur
M. REBAINE Djamal, Prof./à l'Université de Chicoutimi, Canada	Examinateur

Résumé

L'objet de cette thèse est la résolution d'un problème d'ordonnancement de flow shop robotique à deux étages avec deux machines dédiées au premier étage et une machine commune au deuxième étage. Tout d'abord, nous proposons une résolution du problème mono-objectif dont l'objectif est de minimiser le makespan. En effet, nous avons traité le cas à capacité unitaire du robot et le cas de capacité non unitaire limitée à plusieurs tâches. Nous proposons deux modèles mathématiques en variables mixtes. Ces deux modèles sont testés sous le solveur CPLEX sur plusieurs schémas de génération. Des bornes inférieures ont été établies pour le makespan. Nous avons également procédé à la conception de trois heuristiques pour la résolution du problème général avec capacité quelconque du robot et une étude de complexité de ce problème général a été effectuée. Nous avons également conçu deux métaheuristiques pour sa résolution dans le cas de capacité unitaire du robot. La première est basée sur les algorithmes génétiques et la deuxième sur la méthode de recherche à voisinage variable en proposant plusieurs variantes de RVV avec des structures de voisinage efficaces. Ensuite, nous avons développé une recherche tabou multi-objectif que nous avons appliquée sur la même configuration du problème mono-objectif avec capacité unitaire mais en minimisant deux objectifs dont l'un est global. Cependant, la méthode est aussi applicable au cas avec plusieurs critères. Les différents tests effectués révèlent l'efficacité et la performance des méthodes proposées utilisées en terme de la qualité de la solution et le temps d'exécution.

Mots clés : Ordonnancement, Flow shop robotique, transport, makespan, deux étages, MIP, algorithmes génétiques, programmation dynamique, recherche à voisinage variable, recherche tabou multi-objectif.

Abstract

This thesis deals with the problem of scheduling a two stage flow shop in a robotic cell with respect to, first, one criterion (the makespan) and, then, two criteria (two makespan of two agents). The jobs are transported between stages by a robot with limited capacity. We formulated the general problem as a mixed integer programming problem with respect to the makespan and established the complexity of different variants of this problem. We have also proposed several lower bounds and approximate solution methods to solve it.

We have developed methods based on genetic algorithm and on variable neighborhood search to solve the NP-hard problem with unit capacity. We have used several efficient neighborhood structures. On the multicriteria front, we designed methods based on tabu

search with several efficient neighborhood structures for the case of unit capacity. This method is applied to bi-criteria problem to minimize the makespan of two agents such that one agent is global. The evaluation of the proposed approaches on different tests shows a remarkable efficiency and effectiveness.

Keywords : scheduling, flow shop, transport, heuristic, two-stage, MIP, variable neighborhood search, genetic algorithm, dynamic programming, multi-objective tabu search.

Table des matières

1 La théorie de l'ordonnancement **12**

1.1 Introduction . . . 12
1.2 Typologie des problèmes d'ordonnancement . . . 13
1.2.1 Environnement des machines . . . 13
1.2.2 Caractérisation des tâches . . . 14
1.2.3 Critères d'optimalité . . . 15
1.3 Flow shop . . . 15
1.3.1 Flow shop classique . . . 16
1.3.2 Flow shop hybride . . . 16
1.4 Systèmes Flexibles de Production . . . 17
1.4.1 Définition du SFP . . . 18
1.4.2 Différents types du SFP . . . 18
1.5 Théorie de la complexité . . . 19
1.5.1 Complexité algorithmique . . . 19
1.5.1.1 Notations de la complexité algorithmique . . . 19
1.5.2 Complexité des problèmes . . . 20
1.5.2.1 Prouver la NP-complétude d'un problème . . . 21
1.5.2.2 Hiérarchie de complexité . . . 21
1.6 Problème d'optimisation . . . 22
1.7 Méthodes de résolution . . . 23
1.7.1 Méthodes exactes . . . 23
1.7.1.1 Modélisation analytique . . . 24
1.7.1.2 Programmation dynamique . . . 24
1.7.2 Méthodes heuristiques constructives . . . 25
1.7.3 Méthodes amélioratrices . . . 27

1.7.3.1 Méthode de descente . . . 27
1.7.3.2 Recherche avec tabous . . . 28
1.7.3.3 Recherche à voisinage variable . . . 30
1.7.3.4 Recherche locale itérée . . . 31
1.7.3.5 Algorithmes Génétiques . . . 32
1.8 Conclusion . . . 34

2 Flow shop sous contraintes de transport 35
2.1 Introduction . . . 35
2.2 Etat de l'art . . . 36
2.3 Enoncé du problème . . . 39
2.4 Programmation mathématique à variables mixtes . . . 41
2.5 Bornes inférieures . . . 43
2.6 Heuristiques . . . 45
2.7 Expérimentations numériques . . . 48
2.8 Conclusion . . . 55

3 Recherche avec tabous multi-objectif 56
3.1 Introduction . . . 56
3.2 Problème multi-agent . . . 58
3.3 Optimisation multi-objective et dominance . . . 59
3.3.1 Concept de dominance . . . 60
3.3.2 Degré de dominance . . . 61
3.3.3 Front de Pareto et points de référence . . . 62
3.4 Approches multicritère . . . 63
3.4.1 Méthodes agrégées . . . 63
3.4.2 Méthodes basées sur l'équilibre de Pareto . . . 64
3.4.3 Méthodes non agrégées et non Pareto . . . 65
3.5 Recherche tabou multi-objectif . . . 65
3.5.1 Structures de voisinages utilisées . . . 65
3.5.2 Algorithme de la recherche avec tabous multi-objectif . . . 66
3.6 Résolution par la recherche avec tabous multi-objectif . . . 68
3.7 Evaluation numérique . . . 68
3.8 Conclusion . . . 75

Conclusion générale 77

Bibliographie **85**

Table des figures

1.1 Hiérarchie de complexité entre différents problèmes 22
1.2 Différence entre un optimum global et des optima locaux 23

2.1 Illustration du problème 38
2.2 Diagramme de Gantt de l'exemple. 40
2.3 Procédure de construction des batches 45
2.4 Organigramme de l'heuristique H_1 47
2.5 Organigramme de l'heuristique H_2 48
2.6 Comparaison entre le makespan de l'heuristique H_i et la borne inférieure LB ($Classe: \ Cl_2, \ n = 100$) 54

3.1 Différents scénarios pour un problème d'ordonnancement à deux agents . . 59
3.2 Points de référence 62
3.3 Types de voisinages utilisés 66
3.4 Organigramme de la recherche tabou 67
3.5 Exemple de front de Pareto généré aléatoirement pour une instance avec $n = 20$ et $n_1 = 10$ 74
3.6 Exemple de front de Pareto généré aléatoirement pour une instance avec $n = 200$ et $n_1 = 100$ 75

Liste des tableaux

1.1 Notations de la complexité 20

2.1 Temps de traitement 40
2.2 Expérimentations numériques sur des instances de petite taille. 50
2.3 Expérimentations numériques pour $n = 50$. 51
2.4 Classes d'instances 52
2.5 Expérimentations numériques sur des instances de grande taille. 53

3.1 Expérimentations numériques sur des instances de petite taille. 70
3.2 Expérimentations numériques sur des instances de grande taille. 71
3.3 Résultats des instances de petite taille. 72
3.4 Résultats des instances de grande taille. 73

Introduction générale

Les problématiques abordées dans cette thèse rentrent dans le cadre de l'optimisation combinatoire. Les enjeux économiques et environnementaux renforcent l'importance de l'optimisation combinatoire dans les domaines de la recherche opérationnelle et l'informatique. C'est pour cela que de nos jours, l'optimisation des transports et la performance logistique sont des enjeux économiques très importants pour les entreprises. Particulièrement, le problème de transport de pièces semi-finies dans les ateliers de production occupe une place importante dans la vie économique des sociétés modernes. Avec les contraintes temporelles et économiques qu'implique ce problème, sa résolution devient très difficile, nécessitant l'utilisation d'outils issus de disciplines différentes (Recherche Opérationnelle, Productique, informatique, ... etc.). En effet, les processus issus des systèmes de transports et de l'ordonnancement sont de plus en plus complexes, par leurs dimensions importantes, par la nature de leurs relations dynamiques, et par la multiplicité des contraintes auxquelles ils sont soumis. Ainsi, la plupart des solutions commerciales proposant une optimisation n'aboutissent pas souvent sur de véritables outils d'optimisation mais proposent plutôt, des solutions organisationnelles.

Les modèles d'ordonnancement d'ateliers, étudiés dans la littérature, ne reflètent pas toujours les difficultés rencontrées dans la pratique. Il est utile de mentionner la remarque de Lee et Chen [50] " les problèmes, s'intéressant à la coordination optimale d'ordonnancement de machines et de transport des tâches, ont une approche certainement plus applicable que les problèmes d'ordonnancement qui n'incluent pas ces facteurs". Ceci pouvant s'illustrer par des problèmes ne traitant qu'un petit nombre de machines ou de convoyeurs. Ainsi, l'étude de Kise et al. [46] explique cette réduction par le fait que les petits systèmes peuvent être construits et améliorés rapidement et cela, de façon économique. De plus, leurs maintenance et gestion se trouvent ainsi facilitées, tant au niveau matériel que logiciel.

Les systèmes de production et de fabrication actuels offrent une grande flexibilité, tel que les systèmes flexibles de production (Flexible Manufacturing System - FMS) qui fournissent des avantages divers comme l'accroissement de l'utilisation des ressources, l'amplification de la productivité, la diminution des encours, des retards,... etc.

Aussi des études analytiques sont menées sur l'ordonnancement de flow shop automatisé. Kise et al. [46] ont étudié le cas de deux machines flow-shop et un seul véhicule autoguidé (AVG). Pour trouver un ordonnancement optimal qui minimise le makespan, les auteurs ont donné un algorithme polynomial basé sur l'algorithme connu de Gilmore-Gomory. Kise [47] a aussi étudié le problème à deux machines flow shop avec un espace de stockage à chaque machine et un-AGV (ou un robot). L'objectif est de minimiser le makespan. Il a montré que le problème est un cas particulier du problème d'ordonnancement classique à trois machines flow-shop, et il a prouvé que ce problème est NP-difficile. Kise et al.[45] ont proposé un algorithme de type branch-and-bound pour le même problème, et ont démontré par des expériences numériques que l'algorithme peut résoudre des instances de taille inférieure ou égale à 200 tâches dans un temps d'exécution raisonnable. Panwalker [66] a étudié le problème de la minimisation du makespan d'un flow shop à deux machines avec un convoyeur faisant l'aller retour entre les deux machines. Il n'y a pas de stock en sortie de la première machine mais il y a un stock infini en entrée de la seconde machine. Il a donné un algorithme polynomial en temps quadratique pour résoudre le problème. Levner et al. [55] ont considéré une cellule robotique de même configuration que celle considérée par Panwaler sauf qu'ils n'ont pas négligé les temps de chargement et de déchargement, ils ont donné un algorithme polynomial qui minimise la date d'achèvement de tous les travaux. Cheng et Kise [11] ont trouvé un algorithme optimal qui minimise le makespan pour un problème d'ordonnancement de flow-shop automatisé à deux machines avec des opérations intermédiaires. Stern et Vitner [71] ont considéré le problème d'ordonnancement à deux machines flow-shop avec un robot et des temps de transport dépendants des tâches. Ils ont montré que le problème de la minimisation du makespan est NP-difficile et ont proposé un algorithme approché.

Le travail présenté dans cette thèse porte sur le problème d'ordonnancement de lignes de production. Dans ce type d'atelier, les ressources de transport revêtent une importance capitale. En effet, ce sont elles qui constituent les éléments critiques dont dépendent les performances globales du système. Dans ce type d'atelier, les temps de transports des produits à l'intérieur des ateliers ne peuvent pas être négligés devant les durées opératoires. C'est une caractéristique du problème d'ordonnancement dans les ateliers flexibles de production doté d'un système de transport automatisé et autoguidé. L'automatisation de

ces lignes de production nous incite à chercher un ordonnancement optimisé des opérations de transport du robot (ou véhicule autoguidé), ayant pour fonction d'assurer le transport des produits entre les machines de traitement ou les étages de production.

Dans certaines situations dans la pratique, l'application de la même mesure pour tous les travaux n'est pas pertinente. En effet, il est possible d'envisager un atelier où les travaux ont la particularité suivante. Pour chaque sous-ensemble, le décideur veut optimiser un certain critère. L'ordonnancement de chaque sous-ensemble est évalué en fonction de différents objectifs, mais les travaux sont tous en concurrence pour l'utilisation des machines. Ce genre de problèmes d'ordonnancement sont connu dans la littérature par les problèmes multi-agents qui ont attiré beaucoup d'attention dans la dernière décennie [1]. Ce sont aussi des problèmes multi-objectif [74, 60], dont le but est de trouver un ensemble de solutions offrant différents compromis entre les critères considérés. Résoudre ce type de problèmes de façon exacte est souvent impossible en raison de la difficulté intrinsèque du problème ou du grand nombre d'optima. Ainsi, des méthodes approximatives telles que les métaheuristiques sont couramment utilisées afin de trouver une bonne approximation de l'ensemble de solutions de compromis.

Dans cette thèse, nous nous intéressons à un problème d'ordonnancement dans un atelier flexible de type flow shop à deux étages avec machines dédiées, et dans lequel un robot est chargé de transporter les tâches entre les deux étages. Dans cet atelier, nous avons deux types de tâches (deux agents). Les tâches du premier type passent d'abord par la première machine dédiée au premier étage puis par la machine commune qui se trouve au deuxième étage. Par ailleurs, les tâches de deuxième type passent par la deuxième machine au premier étage, puis par la machine commune au deuxième étage. Donc, les tâches de ces deux agents concurrents partagent la même ressource (machine commune) qui se trouve au deuxième étage.

Dans l'étude de notre problème, on considère en premier lieu, un seul critère à minimiser qui est le temps total d'achèvement de toutes les tâches (le makespan de l'agent global), et, puis en second, deux critères à minimiser dont le premier critère est le makespan de l'agent global et le second critère est le makespan du premier agent qui contient uniquement les tâches du premier type. Au mieux de notre connaissance, dans la littérature, notre problème n'a pas été traité dans les deux cas mono-objectif et multi-objectif. En plus, notre apport consiste aussi en l'ajout de contraintes de transport en prenant en compte la capacité unitaire et non unitaire du moyen de transport (robot), dans un environnement de flow shop flexible avec machines dédiées qui est un nouveau modèle (flow shop de différenciation [59]).

Cette thèse s'articule autour de cinq chapitres et structurée comme suit.

Le chapitre 1 présente le domaine d'étude. En effet, après une présentation générale des problèmes d'ordonnancement, nous rappelons les principales approches utilisées pour leur résolution.

Le chapitre 2 aborde le problème du flow-shop à deux étages avec deux machines dédiées au premier étage et une machine commune au deuxième étage en prenant en compte les contraintes liées au transport des tâches entre les deux étages par un robot. Nous présentons pour ce problème, un état de l'art et une modélisation. En particulier, nous donnons deux formulations linéaires en nombres entiers avec une comparaison théorique et expérimentale entre ces deux modèles. Nous établissons des bornes inférieures pour la fonction objectif qui sont utilisées comme une référence afin d'évaluer les performances des méthodes proposées. Nous avons développé des heuristiques pour la résolution de problème général avec des résultats numériques basés sur plusieurs schémas de génération selon la loi uniforme. Ces tests révèlent la performance des heuristiques proposées.

Le chapitre 3 est consacré à une étude de complexité de quelques cas particuliers du problème. Nous démontrons les cas NP-difficiles du problème. Nous cernons également les cas des problèmes résolubles en des temps polynomiaux et nous présentons les algorithmes pour les résoudre de façon optimale. En effet, pour résoudre un problème d'ordonnancement, nous devons toujours chercher à établir sa complexité, car cela détermine la nature de l'algorithme à mettre en œuvre.

Le chapitre 4 est dédié à une variante du problème dans le cas de capacité unitaire du robot. Pour résoudre efficacement ce problème, nous avons proposé deux approches. La première est basée sur les algorithmes génétiques et la seconde approche est basée sur la méthode de recherche à voisinage variable (RVV). La RVV utilise un ensemble de voisinages que nous avons défini pour ce problème, et met en oeuvre une stratégie d'exploration des voisinages afin d'améliorer la solution. Les résultats obtenus sur des instances générées aléatoirement montrent l'efficacité des méthodes proposées qui donnent des solutions de très bonne qualité en un temps raisonnable.

Dans le chapitre 5, nous avons proposé une métaheuristique qui est la méthode de recherche tabou multi-objectif. Nous l'avons appliqué pour résoudre le problème d'ordonnancement de type flow shop robotique bi-critère avec deux agents concurrents dont l'un est global et l'autre est le premier agent qui contient uniquement les tâches du premier type. Chaque agent minimise son makespan.

Cette thèse se termine par une conclusion dans laquelle, nous proposons des pistes de recherche dans la continuité de nos travaux.

Chapitre 1

La théorie de l'ordonnancement

1.1 Introduction

La théorie de l'ordonnancement est connue comme un domaine de recherche dans les années 1950. L'une des premières publications d'ordonnancement qui sont apparues dans la littérature de recherche opérationnelle sont les résultats de Johnson [42]. Pendant les années 1960, une grande quantité de travail a été effectué sur la programmation des formulations en nombre entiers et la programmation dynamique des problèmes d'ordonnancement. Après, la recherche dans les années 1970 se concentre principalement sur la hiérarchie de la complexité des problèmes d'ordonnancement. Dans les années 1980, différentes directions ont été poursuivis dans la recherche académique et dans l'industrie. Depuis lors, le domaine a attiré beaucoup d'attention de chercheur et des papiers impressionnants ont été publié [44].

De façon précise, l'ordonnancement est une branche importante de l'optimisation combinatoire. Plusieurs applications dans des situations de la vie réelle, leurs modèles mathématiques correspondants et la difficulté surprenante de ces problèmes d'ordonnancement encouragent les communautés de recherche de poursuivre les recherches dans cette branche. L'ordonnancement continue à offrir des possibilités fructueuses entre la théorie et la pratique.

Pour une étude complète de la théorie de l'ordonnancement, on peut se référer à quelques papiers [68, 9]. Avant sa description formelle et afin de positionner les problèmes d'ordonnancement de cette thèse, nous présentons un aperçu des problèmes d'ordonnancement et des approches de résolution. Ce chapitre couvre une typologie des problèmes d'ordonnancement, les résultats de base pour certains problèmes de planification et les méthodes de résolution associées.

1.2 Typologie des problèmes d'ordonnancement

Dans cette section, nous distinguons la typologie et la notation des problèmes d'ordonnancement. Une typologie est une classification des problèmes en fonction de leur nature. L'ordonnancement dépend généralement de l'environnement des machines et sur les particularités des tâches. Une notation nous permet d'identifier rapidement un problème. Les notations traditionnelles dans l'ordonnancement sont clairement basées sur les typologies existantes.

Graham et al. [32] ont introduit la notation standard $\alpha/\beta/\gamma$ pour représenter les problèmes d'ordonnancement. Cette notation incarne les trois principaux éléments qui définissent le problème d'ordonnancement : l'environnement de la machine α, les caractéristiques de la tâche β, et le critère d'optimisation γ. Dans la suite, nous donnons brièvement les détails de ces trois champs.

1.2.1 Environnement des machines

Il y a plusieurs environnements d'ateliers représentés par le champs $\alpha = \alpha_1\alpha_2$ qui sont résumés ci-après :

- Machine discrète (simple) $\alpha_1 = 1$: Une seule machine est disponible pour le traitement des tâches. Il est l'environnement le plus élémentaire. En outre, la résolution de problèmes plus complexes est souvent atteinte par l'étude des problèmes avec machine discrète. Nous pouvons trouver un domaine d'application comme une machine de calcul, si l'on pense de la machine comme un seul processeur de l'ordinateur. Les tâches à traiter sont nécessairement mono-opération.
- machines parallèles : Il est une simple extension de l'environnement sur une seule machine, où il y a m machines disponibles qui puissent traiter chaque tâche. Il existe trois types de machines parallèles.
  - Machines parallèles identiques ($\alpha_1 = P$) : le temps de traitement p_j de la tâche j est le même pour toutes les machines.
  - Machines parallèles uniformes ($\alpha_1 = Q$) : Le temps de traitement p_{ij} de la tâche j dépend de la vitesse v_i de la machine i ($p_{ij} = \frac{p_j}{v_i}$).
  - Machines indépendantes ($\alpha_1 = R$) : ces machines ont des vitesses différentes, mais ces vitesses dépendent des tâches ($p_{ij} = \frac{p_j}{v_{ij}}$).
- Flow shop ($\alpha_1 = F$) : Les machines sont en série. Chaque tâche doit subir de multiples opérations sur un certain nombre de machines différentes et doit être traitée sur chaque machine dans le même ordre (les cheminements de toutes les tâches sont

identiques). Dans un atelier de flow shop de permutation, nous constatons en outre que chaque machine a la même séquence de tâches.

- Job shop ($\alpha_1 = J$) : Ce modèle est similaire au flow shop, avec une seule différence telle que chaque tâche a une voie qui lui est propre, c'est-à-dire elle passe par les machines avec son propre ordre. Dans un job shop, un job peut-être aussi exécuté plusieurs fois par la même machine.
- Open shop ($\alpha_1 = O$) : Il est similaire à l'atelier de Job shop où chaque tâche doit être traitée sur chacune des machines. Cependant, il n'y a aucune restriction sur le routage de chaque tâche.

Quelques informations supplémentaires sur l'environnement machine peuvent être trouvées dans le champs α. Par exemple, $\alpha = F2$ indique qu'il y a deux machines dans l'atelier de flow shop considéré.

1.2.2 Caractérisation des tâches

Le second champs β porte sur les caractéristiques des tâches et peuvent contenir plus d'un élément. Ci-dessous, nous énumérons certaines de ces caractéristiques :

- La préemption (pmtn) : Le traitement de toute opération peut être interrompu et repris à une date ultérieure.
- Les contraintes de précédence (prec) : une règle de priorité des contraintes entre les tâches exige que une ou plusieurs tâches doivent être traitées avant le début de traitement d'une autre tâche.
- Date de disponibilité (r_j) : elle représente le premier instant où la tâche j peut commencer son traitement.
- Date d'échéance (d_j) : Une tâche j peut avoir une date d_j pour laquelle elle devrait être terminée.
- Setup time (s_j) : Dans certaines applications, les machines doivent être mis en place avant de pouvoir commencer le traitement. Alors, chaque tâche j a un temps de set up, au cours duquel la machine qui traite la tâche j est bloquée.
- Poids (w_j) : Le poids w_j de la tâche j note son importance (ou priorité) par rapport aux autres tâches dans le système.
- Les contraintes de manutention des matériaux : systèmes de montage modernes qui possèdent souvent des moyens de transport automatisés qui transmettent les tâches d'une machine à l'autre et ont besoin d'un temps de transport non négligeable désigné par t_k. Par conséquent, un système de manutention impose une forte dépendance entre le temps de démarrage d'une opération donnée et les temps de réalisation de

ses prédécesseurs. En outre, la présence d'un système de manutention limite souvent la quantité d'espace de stockage, ce qui limite la quantité de travail en cours de fabrication.

– Les contraintes de blocage : Une tâche ayant terminé son traitement sur une machine ne peut pas la quitter que si la machine suivante est libre.

1.2.3 Critères d'optimalité

La troisième champ γ se réfère au critère d'optimalité. Les critères les plus couramment choisis impliquent la minimisation de :

– Makespan (C_{max}) : Le temps de la fin de la dernière tâche sur la dernière machine.
– Retard maximum (L_{max}) : La pire violation des dates d'échéance. Le retard de la tâche est non négatif si elle est terminée tard et négatif autrement.
– Retard maximal (T_{max}) : La différence entre le retard maximum et le retard maximal est que le retard maximal est égal à zéro si le travail est achevé tôt ($T_{max} = max\{0; L_{max}\}$).
– Temps d'écoulement maximal (F_{max}) : Le temps d'écoulement d'une tâche désigne le temps écoulé depuis son entrée à sa sortie du système.
– Temps d'achèvement total (pondéré) ($\sum C_j$ ou $\sum w_j C_j$) : La somme des temps d'achèvement (pondéré). Il indique les coûts totaux de maintien (ou de l'inventaire) encourus par l'ordonnancement. Ce critère est souvent désigné sous le nom de temps total d'écoulement (pondéré).
– Retard Total (pondéré) ($\sum Tj$ ou $\sum w_j T_j$) : Il est une généralisation de la fonction de temps (pondéré) de l'achèvement total.
– Nombre de tâches tardives (pondéré) ($\sum U_j$ ou $\sum w_j U_j$) : Une tâche est considérée comme tardive si elle est achevée après la date d'échéance.

1.3 Flow shop

En flow shop, tous les jobs visitent les machines dans le même ordre, avec des durées opératoires pouvant être différentes. Les machines sont disponibles sur tout l'horizon de planification et elles ne peuvent exécuter qu'une seule opération à la fois. Par ailleurs, une opération ne peut s'exécuter que sur une seule machine à la fois. En pratique, ce cas est fréquemment rencontré, citons l'exemple d'une chaîne de fabrication ou de montage.

1.3.1 Flow shop classique

Dans un flow shop classique, il y a m machines en série. Chaque tâche doit être traitée avec ou (sans) préemption sur chacune des m machines. Toutes les tâches ont le même itinéraire à travers lequel elles doivent être traitées en premier sur la machine 1, puis sur la machine 2 et ainsi de suite. Depuis la publication de l'article fondateur de Johnson [42], le problème de flow shop est devenu l'un des sujets les plus étudiés dans la théorie de l'ordonnancement. Le problème de flow shop est toujours considéré comme un problème très difficile à résoudre. Il est bien connu que le cas de minimiser le makespan dans deux machines ($m = 2$), pourrait être facilement résolu en utilisant la règle de Johnson qui génère un ordonnancement optimal en $O(n \log n)$. L'algorithme de Johnson est fondé sur le calcul itératif du minimum des durées des tâches associées aux travaux non encore placés. Nous présentons ci-dessous l'algorithme de Johnson [42]. Au-delà de deux machines, le

Algorithm 1 Algorithme de Johnson :

1: U = $\emptyset$; V = $\emptyset$;
2: **Tant que** la liste des tâches est non vide **faire**
3: Placer dans l'ensemble U les tâches pour lesquels $p_{1j} <= p_{2j}$;
4: Placer dans l'ensemble V les tâches pour lesquels $p_{1j} > p_{2j}$;
5: **Fin tant que**
6: Ordonnancer les tâches de l'ensemble U dans l'ordre croissant des p_{1j} ;
7: Ordonnancer les tâches de l'ensemble V dans l'ordre décroissant des p_{2j} ;
8: Fusionner les deux ensembles U et V ;
9: Exécuter la permutation obtenue sur les deux machines tout en respectant le même ordre.

flow-shop est un problème NP-difficile [29], il n'existe pas l'algorithme menant à une solution optimale en temps polynomial, des heuristiques permettent néanmoins d'obtenir des solutions intéressantes.

1.3.2 Flow shop hybride

Afin d'être toujours plus réactives et productives, les entreprises ont cherché à augmenter la flexibilité de leurs systèmes de production. Pour atteindre ce but, il est possible de multiplier le nombre des machines qui peuvent réaliser une même opération. Ces machines, considérées comme identiques, sont regroupées en étage. Le modèle résultant est connu dans la littérature sous le nom de Flow-Shop hybride (FH). Le Flow-Shop hybride

est donc une généralisation du Flow-Shop classique au cas où plusieurs machines sont disponibles sur un ou plusieurs étages pour exécuter les différentes tâches du Flow-Shop. Ces problèmes présentent alors une difficulté supplémentaire par rapport aux problèmes sans flexibilité des ressources. En effet, la machine qui sera utilisée pour exécuter une opération n'est pas connue d'avance, mais doit être sélectionnée parmi un ensemble donné pour construire une solution au problème.

Dans un problème d'ordonnancement de type Flow-Shop hybride, un ensemble de N jobs, $J = \{J_1, J_2, ..., J_N\}$, doit être traité dans un atelier de production composé de K étages, $E = \{E_1, E_2, ..., E_K\}$. Chaque étage E_k contient un certain nombre de machines parallèles identiques, avec $(k = 1, 2, ..., K)$. Tous les jobs exigent le même ordre des opérations, $O_i = \{O_{i1}, O_{i2}, ..., O_{iK}\}$, qui doivent être exécutées selon le même processus de fabrication. Une machine ne peut appartenir qu'à un seul étage et ne peut effectuer qu'une seule opération à la fois. Chaque job ne peut avoir qu'une seule opération en cours de réalisation simultanément et doit être traité par une seule machine de l'étage E_k, sans interruption.

Le problème du flow shop à m étages avec machines dédiées est un autre modèle de flow shop qui également connu sous le nom du flow shop de différenciation [59]. La différence principale par rapport à un flow shop hybride est dans l'existence des machines dédiées au lieu des machines parallèles. Ces machines dédiées ne sont pas identiques et chaque machine est dédiée à un type de tâches et ne peut traiter que les travaux de ce type parmi tout l'ensemble des travaux. On peut trouver la configuration du flow shop avec machines dédiées dans de nombreuses organisations industrielles telles que les industries du bâtiments, la fabrication des systèmes électroniques, la fabrication des films photographiques, l'industrie du papier et beaucoup d'autres comme les ateliers de peinture et d'imprimerie, les laboratoires de chimie et les industries pharmaceutiques [35].

On note que la majorité des travaux dans la littérature s'intéressent essentiellement aux problèmes ne contenant qu'un seul étage avec des machines dédiées, tandis que dans les situations réelles, nous avons tendance à trouver des machines dédiées sur plusieurs étages.

1.4 Systèmes Flexibles de Production

Des systèmes de production de plus en plus complexes sont mis en œuvre dans le milieu industriel, afin de satisfaire au mieux, qualitativement et/ou quantitativement, et dans un contexte économique donné, une demande provenant d'un marché incertain [49].

1.4.1 Définition du SFP

Il existe plusieurs définitions d'un système flexible de production en fonction des spécificités considérées. Les SFP sont définit relativement à leurs composants comme suit :

Définition 1. *Un SFP combine un ensemble de machines de production à commande numérique inter reliées par un système automatisé de transport/manutention (T/M) et une infrastructure informatique permettant la gestion et le contrôle du système.*

Pour faire face à une concurrence internationale de mieux en mieux organisée et à un marché de plus en plus fluctuant, les industriels ont développé la flexibilité de leurs systèmes de production afin d'améliorer leur productivité et leur réactivité, tout en réduisant le plus possible les coûts de production et en augmentant la qualité des produits fabriqués.

1.4.2 Différents types du SFP

L'existence de nombreuses définitions d'un SFP, il y a un consensus de la communauté internationale pour admettre la présence des trois sous-systèmes suivants dans un SFP :

- un système de fabrication composé de machines à commande numérique capables de changements autonomes d'outils. Ces outils peuvent être rangés dans un magasin local (i.e. pour chaque machine), ou bien dans un magasin central auquel cas il faut prévoir un système de gestion de ces outils en fonction des demandes des machines. Le système de fabrication a un impact direct sur tous les critères de flexibilité cités précédemment, et plus particulièrement sur la versatilité, ou flexibilité, des machines ;
- un système automatisé de transport/manutention/stockage : réseau de chariots autoguidés, tapis roulants avec embranchements, robots dédiés au déplacement des pièces. Ce système assure les phases de T/M de pièces entre machines et/ou zones de stockage, ces dernières pouvant être locales ou centrales. Ce système est très lié au troisième critère de flexibilité : la nature du réseau des mouvements de produits entre unités de production ;
- un système informatique de contrôle assurant des fonctions de planification de la production, suivi de la production afin de surveiller et de piloter le déroulement de la production (gestion des opérations, transmission des programmes d'usinage aux machines, détection des pannes).

1.5 Théorie de la complexité

La théorie de la complexité s'intéresse à l'étude formelle de la difficulté des problèmes en informatique. On se pose alors la question fondamentale de savoir : "entre différents algorithmes réalisant une même tâche, quel est le plus performant ? ". Pour comparer les algorithmes entre eux, on pourrait calculer le temps mis par chaque algorithme pour s'exécuter, une fois implémenté sur une machine. Cette comparaison entre deux approches ou algorithmes est souvent une chose délicate, car on se contentait au départ de ne tenir compte que du temps d'exécution pour décider de la performance des deux approches. Or cette manière est relativement biaisée car, d'autres paramètres rentrent en jeu comme la vitesse d'horloge de la machine sur laquelle s'exécutent l'une ou l'autre approche. Pour une comparaison objective, il est impératif d'adopter des critères indépendants du matériel et des logiciels utilisés pour tester l'une ou l'autre méthode : c'est ainsi que la théorie de complexité est née [8].

1.5.1 Complexité algorithmique

Dans [49] on trouve la définition suivante pour ce qui concerne la complexité d'un algorithme : " La complexité d'un algorithme A est une fonction $(C_A(n))$, donnant le nombre d'instructions caractéristiques exécutées par A dans le pire des cas, pour une donnée de taille n." La complexité algorithmique exprime donc une fonction du nombre d'opérations élémentaires de calcul effectuées par la méthode ou par l'algorithme de résolution en fonction du nombre des données du problème traité.

1.5.1.1 Notations de la complexité algorithmique

La théorie de la complexité s'intéresse, entre autres, à l'ordre de grandeur de la complexité dans le pire des cas qui est dite en grand O. Pour estimer la complexité d'un algorithme, en grand O, il suffit de borner le nombre d'instructions élémentaires qu'il engendre. Cette borne est exprimée en fonction de la taille des données n. Il existe plusieurs types de complexité qui sont donnés dans le Tableau suivant :

Complexité	Notations
Complexité constante (indépendante de la taille des données)	$O(1)$
Complexité logarithmique	$O(\log(n))$
Complexité linéaire	$O(n)$
Complexité quasi linéaire	$O(n\log(n))$
Complexité quadratique	$O(n^2)$
Complexité cubique	$O(n^3)$
Complexité polynomiale	$O(n^p)$
Complexité quasi polynomiale	$O(n^{\log(n)})$
Complexité exponentielle	$O(2^n)$
Complexité factorielle	$O(n!)$

TABLE 1.1 – Notations de la complexité

1.5.2 Complexité des problèmes

L'expérience montre que certains problèmes sont plus faciles à résoudre que d'autres. Une théorie de la complexité a été développée et permet mathématiquement de classer les problèmes faciles et difficiles en deux classes : les classes P et NP. La complexité problématique est liée à la difficulté du problème à résoudre et au nombre d'opérations élémentaires qu'un algorithme déterministe peut effectuer pour trouver l'optimum en fonction de la taille du problème [28]. Selon son degré de complexité, un problème peut appartenir à l'une des quatre classes suivantes :

- **Les problèmes les plus difficiles :** ce sont les problèmes pour lesquels il n'existe aucune méthode de résolution. Ils sont également dits indécidables.
- **Les problèmes de la classe P :** dits polynomiaux s'il existe un algorithme de complexité polynomiale pour leur résolution.
- **Les problèmes de la classe NP :** C'est l'abréviation pour " non deterministic polynomial time". Cette classe renferme tous les problèmes de décision dont on peut associer à chacun d'eux un ensemble de solutions potentielles (de cardinal au pire exponentiel) tel qu'on puisse vérifier en un temps polynomial si une solution potentielle satisfait la question posée.
- **Les problèmes NP-complets :** La théorie de la NP-complétude concerne la reconnaissance des problèmes les plus durs de la classe NP. La notion de la difficulté d'un problème qui est introduite dans cette classe est celle d'une classe de problème

qui sont équivalents en ce sens que si l'un d'eux est prouvé être facile alors tous les problèmes de NP le sont. Inversement, si l'un d'eux est prouvé être difficile, alors la classe NP est distincte de la classe P.

1.5.2.1 Prouver la NP-complétude d'un problème

La démonstration d'appartenance d'un problème de reconnaissance à la classe des problèmes NP-complets est très importante puisque, une fois cette démonstration faite, on peut considérer comme peu vraisemblable l'existence d'un algorithme polynomial pour résoudre ce problème. Prouver qu'un problème de décision P est NP-complet consiste en les quatre étapes suivantes :

1. Montrer que P est dans NP.
2. Choisir $P\prime$, un problème NP-complet connu.
3. Construire une transformation f de $P\prime \mapsto P$.
4. Prouver que f est une réduction polynomiale.

1.5.2.2 Hiérarchie de complexité

Suite à la hiérarchie de la complexité, et puisque les problèmes d'ordonnancement sont aussi des problèmes d'optimisation, la notion de réduction introduite précédemment peut leur être appliquée. Une hiérarchisation des problèmes d'ordonnancement est donc possible.

Plusieurs hiérarchies de complexité sont possibles suivant la ou les caractéristiques étudiées à savoir environnement des machines, relations de précédence, contraintes sur les durées opératoires et relations entre les critères comme présenté par la Figure 1.1. A titre d'exemple, une hiérarchie générale utile à connaître est celle des problèmes ayant des configurations machines différentes dans des environnements identiques. Comme on peut le constater, la configuration la plus simple est celle qui est à une machine, puis l'ajout d'autres machines et de différentes contraintes augmentent la complexité.

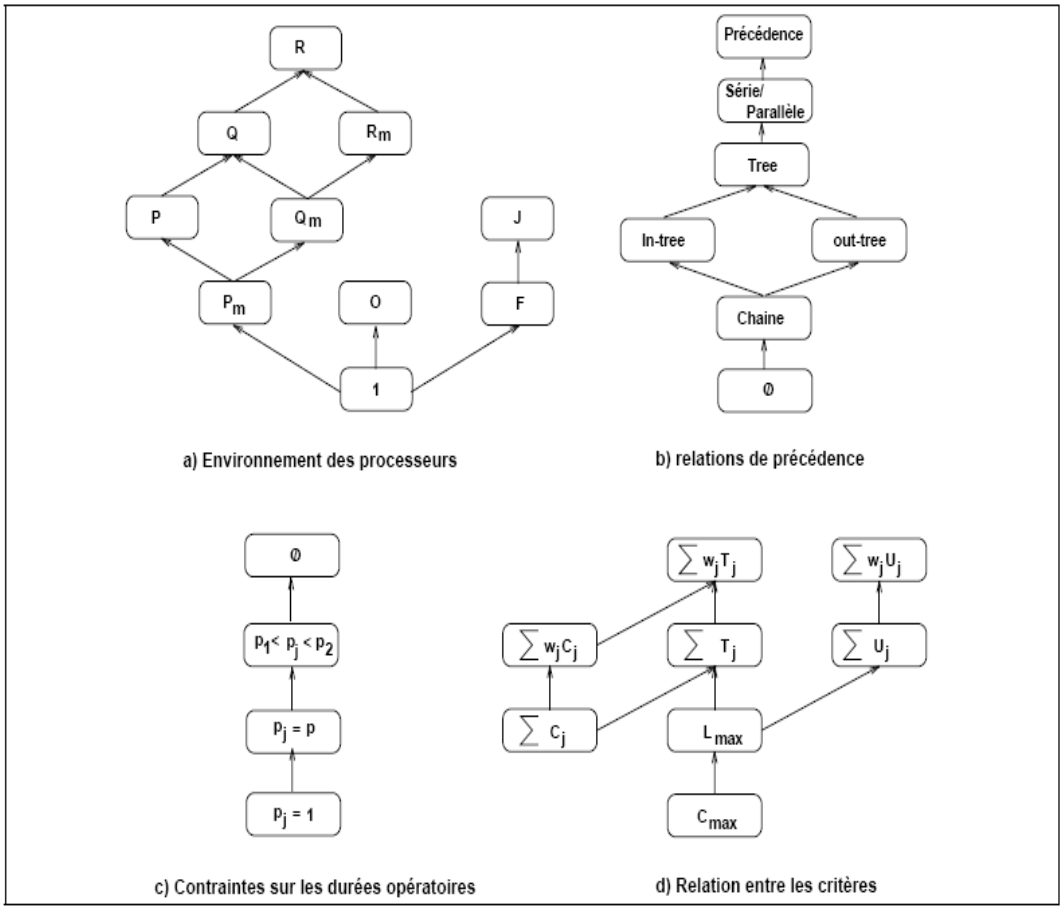

FIGURE 1.1 – Hiérarchie de complexité entre différents problèmes

1.6 Problème d'optimisation

Un problème d'optimisation se définit comme la recherche, parmi un ensemble de solutions possibles S (appelé aussi espace de décision ou espace de recherche), de la (ou des) solution(s) x^* qui rend(ent) minimale (ou maximale) une fonction mesurant la qualité de cette solution. Cette fonction est appelée fonction objectif ou fonction coût. Si l'on pose $f : S \mapsto R$ la fonction objectif à minimiser (respectivement à maximiser) à valeurs dans R, le problème revient alors à trouver l'optimum $x^* \in S$ tel que $f(x^*)$ soit minimal (respectivement maximal). Lorsque l'on veut résoudre un problème d'optimisation, on recherche la meilleure solution possible à ce problème, c'est-à-dire l'optimum global [8]. Cependant, il peut exister des solutions intermédiaires, qui sont également des optimums, mais uniquement pour un sous-espace restreint de l'espace de recherche : on parle alors d'optimums locaux. Cette notion est illustrée dans la Figure 1.2. La seule hypothèse faite sur S est qu'il s'agit d'un espace sur lequel est définie une notion de voisinage.

Cette hypothèse est nécessaire pour définir la notion de solutions locales du problème d'optimisation. On peut alors définir un optimum local (relativement au voisinage V) comme la solution x^* de S telle que $f(x^*) \leq f(x), \forall x \in V(x^*)$.

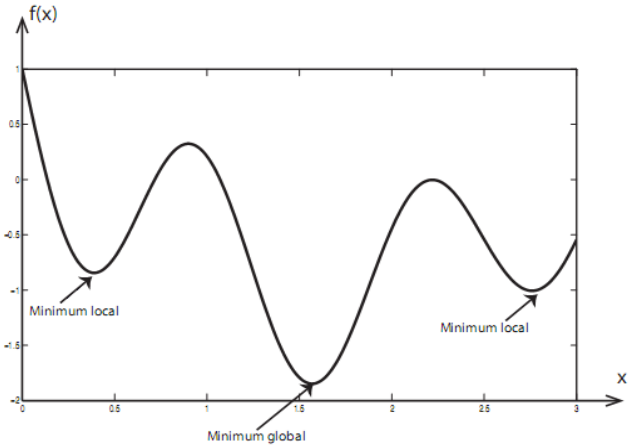

FIGURE 1.2 – Différence entre un optimum global et des optima locaux

Les problèmes d'ordonnancement sont, en général, des problèmes d'optimisation puisque leur objectif est de minimiser (ou maximiser) une fonction objectif tout en respectant certains critères.

1.7 Méthodes de résolution

Quant aux méthodes de résolution, de nombreux efforts ont été mis au point afin d'obtenir des ordonnancements optimaux. Les différentes méthodes de résolution ont plusieurs caractéristiques importantes. La première caractéristique est la qualité de la solution. La deuxième caractéristique est le temps de calcul.

1.7.1 Méthodes exactes

Il existe trois types de méthodes exactes à savoir la programmation dynamique, la méthode de séparation et évaluation et la résolution à partir d'une modélisation analytique. Le temps de calcul de ces méthodes est exponentiel ce qui explique qu'elles ne sont utilisables que sur des problèmes de petites tailles. Parmi ces méthodes exactes, on cite essentiellement :

1.7.1.1 Modélisation analytique

La modélisation analytique d'un problème permet, non seulement de mettre en évidence l'objectif et les différentes contraintes du problème, mais également de le résoudre. L'idéal est de modéliser le problème sous la forme d'un programme linéaire dont les variables sont réelles. Dans ce cas, il existe un bon nombre de solveurs permettant de le résoudre. Dès que le problème comporte des coûts fixes ou des décisions nécessitant l'utilisation de variables entières, les modèles deviennent beaucoup plus difficiles à résoudre. En ce qui concerne la programmation linéaire en nombres entiers (PLNE), plusieurs modèles proposés pour la résolution de problèmes d'optimisation combinatoire et d'ordonnancement utilisent la programmation linéaire, qui constitue l'une des principales pistes pour ce type de problèmes.

Actuellement, plusieurs langages de modélisation existent. Ils permettent d'écrire les programmes linéaires de façon formelle, proche de l'écriture mathématique et qui peuvent être exécutés par des solveurs tels que CPLEX, OSL, XPRESS...etc. Malheureusement, tous les problèmes ne peuvent être résolus par cette approche car la résolution de programmes linéaires avec des variables entières, par exemple, demande souvent trop de temps de calcul [28].

1.7.1.2 Programmation dynamique

Introduite par Richard Bellman [6], la programmation dynamique permet de résoudre tout problème d'optimisation dont la fonction objectif se décrit comme la somme de fonctions monotones non-décroissantes des variables. Il s'agit d'une méthode exacte consistant à plonger le problème proposé dans un problème plus général, dépendant de paramètres entiers, le problème initial correspondant à une valeur précise de ceux-ci. On essaie alors de résoudre le problème général par récurrence sur ces paramètres entiers. La programmation dynamique utilise le principe d'optimalité que l'on peut énoncer de la façon suivante : dans une séquence optimale de décisions, quelle que soit la première décision prise, les décisions suivantes forment une sous-suite qui est optimale, compte tenu des résultats de la première [28].

Méthodes approchées

Dans ce qui suit, nous introduisons une panoplie de méthodes approchées qui permettront de trouver des solutions et de les améliorer. Pour trouver une "bonne" solution dans un délai raisonnable, il existe deux types d'algorithmes qui peuvent être développés : des heuristiques et des métaheuristiques. Les heuristiques et les métaheuristiques sont toutes

les deux des méthodes d'approximation, en ce sens qu'elles produisent des solutions pas forcément exactes. La différence réside dans le fait que la première citée génère une seule solution suivante une ou des règles, et la deuxième en génère plusieurs et ce, d'une manière itérative.

Le mot heuristique vient du grec eurisko qui signifie "je trouve " d'où le célèbre Eureka d'Archimède. Une heuristique est un algorithme qui fournit rapidement (en temps polynomial) une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation NP-difficile. On oppose les méthodes approchées aux méthodes exactes, qui trouvent toujours l'optimum si on leur en laisse le temps (énumération complète, méthodes arborescentes, programmation dynamique). Les approches heuristiques, contournent le problème de l'explosion combinatoire en faisant délibérément des impasses et n'explorent qu'une partie de l'espace des combinaisons.

Le mot métaheuristique est dérivé de la composition de deux mots grecs : méta, du grec $\mu\epsilon\tau\alpha$ signifiant "au-delà" (ou " à un plus haut niveau"). En effet, ces algorithmes se veulent des méthodes génériques pouvant optimiser une large gamme de problèmes différents, sans nécessiter de changements profonds dans l'algorithme employé. En utilisant de telles méthodes, de très grandes instances de problèmes difficiles (par exemple NP-difficilee) peut être résolu d'une manière approximative dans un temps relativement court. En outre, différents procédés peuvent être combinés de nombreuses manières les uns aux autres sous la forme de techniques hybrides [8].

1.7.2 Méthodes heuristiques constructives

Le principe de ces méthodes est de construire une solution à partir des données initiales. Ce sont des méthodes itératives où, à chaque itération, on complète une solution partielle. En ordonnancement, on peut développer de nombreuses méthodes constructives. Par exemple celles qui à chaque itération placent successivement toutes les opérations d'un travail, celles qui au contraire à chaque itération placent une opération au plus sur une des machines. Les méthodes constructives se distinguent par leur rapidité et leur grande simplicité. Le principal défaut de ces méthodes réside malheureusement dans la qualité des solutions obtenues. Le fait de vouloir opérer à tout prix le meilleur choix à chaque étape est une stratégie dont les effets peuvent être catastrophiques à long terme. La plupart de ces méthodes sont des algorithmes gloutons car elles considèrent les éléments (travaux) dans un certain ordre sans jamais remettre en question un choix une fois qu'il a été effectué. D'un principe très simple, ces méthodes permettent de trouver une solution très rapidement. Un premier type de telles méthodes consiste, dans une première

phase, à calculer une liste qui donnera l'ordre de prise en compte des éléments. Cette liste construite à priori, à partir d'un critère bien défini, n'est plus remise en cause au cours de l'ordonnancement. La deuxième phase de l'algorithme se réduit à considérer les éléments (travaux) dans l'ordre de la liste pour construire l'ordonnancement. Un deuxième type de ces méthodes consiste à choisir, au cours de la construction, l'affectation d'un travail à une machine en utilisant des règles de priorité [28]. Le problème de la détermination de règles de priorité performantes a suscité et suscite toujours une littérature très abondante. Un très grand nombre de règles ou combinaisons de règles ont été définies et testées. L'analyse de cette littérature montre que, à part quelques résultats généraux bien connus, il est extrêmement délicat de vouloir définir le domaine d'application (type d'atelier, critère d'évaluation) d'une règle donnée. Les règles de priorité étant souvent aisées à implémenter, cette méthode est très répandue et permet d'obtenir de bons ordonnancements en un temps raisonnable. Chaque règle appliquée séparément sur la file d'attente d'une ressource est généralement performante pour un critère particulier. Cependant, des approches permettant de combiner, par agrégation, différentes règles élémentaires afin de réaliser des compromis pour tenter de satisfaire des objectifs multiples ont également été développées.

Les règles de priorité peuvent être classées selon plusieurs critères [43] :

- Elles peuvent être locales ou globales. Une règle est locale si elle ne prend en compte que les informations locales à la file d'attente d'une ressource. Tandis qu'elle est globale si elle prend en compte, par exemple, la charge sur la machine suivante.
- Elles peuvent être statiques ou dynamiques. Une règle est statique si la valeur de la priorité reste invariante pendant le séjour du travail dans la file d'attente. Une règle dynamique peut conserver l'ordre relatif entre deux travaux, mais la valeur de la priorité doit être remise à jour dès qu'un nouveau travail arrive.
- Elles peuvent être classées selon les informations prises en compte par celles ci. On peut distinguer les règles qui prennent en compte les durées opératoires, les dates de fin au plus tard, combinent les deux, n'utilisent ni l'un ni l'autre,...etc.
- Elles peuvent être classées selon leur complexité. Cette complexité peut être due à la combinaison pondérée de règles, utilisation de paramètres à régler, prise en compte des spécificités de l'atelier,...etc.

Par exemple, parmi les règles statiques, la règle SPT (Shortest Processing Time) qui consiste à séquencer les tâches dans l'ordre croissant des durées opératoires, la règle WSPT (Weighted Shortest Processing Time) qui consiste à séquencer les tâches dans l'ordre croissant des quotients durées/poids (pi/wi) et la règle EDD (Earliest Due Date) qui consiste à séquencer les tâches dans l'ordre croissant des dates de fin au plus tard.

1.7.3 Méthodes amélioratrices

Le principe de ces méthodes n'est plus de construire un ordonnancement à partir des données initiales du problème, mais de modifier le résultat d'une solution admissible en vue d'améliorer la valeur de la fonction objectif. La plupart de ces méthodes utilisent la notion de voisinage de solution, on parle aussi de méthodes par voisinage. A partir d'une solution initiale (générée aléatoirement ou avec une heuristique), une méthode par voisinage ou encore méthode de recherche locale réalise un processus itératif qui consiste à remplacer la solution courante par l'un de ses voisins en tenant compte de la fonction objectif [28].

Les méthodes de recherche locale ou métaheuristiques à base de voisinages s'appuient toutes sur un même principe. A partir d'une solution unique x_0, considérée comme point de départ (et calculée par exemple par une heuristique constructive), la recherche consiste à passer d'une solution à une solution voisine par déplacements successifs. L'ensemble des solutions que l'on peut atteindre à partir d'une solution x est appelé voisinage $N(x)$ de cette solution. Déterminer une solution voisine de x dépend bien entendu du problème traité.

De manière générale, les opérateurs de recherche locale s'arrêtent quand une solution localement optimale est trouvée, c'est à dire quand il n'existe pas de meilleure solution dans le voisinage. Mais accepter uniquement ce type de solution n'est bien sûr pas satisfaisant. Parmi ces méthodes, on trouve les méthodes de recuit simulé, la recherche tabou, la recherche locale itérée, la recherche à voisinage variable,...[70].

Dans ce qui suit, on cite essentiellement :

1.7.3.1 Méthode de descente

La méthode de descente (DM : Descent method) est l'une des méthodes les plus simples de la littérature. Elle est également appelée *hill climbing* dans les problèmes de maximisation. Son principe consiste, à partir d'une solution initiale, à choisir à chaque itération un point dans le voisinage de la solution courante qui améliore strictement la fonction objectif. Il existe plusieurs moyens de choisir ce voisin : soit par le choix aléatoire d'un voisin parmi ceux qui améliorent la solution courante (first improvement) ; soit en choisissant le meilleur voisin qui améliore la solution courante (best improvement). Dans tous les cas, le critère d'arrêt est atteint lorsque plus aucune solution voisine n'améliore la solution courante.

L'algorithme suivant présente une méthode de descente simple. A partir d'une solution

initiale x, on choisit une solution x' dans le voisinage $N(x)$ de x. Si cette solution est meilleure que x, $(f(x') < f(x))$ alors on accepte cette solution comme nouvelle solution x et on recommence le processus jusqu'à ce qu'il n'y ait plus aucune solution améliorante dans le voisinage de x.

Algorithm 2 Algorithme de descente simple

1: initialise : Trouver une solution initiale x
2: **répéter**
3: *Recherche dans le Voisinage* : Trouver une solution $x' \in N(x)$
4: **si** $f(x') < f(x)$ alors
5: $x \leftarrow x'$
6: **fsi**
7: **jusqu'à** $f(y) \geq f(x), \forall y \in N(x)$

Cette méthode est évidemment sujette à de nombreuses critiques. Elle se base sur une amélioration progressive de la solution et donc restera bloquée dans un minimum local dés qu'elle en rencontre un. Il existe de manière évidente une absence de diversification. Une amélioration de cet algorithme consiste à redémarrer plusieurs fois, lorsqu'un optimum local est trouvé, à partir d'une nouvelle solution générée aléatoirement. On parle alors d'algorithme de descente avec relance (multiple start random hill climbing) [70].

1.7.3.2 Recherche avec tabous

Le terme tabu search est apparaît la première fois par Glover [30] en 1986. La méthode de recherche avec tabou, ou simplement recherche tabou (TS : Tabu Search) utilise explicitement l'historique de la recherche, à la fois pour échapper aux minima locaux et pour mettre en œuvre une stratégie d'exploration. Sa principale caractéristique est en effet basée sur l'utilisation de mécanismes inspirés de la mémoire humaine.

La recherche tabou ne génère qu'une seule solution x' aléatoirement dans le voisinage $N(x)$ de la solution courante x, la méthode tabou, dans sa forme la plus simple, examine le voisinage $N(x)$ de la solution courante x. La nouvelle solution x' est la meilleure solution de ce voisinage (dont l'évaluation est parfois moins bonne que x elle-même). Pour éviter de cycler, une liste tabou (qui a donné le nom à la méthode) est tenue à jour et interdit de revenir à des solutions déjà explorées. Dans une version plus avancée de la méthode tabou, on peut voir dans cette recherche une modification temporaire de la structure de voisinage de la solution x permettant de quitter des optima locaux. Le voisinage $N^*(x)$

intégrant ces modifications de structure est régit par l'utilisation de structures de mémoire spécifiques. Il s'agit de mémoire à court terme ou de mémoire à long terme.

La mémoire à court terme correspond à la mise en place d'une liste tabou. La liste contient les quelques dernières solutions qui ont été récemment visitées. Le nouveau voisinage $N^*(x)$ exclut donc toutes les solutions de la liste tabou. Lorsque la structure de donnée correspondant aux solutions est trop complexe ou occupe une grande place mémoire, il est courant de ne garder dans la liste tabou que des informations soit sur les caractéristiques des solutions, soit sur les mouvements. Ce type de mémoire à court terme est aussi appelé *recency-based memory*. En conservant des caractéristiques des solutions ou des mouvements, il est possible alors qu'une solution de bien meilleure qualité ait un statut tabou.

Accepter tout de même cette solution revient à outrepasser son statut tabou, c'est l'application du critère d'aspiration. Si le voisinage d'une solution est très grand, évaluer toutes les solutions de ce voisinage peut-être impossible. Il convient alors de mettre en place des stratégies permettant sa réduction. Les solutions les plus courantes proposent des listes de solutions candidates qui pourraient conduire à des solutions de bonne qualité (*candidate list strategy*). La mémoire à long terme permet d'une part d'éviter de rester dans une seule région de l'espace de recherche et d'autre part d'étendre la recherche vers des zones plus intéressantes. Par exemple, la frequency-based memory ou mémoire à base de fréquence attribue des pénalités à des caractéristiques des solutions plusieurs fois visitées au cours de la recherche. Cette technique simple permet de diversifier la recherche facilement. Par ailleurs, les mouvements ayant conduit à des bonnes solutions peuvent être aussi encouragés. On peut par exemple garder en mémoire une liste de solutions élites que l'on utilisera comme nouveau point de départ quand la recherche deviendra improductive pendant plusieurs itérations consécutives.

Algorithm 3 Algorithme de Recherche Tabou

1: initialiser : Trouver une solution initiale x
2: **répéter**
3: *Recherche dans le Voisinage* : Trouver une solution $x' \in N^*(x)$.
4: *Mise à jour* : de liste tabou.
5: *move* $x \leftarrow x'$.
6: **jusqu'à** critère d'arrêt est satisfait.

L'algorithme 8 présente une version très simplifiée d'un algorithme de recherche tabou. L'avantage de cette présentation, c'est qu'elle n'exclut aucune des extensions possibles

mais évidemment avec un peu moins de clarté. L'alternance entre intensification et diversification est bien respectée. L'intensification réside dans la technique de base qui est l'acceptation d'une solution améliorante et bien sûr par le critère d'aspiration. Dans la mémoire à long terme, la conservation de solutions élites est aussi un facteur d'intensification. Pour l'aspect diversification, le rôle le plus important est joué par la liste tabou elle-même, mais aussi par certains points de la mémoire à long terme (la frequency-based memory par exemple) [70].

1.7.3.3 Recherche à voisinage variable

La recherche à voisinage variable (VNS : Variable neighborhood search) est une métaheuristique et une méthode récente et pourtant très simple, basée sur la performance des méthodes de descente. Introduite par Hansen et Mladenovic en 1997 [63]. Elle est basée sur le principe de changement systématique de voisinage durant la recherche, utiliser plusieurs voisinages successifs quand on se trouve bloqué dans un minimum local.

Avant tout à l'étape d'initialisation, il est nécessaire de définir un ensemble de k_{max} voisinages. Ces voisinages peuvent être choisis arbitrairement, mais souvent une séquence dénotés par $N_k = 1...k_{max}$ est rangée par ordre de taille croissante des voisinages qu'elle utilise. La procédure de VNS se compose de trois étapes : perturbation (shaking), recherche locale et déplacement. On choisit une solution de départ x par heuristique. Ensuite, à partir d'une solution initiale x' choisie dans le premier voisinage $N(x)$ de x, on applique une méthode de descente (ou une autre méthode de recherche locale) jusqu'à arriver dans un minimum local (ou que la recherche locale s'arrête). Si la solution trouvée x'' est meilleure que x alors on recentre la recherche en repartant du premier voisinage, sinon on passe au voisinage suivant. La recherche s'arrête quand tous les voisinages ne sont plus capables d'améliorer la solution. Cet algorithme est efficace si les structures de voisinage sont complémentaires en ce sens qu'un minimum local pour un voisinage n'en n'est pas nécessairement un pour un autre.

La méthode est décrite en détail par Hansen et Mladenovic [63] et un algorithme général de base de cette méthode est proposé dans l'algorithme 4.

Algorithm 4 Algorithme de base de RVV (VNS)

initialiser : Trouver une solution initiale x, $k \leftarrow 1$
répéter
  perturbation : Générer aléatoirement un point $x' \in N_k(x)$.
  recherche locale : Appliquer une recherche locale (x' solution de départ)
  si x'' est meilleur que x alors
  $x \leftarrow x''$ et $k \leftarrow 1$ (centrer la recherce autour de x'')
  sinon
  $k \leftarrow k + 1$ (changer de voisinage)
  fsi
jusqu'à $k = k_{max}$

Dans cet algorithme, la diversification est gérée par deux techniques. La première consiste à choisir dans le voisinage courant une solution aléatoirement (étape shake) et la seconde c'est le changement de voisinage lui-même qui agrandit l'espace de recherche autorisé et donc agrandit le voisinage exploré. L'intensification de la recherche est effectuée par l'appel à une procédure de recherche locale. Le plus simple est d'implémenter une méthode de descente (intensification uniquement), mais parfois on peut souhaiter appliquer à cet endroit une recherche plus évoluée [70].

1.7.3.4 Recherche locale itérée

La recherche locale itérée (ILS :Iterated local search) [72] est une métaheuristique basée sur une idée simple : au lieu de l'application répétée d'une procédure de recherche locale à partir de solutions générées aléatoirement, ILS génère la solution de départ pour la prochaine itération en appliquant une perturbation sur l'optimum local trouvé à l'itération courante. Ceci est fait dans l'espoir que le mécanisme de perturbation fournit une solution située dans le bassin d'attraction d'un meilleur optimum local. Le mécanisme de perturbation est un élément clé de ILS : d'une part, une perturbation trop faible peut ne pas être suffisante pour s'échapper du bassin d'attraction de l'optimum local actuel ; d'autre part, une perturbation trop forte correspond à une recherche locale multi-départs à partir de combinaisons initiales générées aléatoirement. Un critère d'acceptation définit les conditions qu'un nouvel optimum local doit satisfaire afin de remplacer l'optimum local courant.

Algorithm 5 Algorithme de base de ILS

1: initialiser : Trouver une solution initiale x, $k \leftarrow 1$
2: **répéter**
3: *perturbation aléatoire* : Trouver une solution x' proche de x.
4: *recherche locale* : Appliquer une recherche locale (x' solution de départ) et trouver un otimum local x''
5: **si** x'' est accepté (exp. meilleur que x) alors
6: $x \leftarrow x''$
7: **fsi**
8: **jusqu'à** critère d'arrêt est satisfait

Le critère d'acceptation, combiné avec le mécanisme de perturbation, permet de contrôler le compromis entre l'intensification et la diversification. Par exemple, un critère d'acceptation qui favorise l'intensification doit accepter seulement les solutions qui sont des améliorations à la valeur généralement optimale. En revanche, le critère qui favorise la diversification accepte toutes les solutions perturbées, quelle que soit leur qualité [8].

Parmi les méthodes amélioratrices, les méthodes à base de population, comme leur nom l'indique, travaillent sur une population de solutions et non pas sur une solution unique. On peut trouver d'autres noms génériques pour ces méthodes, le plus en vogue étant sans doute algorithmes évolutionnaires. Le principe général de toutes ces méthodes consiste à combiner des solutions entre elles pour en former de nouvelles en essayant d'hériter des "bonnes" caractéristiques des solutions parents. Un tel processus est répété jusqu'à ce qu'un critère d'arrêt soit satisfait (nombre de générations maximum, temps maximum, etc). Parmi ces algorithmes à population, on retrouve deux grandes classes qui sont les algorithmes génétiques et les colonies de fourmis. Les algorithmes génétiques ont beaucoup fait parler d'eux et depuis longtemps. Les algorithmes génétiques qui sont inspirés de la théorie de l'évolution. Ce sont des algorithmes aléatoires, mais au lieu de chercher à améliorer progressivement une seule solution (comme les algorithmes de recherche locale), ils font évoluer une population de solutions qui s'améliore globalement par des techniques de reproduction, de sélection et de mutation. Dans ce qui suit on donne un aperçu des algorithmes génétiques.

1.7.3.5 Algorithmes Génétiques

Il s'agit d'algorithmes d'exploration fondés sur les mécanismes de la sélection naturelle et de la génétique. Contrairement aux méthodes de recherche, les algorithmes génétiques

(Hollande, 1975) fonctionnent avec une population de solutions. Une population d'individus (correspondants à des solutions) évoluent en même temps comme dans l'évolution naturelle en biologie. Une nouvelle population est créée en permettant des solutions pères dans une génération à produire une descendance, qui sont comprises dans la prochaine génération. Pour chacun des individus, on mesure sa faculté d'adaptation au milieu extérieur par *le fitness*. un principe de "la survie du plus apte" est appliqué pour garantir que la qualité globale des solutions augmente à mesure que l'algorithme progresse d'une génération à l'autre. A chaque génération, une nouvelle population est créée en conservant les meilleurs éléments (sélection) après avoir transformé (par mutation ou par croisement) l'ensemble de la population de la génération précédente.

Les algorithmes génétiques s'appuient alors sur trois fonctionnalités :

- **la sélection** qui permet de favoriser les individus qui ont un meilleur fitness (pour nous le fitness sera le plus souvent la valeur de la fonction objectif de la solution associée à l'individu.
- **le croisement** qui combine deux solutions parents pour former un ou deux enfants (offspring) en essayant de conserver les "bonnes" caractéristiques des solutions parents.
- **la mutation** qui permet d'ajouter de la diversité à la population en mutant certaines caractéristiques (gènes) d'une solution.

L'algorithme 6 résume les étapes d'un algorithme génétique simple.

Algorithm 6 Algorithme génétique simple

initialiser : générer une population initiale P de solutions de taille $|P| = n$
répéter
  $P' \leftarrow \emptyset$
  répéter
  selection : choisir 2 solutions x et x' de P
  croisement : combiner les solutions parents x et x' pour former des solutions enfants y et y' avec une grande probabilité
  mutation : muter y et y' avec une petite probabilité
  Ajouter y et y' à P'
  jusqu'à $|P'| = n$
jusqu'à critère d'arrêt est satisfait

La représentation des solutions (le codage) est un point critique de la réussite d'un algorithme génétique. Il faut bien sûr qu'il s'adapte le mieux possible au problème et

à l'évaluation d'une solution. Le codage phénotypique ou codage direct correspond en général à une représentation de la solution très proche de la réalité. L'évaluation d'une solution représentée ainsi est en général immédiate. On peut aussi utiliser un codage indirect (codage génotypique) qui est souvent plus éloigné de la réalité et qui nécessite un algorithme de décodage pour reconstituer une solution valide [70].

1.8 Conclusion

Dans ce chapitre, nous avons présenté des notions de base des problèmes d'ordonnancement d'ateliers et les approches utilisées pour résoudre ce type de problème, combinatoire. Nous avons vu que les méta-heuristiques utilisent la notion de voisinage. L'efficacité d'une méthode dépend en grande partie de son système de voisinage qui doit lui permettre d'explorer rapidement de bonnes solutions. Dans une situation idéale, le système de voisinage ne devrait contenir que des solutions meilleures que la solution courante, ce qui, associé à des mécanismes de diversification donnerait aux méta-heuristiques un pouvoir d'exploration élevé.

Chapitre 2

Flow shop sous contraintes de transport

2.1 Introduction

L'ordonnancement, comme un processus de prise de décision, joue un rôle très important dans la majorité des systèmes de fabrication, de production, de transport et dans d'autres domaines d'industrie. L'utilisation des robots dans les systèmes industriels de production promet une variété d'avantages comme l'augmentation de volume de production et l'amélioration de la qualité de productivité. Dans de nombreux ateliers de fabrication et d'assemblage, un certain nombre d'opérations sur chaque tâche doivent être traitées. Souvent, ces opérations doivent être effectuées sur toutes les tâches dans le même ordre et donc tous les travaux suivent le même cheminement. Cet environnement se désigne par un flow shop (atelier à cheminement unique).

Les systèmes robotiques flow-shop se composent d'un ensemble de robots qui sont responsables pour le transfert des tâches entre des machines consécutives, apparaissent largement dans les systèmes de fabrication automatiques. Dans la plupart des études relatives au flow-shop, l'hypothèse courante est que les temps de transfert des tâches entre des stations consécutives (machines) soient ne sont pas pertinents ou négligeables. Cependant, dans de nombreux cas un robot est nécessaire pour les opérations de transfert. Dans ce chapitre, nous étudions un problème de production flow shop à deux étages dans une cellule robotique avec des machines dédiées au premier étage et une machine commune au deuxième étage. Une machine est appelée dédiée si elle est spécifique à un type de tâches et elle traite uniquement l'ensemble des tâches de ce type.

Les problèmes d'ordonnancement avec des machines dédiées ont de nombreuses applications industrielles telles que les industries chimiques et pharmaceutiques [41], et l'industrie de semi-conducteurs [10]. Notre problème est largement différent de l'environnement

classique d'un flow shop hybride à deux étages en raison de la présence des machines dédiées au premier étage. En outre, la capacité du robot et les temps de transport sont pris en compte.

Ce chapitre est organisé comme ceci : la section suivante présente un bref survol des travaux liés dans la littérature. Section 2.3 est consacrée à la description du problème. Deux formulations mathématiques du problème pour déterminer l'ordonnancement qui minimise le makespan ont été proposées dans la Section 2.4. La Section 2.5 est dédiée au calcul des bornes inférieures. Dans la Section 2.6, trois heuristiques sont présentées pour résoudre le problème général et des tests numériques sont effectués dans la Section 2.7 pour montrer la performance de ces méthodes. Finalement, une conclusion et des perspectives sont présentées dans la dernière section.

2.2 Etat de l'art

L'ordonnancement d'un atelier de type flow shop avec plusieurs machines parallèles par étage, habituellement désigné comme, un flow shop hybride, est un problème combinatoire complexe rencontré dans de nombreuses applications du monde réel. Une grande partie des recherches ont été effectuées sur les problèmes de flow shop hybride à deux étages qui sont connus pour être fortement NP-difficile [34]. O$\widehat{g}$uz et al. (1997) [65] ont étudié le problème de la minimiseation du makespan dans un problème d'ordonnancement flow shop à deux étages avec deux machines dédiées au premier étage et une commune machine au deuxième étage. Chaque tâche doit être traitée sur une machine spécifiée au premier étage en fonction du type de tâches, puis la tâche est traitée sur la machine commune au deuxième étage. Le problème est prouvé NP-difficile et une heuristique est présentée pour le résoudre. Lin (1999) [58] a établi pour le même problème une preuve qu'il est fortement NP-difficile par une réduction polynomiale du problème 3-partition, qui est connu pour être NP-difficile au sens fort.

Yang (2013) [78] a considéré un problème similaire comme dans O$\widehat{g}$uz et al. [65] mais avec plusieurs machines dédiées au premier étage. La complexité des différents sous-problèmes est établi et des procédures optimales de résolution polynomiale sont développées pour certains cas particuliers. Riane et al. (2002) [69] ont étudié un problème de flow shop à deux étages avec une seule machine au premier étage et deux machines dédiées au deuxième étage. L'objectif du problème est de minimiser le makespan. Trois heuristiques et un algorithme de programmation dynamique sont développés.

Une variante de ce problème est considéré par Wang et Liu (2013) [75], dans lequel

le premier étage contient une seule machine commune, et le deuxième étage contient plusieurs machines dédiées. Une heuristique basée sur l'algorithme de branch and bound est proposée pour résoudre le problème. Plusieurs bornes inférieures sont dérivées et quatre heuristiques constructives sont utilisées qui permettent d'obtenir des bornes supérieures initiales.

Hoogeveen et al. (1996) [39] ont montré que le flow shop à deux étages *où* un étage est composé de deux machines et l'autre étage est composé d'une seule machine, par rapport au makespan, est NP-difficile au sens fort même si la préemption est permise.

D'autre part, il y a eu plusieurs articles sur les problèmes d'ordonnancement flow-shop avec contraintes de transport. Le premier papier considérant le transport est probablement Maggu et al. (1981) [61]. Kise (1991) [47] a prouvé la NP-Difficulté du problème de flow shop à deux machines et un seul robot dans le cas où les temps de transport sont identiques et l'objectif est de minimiser le makespan. Panwalkar (1991) [66] a étudié une variante de ce problème à deux machines flow shop et a présenté un algorithme polynomial pour le résoudre. L'ordonnancement d'une cellule robotique où les tâches sont traitées sur deux machines est étudié par Levner et al. (1995) [56]. Le transport de tâches entre les machines est effectué par un robot de transport. La cellule robotique a des limitations sur l'espace intermédiaire entre les machines pour stocker l'encours de tâches. Un algorithme polynômial optimal est proposé pour résoudre le problème. En outre, Lee et Chen [50] ont étudié le problème d'ordonnancement pour le transfert de tâches semi-finies et la livraison des travaux finis dans lesquels la capacité des transporteurs et les temps de transport ont été explicitement pris en compte dans un environnement flow shop. Ils ont établi la complexité de divers problèmes. Hurink et Knust (2001) [40] ont considéré le problème de minimiser le makespan dans un problème de flow shop avec m machines et un seul transporteur. Des nouveaux résultats de complexité sont établis pour des cas particuliers où les temps de traitement ou de transport sont des valeurs constantes.

D'autres travaux liés à des problèmes d'ordonnancement avec transport et machines par batch ont été réalisés par divers chercheurs dans la littérature. Tang et Liu (2009) [73] ont considéré un problème de flow shop à deux machines où une machine discrète est suivie par une machine par batch avec un transporteur entre les machines. L'objectif est de minimiser le makespan. Le problème est formulé comme un modèle de programmation mixte en nombres entiers et a été prouvé qu'il est fortement NP-difficile. Une heuristique est proposée pour résoudre le problème. Behnamian et al. (2012) [5] ont étudié un problème d'ordonnancement flow shop (single-batch) et (batch-single) en prenant en compte le transport des tâches entre les machines. Le problème est formulé comme un modèle de

programmation mixte en nombres entiers pour trouver le makespan optimal.

De nombreux articles se sont focalisé sur les problèmes de flow shop à deux étages avec transport. Zhong et Lv (2014) [83] ont étudié un problème de flow shop à deux étapes avec un seul transporteur entre les étages. Il y a une seule machine au premier étage et deux machines parallèles dans le deuxième étage. Le transporteur ne peut transporter qu'une seule tâche à chaque voyage. Une heuristique rapide est proposée pour résoudre le problème avec une analyse de sa performance. Yao et al. (2012) [80] ont considéré un problème de flow shop hybride à deux étages où une machine discrète est suivie d'une machine par batch. Ils ont analysé la complexité d'une classe de problèmes à deux machines avec des arrivées de tâches dynamiques et des heuristiques sont proposées avec leurs rapports de performance.

Un problème d'ordonnancement à deux étages avec transport et traitement par batch est adressé par Zhu (2012) [84], dans lequel les travaux sont transportés d'une zone d'attente à une machine par batch par plusieurs véhicules auto-guidés dans le premier étage. Chaque véhicule peut transporter au plus une seule tâche à la fois. Chaque batch traité produit un coût de traitement. L'objectif étudié est la minimisation de makespan et le coût de traitement. Un algorithme polynomial est présenté pour un cas spécial et le problème général a été montré fortement NP-difficile. Un algorithme pseudo-polynomial pour obtenir un FPTAS a été également développé. Un problème d'ordonnancement de group-shop avec des temps de transport et des temps de set-up dépendants de la séquence est étudié dans [2]. L'objectif est de trouver un ordonnancement qui minimise le makespan. Le problème d'ordonnancement robotique dans des cellules flow shop hybrides avec blocage est considéré par Elmi et Topaloglu (2014) [26]. Plusieurs types de tâches, machines parallèles indépendantes, plusieurs robots et des contraintes d'admissibilité sont considérés.

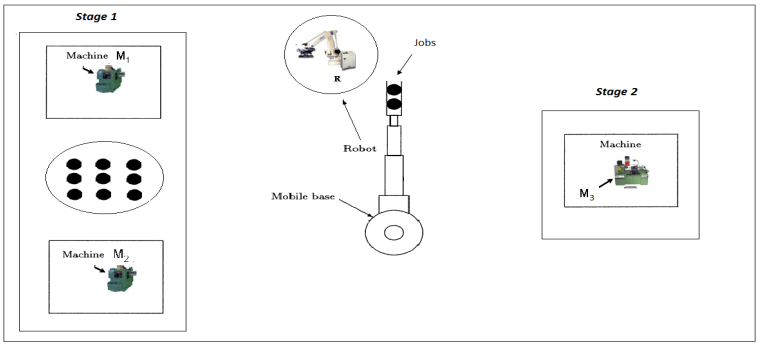

FIGURE 2.1 – Illustration du problème

Le problème considéré dans ce qui suit peut être vu comme une extension du travail

de O$\widehat{g}$uz et al. [65] dans lequel ils traitent un problème de minimisation de makespan dans un flow shop à deux étages avec des machines dédiées au premier étage et une machine commune au deuxième étage. Dans ce chapitre, nous étudions le même problème avec des contraintes additionnelles liées au transport des tâches entre les deux étages par un robot (voir Figure 2.1). La capacité et les temps de transport sont pris en compte. Notre problème peut se produire dans un environnement industriel. Tout dépend des spécifications des produits, les tâches sont traitées sur le premier étage sur l'une des deux machines dédiées. Après les avoir transportées dans des batches par un robot, elles doivent être traitées sur une machine commune se trouvant à l'étage final de fabrication, cette machine peut être une machine d'inspection ou une station de contrôle.

2.3 Enoncé du problème

Nous considérons un ensemble N de n travaux indépendants et non-préemptifs à ordonnancer sur deux étages flow shop. il existe deux machines dédiées au premier étage et une machine commune au deuxième étage. Tous les travaux sont ordonnancés sur le premier étage sans temps mort et chaque machine ne peut traiter qu'une seule tâche à la fois. On suppose que les temps du chargement et de déchargement sont inclus dans les temps de transport et ne sont pas considérés séparément. On suppose aussi que la capacité des espaces de stockage avant et après chaque étage est illimitée. Un robot est chargé de transporter les tâches semi-finies du premier étage vers le deuxième étage par batches. Un batch est un ensemble de tâches qui sont transportées ensemble en un seul voyage. La capacité de transporteur est limitée i.e. il peut transporter jusqu'à c tâches dans un seul voyage $c \leq n$. S'il y a plus de c tâches, les tâches seront transportées selon la règle FCFS (First Come First Served). Un aller-retour de robot nécessite $2t$. Notre objectif est d'ordonnancer les tâches tout en minimisant le makespan (le temps total de traitement).

Nous classons notre problème selon la notation de trois champs utilisée pour les problèmes d'ordonnancement $\alpha|\beta|\gamma$ de Graham et al. [32], α décrit la structure des machines, β donne les caractéristiques de tâches ainsi que les contraintes, et γ définit l'objectif à minimiser.

Nous étendons le schéma de notation de O$\widehat{g}$uz et al. (1997) [65] en suivant la suggestion présentée par Lee et Chen (2001) [50] et nous désignons notre problème par $\mathtt{TF3}|\sigma = 2, \upsilon = 1, \mathtt{c} \geq 1|\mathtt{C}_{\mathtt{max}}$. Dans le champs α, nous utilisons la notation 'TF' pour désigner un problème de flow shop avec transport. Dans le champs β, $\sigma = 2$ signifie qu'il y a deux types de tâches et $\upsilon = 1$ signifie qu'il y a un robot avec une capacité c. Une

autre proposition de notation est $\texttt{TF2(D2,C1)}|v = \texttt{1}, \texttt{c} \geq \texttt{1}|\texttt{C}_{\texttt{max}}$ ('TF2' signifie un flow shop à deux étages avec transport et "(D2,C1)" signifie qu'il existe deux machines dédiées au premier étage et une machine commune au deuxième étage).

Ce problème peut se produire dans un environnement industriel. Tout dépend des spécifications des produits, les tâches sont traitées sur le premier étage sur l'une des deux machines dédiées. Après les avoir transportées dans des batches par un robot, elles doivent être traitées sur une machine commune se trouvant à l'étage final de fabrication, cette machine peut être une machine d'inspection ou une station de contrôle.

Soient N_1 et N_2 une partition de l'ensemble des travaux N i.e. $N_1 \cap N_2 = \emptyset$ et $N_1 \cup N_2 = N$. Soit $N_1 = \{1, 2, ..., n_1\}$ et $N_2 = \{n_1 + 1, n_1 + 2, ..., n_1 + n_2\}$ tel que $n_1 = |N_1|$ est le nombre de travaux de type 1 et $n_2 = |N_2|$ est le nombre de travaux de type 2.

Chaque tâche $i \in N_k$, pour $k = 1, 2$, doit être exécutée sur la machine M_k du premier étage avec des temps de traitement p_i^1, puis transportée vers le deuxième étage pour être traitée sur la machine M_3, avec des temps de traitement associés p_i^2. Le makespan est noté C_{max} i.e. le temps total d'achèvement.

Nous présentons un exemple avec $n = 6$ tâches, où $N_1 = \{1, 2, 3\}$ et $N_2 = \{4, 5, 6\}$. Les temps de transport $t = 3$ et la capacité de robot est $c = 2$. Les temps de traitement de tâches sont présentés dans le Tableau 2.1.

Job i	1	2	3	4	5	6
p_i^1	8	3	2	4	7	5
p_i^2	3	5	9	6	4	1

TABLE 2.1 – Temps de traitement

Le diagramme de Gantt correspondant à la solution optimale est représenté dans la Figure 2.2. Le makespan optimal de ce problème est $C^*_{max} = 33$.

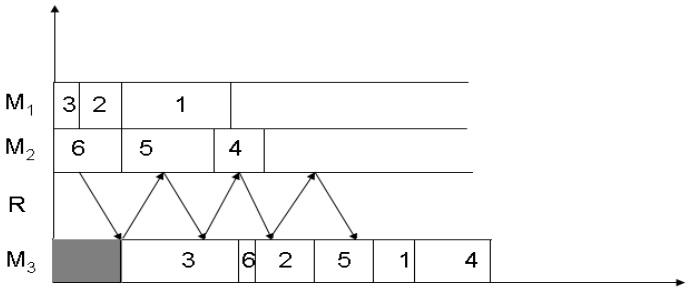

FIGURE 2.2 – Diagramme de Gantt de l'exemple.

Les tâches de type 1 sont traitées sur la machine M_1 selon l'ordre $\{3, 2, 1\}$ et les tâches de type 2 sont traitées sur la machine M_2 dans cet ordre $\{6, 5, 4\}$. Les différents batches transportés associés à cet ordonnancement optimal sont $B_1 = \{3\}$, $B_2 = \{2,\ 6\}$, $B_3 = \{1,\ 5\}$ et $B_4 = \{4\}$. Les dates de transport de ces batches sont respectivement : 2, 8, 14 et 20.

2.4 Programmation mathématique à variables mixtes

Dans cette section, nous proposons deux types de modélisation mathématique pour le problème $TF3|\sigma = 2, \upsilon = 1, c \geq 1|C_{max}(N)$ [20]. Le premier type est basé sur une affectation de travaux sur les machines, le second est basé sur des variables de décision positionnelles. Pour chaque programme mathématique, nous définissons des variables de décision et les paramètres, certaines d'entre elles sont communes, notamment :

- C_{max} est une variable qui définit la valeur de la fonction objectif (makespan).
- M est un très grand nombre,

Nous avons utilisé dans cette formulation les variables de décision suivantes :

t_i^s l'instant de début de traitement de la tâche $i \in N$ à l'étage $s \in \{1, 2\}$.

τ_i l'instant de transport de la tâche $i \in N$.

$x_{ij}^k = 1$ si $t_i^1 < t_j^1$, pour $k \in \{1, 2\}$, $i, j \in N_k$, et 0 sinon.

$y_{ij} = 1$ si $t_i^2 < t_j^2$, $i, j \in N$, et 0 sinon.

$z_{ij} = 1$ si $\tau_i < \tau_j$, $i, j \in N$, 0 sinon.

Le modèle peut être énoncé comme suit :

$$min \quad C_{max}$$

$$s.t.$$

$$t_j^1 \geq t_i^1 + p_i^1 - M(1 - x_{ij}^k) \quad \forall k \in \{1,2\}, i \neq j \in N_k, \tag{2.1}$$

$$t_i^1 \geq t_j^1 + p_j^1 - M x_{ij}^k \quad \forall k \in \{1,2\}, i \neq j \in N_k, \tag{2.2}$$

$$t_j^2 \geq t_i^2 + p_i^2 - M(1 - y_{ij}) \quad \forall\, i \neq j \in N, \tag{2.3}$$

$$t_i^2 \geq t_j^2 + p_j^2 - M y_{ij} \quad \forall\, i \neq j \in N, \tag{2.4}$$

$$x_{ij}^k + x_{ji}^k = 1 \quad \forall k \in \{1,2\}, i \neq j \in N_k, \tag{2.5}$$

$$y_{ij} + y_{ji} = 1 \quad \forall i \neq j \in N, \tag{2.6}$$

$$z_{ij} + z_{ji} \leq 1 \quad \forall i \neq j \in N, \tag{2.7}$$

$$\tau_j - \tau_i \geq 2t z_{ij} - M z_{ji} \quad \forall i \neq j \in N, \tag{2.8}$$

$$\tau_i - \tau_j \geq 2t z_{ji} - M z_{ij} \quad \forall i \neq j \in N, \tag{2.9}$$

$$\sum_{j=1, j\neq i}^{n} (1 - z_{ij} - z_{ji}) \leq c - 1 \quad \forall i \in N, \tag{2.10}$$

$$\tau_i \geq t_i^1 + p_i^1 \quad \forall i \in N, \tag{2.11}$$

$$t_i^2 \geq \tau_i + t \quad \forall i \in N, \tag{2.12}$$

$$t_i^2 + p_i^2 \leq C_{max} \quad \forall i \in N, \tag{2.13}$$

$$x_{ij}^k \in \{0,1\} \quad \forall k \in \{1,2\}, i \neq j \in N_k, \tag{2.14}$$

$$z_{ij},\ y_{ij} \in \{0,1\} \quad \forall i \neq j \in N, \tag{2.15}$$

$$t_i^s,\ \tau_i \geq 0 \quad \forall\, i \in N, s \in \{1,2\}, \tag{2.16}$$

$$\tag{2.17}$$

Les contraintes (2.1) et (2.2) (respectivement (2.3) et (2.4)) sont des contraintes disjonctives qui garantissent qu'un certain ordre existe entre les opérations, si la tâche j est prévue sur une machine avant la tâche i, alors le traitement de la tâche j sur cette machine ne peut pas être lancé avant que le traitement de la tâche i soit terminé. Les contraintes (2.5) et (2.6) expriment que chaque tâche est exécutée avant ou après une autre tâche.

Les contraintes (2.7) expriment que tous les travaux doivent être transportés du premier étage au deuxième étage. Les contraintes (2.8) et (2.9) indiquent que chaque tâche i est transportée au deuxième étage avant ou après une autre tâche j, ou en même temps et montrent que le temps de transport d'un aller-retour du transporteur est $2t$. Les contraintes (2.10) expriment le fait que le nombre de tâches transportées à tout mo-

ment doit être inférieur à la capacité du transporteur. Les contraintes (2.11) rassurent qu'aucune tâche ne peut être transporter au deuxième étage, sauf si son exécution sur le premier étage a été terminée.

Les contraintes (2.12) signifient que le temps de traitement de la deuxième opération d'une tâche ne peut commencer qu'une fois la tâche est arrivée au deuxième étage. Les contraintes (2.13) impliquent que la fin du traitement de chaque tâche est inférieur ou égal au makespan. Les contraintes (2.14)-(2.16) spécifient la non-négativité de t_i^s et τ_i et établissent les restrictions binaires pour x_{ij}^k, y_{ij} et z_{ij}.
Ce modèle a $n_1^2 + n_2^2 + 2n^2$ variables et $3(n_1^2 + n_2^2) + 7n^2 - 7n$ contraintes.

2.5 Bornes inférieures

Pour analyser la performance des heuristiques, nous proposons trois bornes inférieures pour le problème $TF3|\sigma = 2, \upsilon = 1, c \geq 1|C_{max}(N)$, où la charge de travail d'une machine k correspond à la somme des temps de traitement des tâches que la machine doit traiter et elle est notée par W_k.

$$W_1 = \sum_{i \in N_1} p_i^1, \quad W_2 = \sum_{i \in N_2} p_i^1, \quad W_3 = \sum_{i \in N} p_i^2.$$

Ces bornes inférieures sont utilisées dans l'analyse des heuristiques de même que dans les expérimentations [17]. Les deux bornes inférieures LB^1 et LB^2 sont basées sur la charge de travail des machines et la borne inférieure LB^3 est basée sur le temps minimum de transport pour livrer toutes les tâches au deuxième étage.

Proposition 1. *Les trois bornes suivantes sont des bornes inférieures valides :*

- $LB^1 = W_3 + \min\limits_{1 \leq i \leq n} \{p_i^1\} + t.$
- $LB^2 = \max\{W_1, W_2\} + \min\limits_{1 \leq i \leq n} \{p_i^2\} + t.$
- $LB^3 = (\lceil \frac{n}{c} \rceil - 1) \times 2t + t + \min\limits_{1 \leq i \leq n} \{p_i^2\} + \min\limits_{1 \leq i \leq n} \{p_i^1\}.$

Preuve 1. *Pour n'importe quel ordonnancement, la valeur de makespan est plus grande que la charge de travail totale de chaque machine. Par conséquent*

$$C_{max} \geq max\{W_1, W_2, W_3\}.$$

Nous supposons que toutes les tâches sont traitées sur les trois machines sans temps mort. Toutes les tâches doivent être transportées du premier étage au deuxième étage afin d'être traitées sur la machine M_3.

Pour prouver la validité de la borne inférieure LB^1, soit J_1 la première tâche traitée sur le premier étage, elle sera la première tâche à traiter sur M_3. Alors, le temps d'achèvement $C_{max} \geq p_1^1 + t + W_3$. Puisque nous ne connaissons pas la première tâche traitée sur le premier étage, nous avons donc

$$C_{max} \geq W_3 + \min_{i \in N}\{p_i^1\} + t = LB^1.$$

Pour montrer que LB^2 est borne inférieure valide, Soit C^{12} le temps d'achèvement des tâches dans N_1 et N_2 sur les machines dédiées à l'étage 1, nous avons donc

$$C^{12} = \max\{W_1, W_2\}$$

Maintenant, soit J_k la dernière tâche traitée sur le premier étage. La tâche J_k nécessite t unités de temps pour être transportée au deuxième étage où elle est traitée sur la machine M_3, donc nous avons

$$C_{max} \geq C^{12} + t + p_k^2.$$

De même, puisque nous ne connaissons pas la dernière tâche à traiter sur le deuxième étage, nous avons donc

$$C_{max} \geq \max\{W_1, W_2\} + t + \min_{1 \leq i \leq n}\{p_i^2\} = LB^2.$$

Pour prouver que LB^3 est une borne inférieure, soit J_1 et J_n la première et la dernière tâche respectivement qui seront exécutées. Toutes les tâches doivent être transportées au deuxième étage afin d'être traitées sur la machine M_3.

Comme la capacité de transporteur est égale à c et un aller-retour nécessite $2t$ unités de temps, le temps nécessaire pour transporter toutes les tâches est supérieur ou égal à $(\lceil \frac{n}{c} \rceil - 1) \times 2t + t$.

Les tâches sont transportées au second étage si le traitement de la tâche J_1 est fini sur l'étage 1. En outre, une fois que les tâches ont été transportés, J_n doit être exécutée sur M_3, donc nous avons

$$C_{max} \geq p_1^1 + (\lceil \frac{n}{c} \rceil - 1) \times 2t + t + p_n^2$$

Par conséquent

$$C_{max} \geq (\lceil \frac{n}{c} \rceil - 1) \times 2t + t + \min_{1 \leq i \leq n}\{p_i^1\} + \min_{1 \leq i \leq n}\{p_i^2\}$$

□

Corollaire 1. *$LB = \max\{LB^1, LB^2, LB^3\}$ est également une borne inférieure pour le makespan.*

2.6 Heuristiques

Nous proposons trois heuristiques H_1, H_2 et H_3 pour résoudre le problème $TF3|\sigma = 2, \upsilon = 1, c \geq 1|C_{max}$. Pour prendre en compte les temps de transport, nous utilisons une méthode qui intègre les temps de transport dans les temps de traitement et elle est basée sur les règles SPT et LPT [17]. Cette méthode peut être considérée comme une extension de l'algorithme Panwalkar [66] qui résout une variante de problème flow shop à deux machines avec des temps de transport entre les machines. Nous construisons les différents batches grâce à une procédure (Voir la Figure 2.3) qui sera détaillée dans ce qui suit.

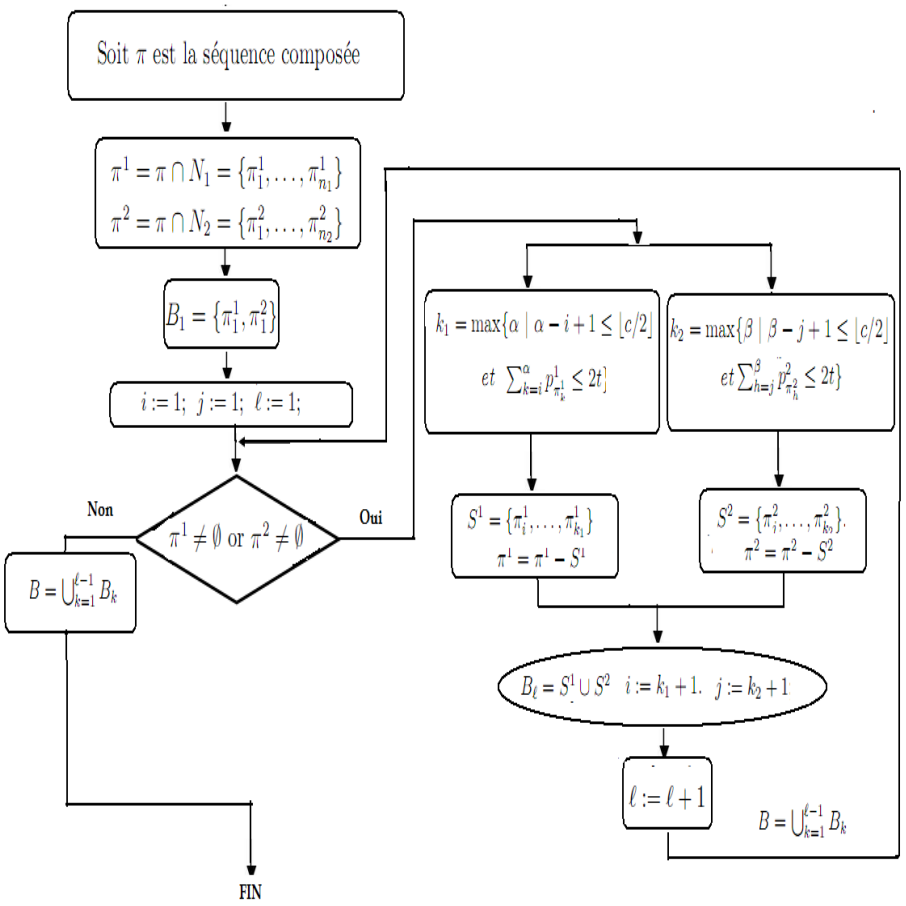

FIGURE 2.3 – Procédure de construction des batches

Algorithm 7 Heuristique H_1

BEGIN
Soit $q_i^1 = \max\{p_i^1, 4t\}$ et $q_i^2 = p_i^2 + p_i^1$, $\forall i \in N$.
Soit $U = \{i : q_i^1 \leq q_i^2\}$ et $V = \{i : q_i^1 > q_i^2\}$;
Trier les tâches de U dans l'ordre non-croissant tel que $q_i^1 \geq q_{i+1}^1$;
Trier les tâches de V dans l'ordre non-croissant tel que $q_i^2 \geq q_{i+1}^2$;
Soit π est la séquence composée par les deux ensembles ordonnés U puis V.
Traiter les tâches dans $\pi^1 = \pi \cap N_1 = \{\pi_1^1, \ldots, \pi_{n_1}^1\}$ sur la machine M_1.
Traiter les tâches dans $\pi^2 = \pi \cap N_2 = \{\pi_1^2, \ldots, \pi_{n_2}^2\}$ sur la machine M_2.
$B_1 = \{\pi_1^1, \pi_1^2\}$ est le premier batch.
$i := 2$; $j := 2$; $\ell := 2$; $\pi^1 := \pi^1 - \{\pi_1^1\}$; $\pi^2 := \pi^2 - \{\pi_1^2\}$;
Tant que $\pi^1 \neq \varnothing$ ou $\pi^2 \neq \varnothing$ **faire**
    $k_1 = \max\{\alpha \mid \alpha - i + 1 \leq \lfloor c/2 \rfloor \ et \ \sum_{k=i}^{\alpha} p_{\pi_k^1}^1 \leq 2t\}$.
    $k_2 = \max\{\beta \mid \beta - j + 1 \leq \lfloor c/2 \rfloor \ and \ \sum_{h=j}^{\beta} p_{\pi_h^2}^2 \leq 2t\}$.
    Soit $S^1 = \{\pi_i^1, \ldots, \pi_{k_1}^1\}$; et $S^2 = \{\pi_j^2, \ldots, \pi_{k_2}^2\}$.
    $B_\ell = S^1 \cup S^2$; et $\ell := \ell + 1$.
    $\pi^1 = \pi^1 - S^1$; et $i := k_1 + 1$.
    $\pi^2 = \pi^2 - S^2$; et $j := k_2 + 1$.
Fin tant que
$B = \bigcup_{k=1}^{\ell-1} B_k$ (B est l'ensemble des batches).
Transporter les batches dans B du premier étage au deuxième étage pour les traiter sur la machine commune.
Traiter les tâches au deuxième étage selon la règle FCFS.
END

Nous décrivons maintenant la première heuristique H_1. Pour définir un ordonnancement, nous considérons d'abord un problème artificiel $F2||C_{max}$ avec des temps de traitement $q_i^1 = \max\{p_i^1, 4t\}$ et $q_i^2 = p_i^2 + p_i^1$, puis nous définissons deux ensembles de tâches U et V tels que $U = \{i : q_i^1 \leq q_i^2\}$ et $V = \{i : q_i^1 > q_i^2\}$. Un schéma illustrait de cette heuristique est présenté dans la Figure 2.4

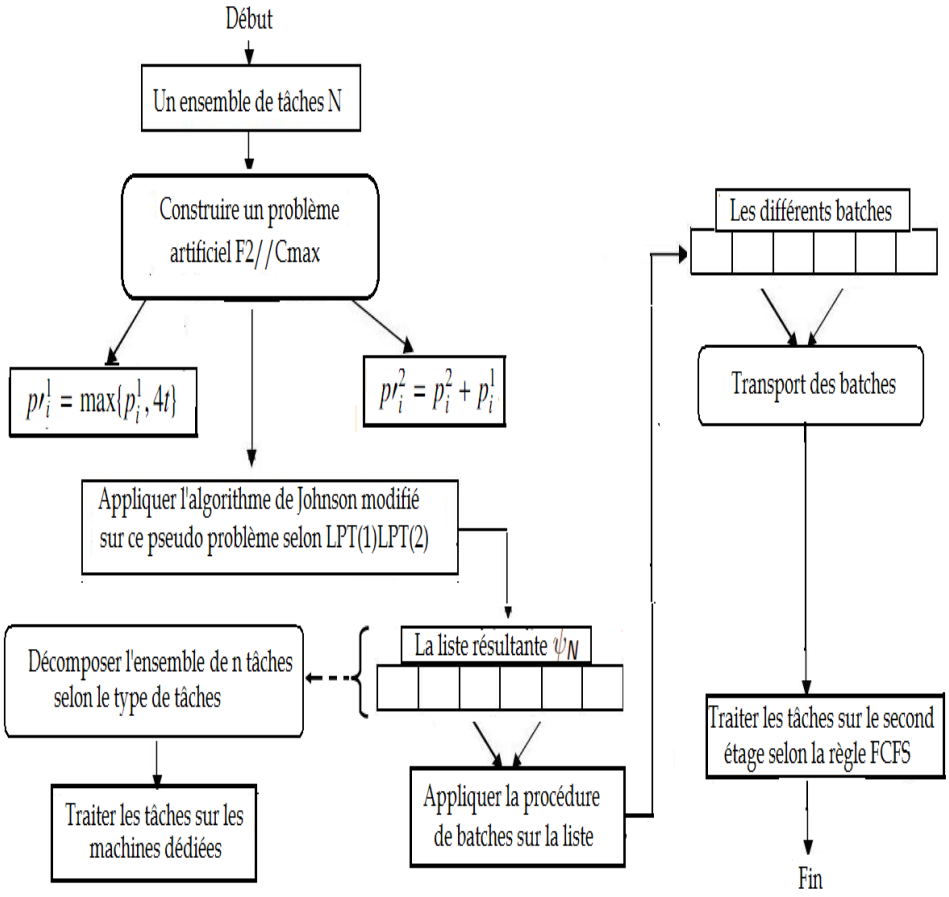

FIGURE 2.4 – Organigramme de l'heuristique H_1

Nous trions ensemble U dans un ordre décroissant en fonction de q_i^1 et nous ordonnons l'ensemble V dans un ordre également décroissant mais en fonction de q_i^2. Nous obtenons la séquence de permutation pi en combinant les deux ensembles ordonnés de tâches U et V. Selon le type de tâches, on divise l'ensemble de tâches π en deux séquences π^1 et π^2. Le premier batch va contenir que deux tâches c'est-à-dire qu'il contiendra la première tâche de π^1 et la première tâche de π^2. Les autres batches seront construits comme suit. Pour chaque type de tâches, à chaque fois, nous choisissons un nombre maximum de tâches tel que la somme de leurs temps de traitement sur la machine dédiée est inférieure ou égale au temps d'un aller-retour $2t$ de transporteur et leur nombre est inférieur ou égal à $\frac{c}{2}$. Une fois ces travaux sont sélectionnés pour chaque type de tâches, ils seront tous réunis dans un seul batch et transportés selon la règle FCFS (premier arrivé premier servi) afin d'être traiter sur la machine commune.

Comme le montre la Figure 2.5, la deuxième heuristique H_2 est une variante de l'heuristique H_1 où les temps de traitement virtuels sont $q_i^1 = p_i^1 + p_i^2$ et $q_i^2 = \max\{p_i^2, 4t\}$. En

outre, les deux ensembles de tâches U et V sont classés différemment. L'ensemble U est trié dans un ordre croissant en fonction de q_i^1 et l'ensemble V est trié également dans un ordre croissant en fonction de q_i^2.

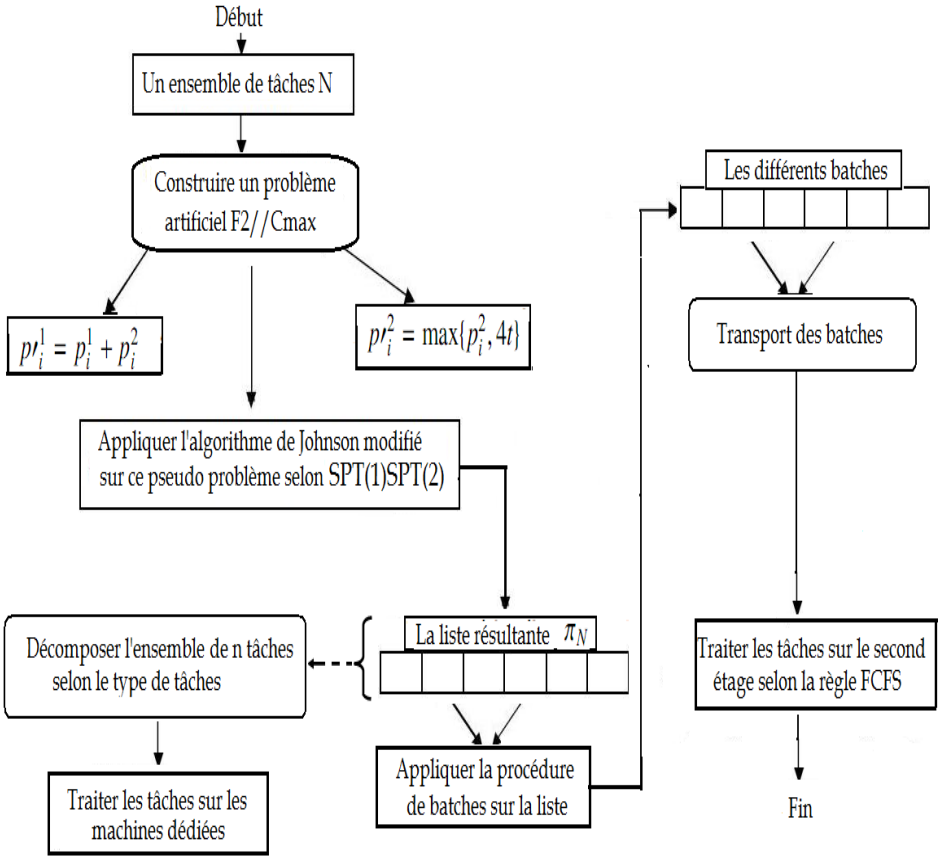

FIGURE 2.5 – Organigramme de l'heuristique H_2

La troisième heuristique H_3 est également une autre variante de l'heuristique H_1 où les temps de traitement virtuels sont $q_i^1 = \max\{p_i^1, 2t\}$ et $q_i^2 = p_i^2$. En outre, les deux ensembles de tâches U et V sont classés différemment. L'ensemble U est trié dans un ordre croissant en fonction de q_i^1 et l'ensemble V est trié dans un ordre décroissant en fonction de q_i^2.

2.7 Expérimentations numériques

Dans la partie expérimentation, des tests numériques ont été menés pour évaluer la performance des deux modèles et des heuristiques proposées. D'après notre connaissance, il n'y a pas de benchmarks dans la littérature pour ce problème. Par conséquent, nous

avons effectué des tests numériques sur un ensemble d'instances générées aléatoirement selon la loi uniforme. Les deux MIP ont été résolu en utilisant IBM Cplex 12,2 Concert Technology, fonctionnant sur un PC Intel Core i7, 2.9 GHz Processeur et 6 Go de RAM dans un temps limité à 1 heure. Nous avons également testé le premier modèle en utilisant des inégalités valides sur la base des trois bornes inférieures présentées dans la section 2.5. Nous avons ajouté ces inégalités valides dans la formulation comme des contraintes.

Pour les instances de petite taille, les temps de traitement p_i^s $(\forall i \in N, s \in \{1, 2\})$ ont été générés selon la loi uniforme sur l'intervalle $[1, 20]$. Les nombres de tâches pour chaque type sont $n_1 = n_2 = 0,5n$. Pour chaque combinaison de paramètres $(n,\ c,\ t)$, nous générons aléatoirement 10 instances et nous donnons les mesures suivantes :

- $TempsM_i$: le temps CPU moyen de modèle "i" en secondes.
- $Temps_{vi}$: temps CPU moyen de modèle 1 avec des inégalités valides.
- $\overline{Dev}\%$: la moyenne de pourcentage de déviation pour les 10 instances de la solution obtenue par l'heuristique avec la meilleure solution trouvée BFS obtenue par les deux modèles, $Dev\% = 100 \times \frac{C_{max}(H)-BFS}{BFS}$.
- $Temps$: le temps CPU moyen de l'heuristique H_i.

Les résultats sont donnés dans le Tableau 2.2.

			Modèle 1		Modèle 2	Heuristique H_1		Heuristique H_2		Heuristique H_3	
n	c	t	$TempsM_1$	$Temps_{vi}$	$TempsM_2$	$Temps$	$\overline{Dev}\%$	$Temps$	$\overline{Dev}\%$	$Temps$	$\overline{Dev}\%$
		1	303.628	6.52	38.25	0.0047	13.14	0.0031	0.225	0.0031	0.31
	1	5	540.912	805.93	32.96	0.0048	13.008	0.0047	05.7	0.0034	1.52
		10	225.743	1129.08	2541.31	0.0031	6.97	0.0016	4.98	0.0031	3.70
		1	240.294	5.52	833.42	0.0078	12.51	0.0031	2.56	0.0031	2.96
10	5	5	502.71	178.35	793.06	0.0032	11.75	0.0032	2.03	0.0031	3.53
		10	589.8	177.85	1394.37	0.0047	9.323	0.0032	2.54	0.0032	5.65
		1	275.028	13.63	1040.36	0.0046	12.51	0.003	2.56	0.0046	2.96
	10	5	289.42	116.53	715.39	0.0048	11.71	0.0047	1.95	0.0031	3.45
		10	554.089	1489.47	1269.34	0.0048	9.728	0.0031	3.46	0.0047	6.55
		1	3600	2994.11	3600	0.0031	7.47	0.0078	0.048	0.0063	0.28
	1	5	3600	3565.53	3600	0.0077	7.51	0.0047	3.596	0.0078	2.35
		10	3600	3600	3600	0.0031	4.14	0.0062	0.96	0.0063	1.91
		1	3600	3240.48	3600	0.0031	7.67	0.0079	0.379	0.0078	0.63
20	5	5	3600	3600	3600	0.0094	7.33	0.0062	0.516	0.0063	2.73
		10	3600	3600	3600	0.0047	7.49	0.0046	1.09	0.0064	4.99
		1	3600	159	3600	0.000	7.67	0.0062	0.379	0.0078	0.58
	10	5	3600	157.5	3600	0.0078	7.33	0.0063	0.516	0.0077	2.73
		10	3600	122.11	3600	0.0063	7.96	0.0062	1.524	0.0079	4.98

TABLE 2.2 – Expérimentations numériques sur des instances de petite taille.

						Heuristique H_1		Heuristique H_2	
n	c	t	BFS	$Gap\%$	LB	$C_{max}(H_1)$	$Temps$	$C_{max}(H_2)$	$Temps$
			553	86.98	526	543	0.00	526	0.015
			554	89.89	510	526	0.016	510	0.016
			475	86.01	431	444	0.015	432	0.00
			583	88.15	551	568	0.00	551	0.016
50	1	1	522	89.22	490	507	0.00	490	0.016
			612	88.87	483	501	0.016	483	0.016
			525	85.47	490	508	0.015	490	0.00
			463	88.42	462	477	0.00	462	0.016
			567	87.12	526	542	0.00	526	0.016
			515	86.98	490	507	0.00	490	0.016

TABLE 2.3 – Expérimentations numériques pour $n = 50$.

Pour un certain nombre de tâches $n = 50$, nous appliquons les deux heuristiques sur 10 instances générées aléatoirement de la même manière que dans le Tableau 2.2. Les résultats sont donnés dans le Tableau 2.3 et nous calculons :

- BFS : La meilleure solution trouvée par les deux modèles dans un temps limité à 1 heure.
- $Gap\%$: Le pourcentage de déviation entre la solution obtenue par Cplex et la borne de Cplex.
- LB : La valeur de la borne inférieure proposée dans la Section 2.5.
- $C_{max}(H_i)$: Le makespan obtenu par l'heuristique H_i.

Le modèle mathématique constitue la meilleure méthode pour résoudre efficacement les instances de petite taille. Nous pouvons observer à partir du tableau 2.2 que pour $n \leq 20$, le temps CPU nécessaire pour obtenir une solution optimale en utilisant les inégalités valides est plus court, ce qui montre ainsi les performances des trois bornes inférieures. Dans la majorité des cas le temps CPU du modèle 1 est plus court que le temps CPU du Modèle 2. Donc le Modèle 1 est meilleur que le Modèle 2.

Nous pouvons également voir que les heuristiques proposées sont performantes. En effet, ces heuristiques fournissent, dans la plupart des cas, de bonnes solutions en un temps CPU très court. L'heuristique H_2 a le pourcentage de déviation le plus faible. La déviation moyenne pour l'heuristique H_1 par rapport aux 180 instances est $9,17\%$ et la déviation moyenne pour l'heuristique H_2 est $1,94\%$. La déviation moyenne pour l'heuristique H_3

par rapport aux 180 instances est $2,87\%$. En outre, l'heuristique H_2 fournit des solutions optimales dans la plupart des cas. En général, l'heuristique H_2 est meilleure que les deux autres heuristiques, car elle donne toujours des solutions de très bonne qualité.

D'après le Tableau 2.3, il devient difficile d'utiliser le modèle (MIP) proposé lorsque le nombre de tâches augmente (i.e. $n \geq 50$). En fait, les instances de grande taille ne peuvent pas être résolues de manière optimale dans un temps de calcul raisonnable.

Pour les instances de grande taille, les tests ont été générés de la façon suivante. Nous avons quatre classes d'instances et les tailles des instances de chaque classe varient dans $n \in \{30, 100, 500, 1000\}$ afin de tester les effets de la variation du nombre de tâches. Les valeurs du nombre de tâches pour chaque type sont $n_1 = n_2 = 0.5n$. Pour chaque valeur de n, 100 instances ont été générées selon la loi uniforme comme est décrit dans le Tableau 2.4.

Classe	c	t	p_i^s
Cl_1	1	$U[1, 10]$	$U[1, 30]$
Cl_2	1	$U[1, 10]$	$U[1, 50]$
Cl_3	1	$U[1, 10]$	$U[1, 100]$
Cl_4	$U[1, 10]$	$U[1, 10]$	$U[1, 100]$

TABLE 2.4 – Classes d'instances

		Heuristique H_1				Heuristique H_2				Heuristique H_3			
Classe	n	Avg.ER	Max.ER	Avg.Time	Max.Time	Avg.ER	Max.ER	Avg.Time	Max.Time	Avg.ER	Max.ER	Avg.Time	Max.Time
	30	6	13	35.75	47	1.4	11.8	38.53	47	0.84	3.19	8,43	16
Cl_1	100	2	5.7	86.26	109	0.6	5.9	83.48	110	0.26	0.88	15,94	32
	500	0.4	1.9	528.81	577	0.1	1.4	504.385	780	0.05	0.19	59,22	63
	1000	0.2	0.9	884.52	999	0.08	0.83	447.76	999	0.04	0.09	118,09	141
	30	5.8	11.3	32.7	47	0.1	2.9	41.05	63	0.44	1.98	9,35	16
Cl_2	100	1.8	2.8	115.58	141	0.08	1.0	130.50	297	0.20	0.70	16,73	32
	500	0.3	0.4	449.31	546	0.01	0.23	492.97	515	0.03	0.14	60,16	78
	1000	0.1	0.2	675.08	999	0.1	0.2	888.9	967	0.02	0.07	125,83	313
	30	5.9	11.4	32.38	47	0.03	1	40.58	47	0.19	1.05	9,07	16
Cl_3	100	1.89	2.9	110.3	125	0.01	0.3	123.58	172	0.08	0.33	16,87	31
	500	0.3	0.4	526.44	983	0.001	0.02	467.52	530	0.02	0.07	59,68	63
	1000	0.1	0.2	914.17	999	0.002	0.07	771.86	999	0.01	0.03	115,27	125
	30	5.86	8.56	36.83	47	0.47	2.77	50.42	78	0.59	2.78	8,4	16
Cl_4	100	1.8	2.2	84.85	109	0.03	0.27	97.67	125	0.12	0.33	17,34	31
	500	0.38	0.42	390.59	593	0.001	0.004	389.21	422	0.02	0.07	62,17	78
	1000	0.19	0.20	762.96	827	0.000	0.000	760.02	873	0.01	0.03	124,08	141

TABLE 2.5 – Expérimentations numériques sur des instances de grande taille.

Pour chaque instance de problème, le makespan $C_{max}(H)$ obtenu par l'heuristique et la borne inférieure LB (voir la section 2.5), ont été calculés. La déviation de performance relative peut être définie comme : Error Ratio (ER)=100 $\times \frac{C_{max}(H)-LB}{LB}$.

Dans la Figure 2.6, on compare les valeurs de la borne inférieure LB et les valeurs de makespan C_i obtenues par l'heuristique H_i.

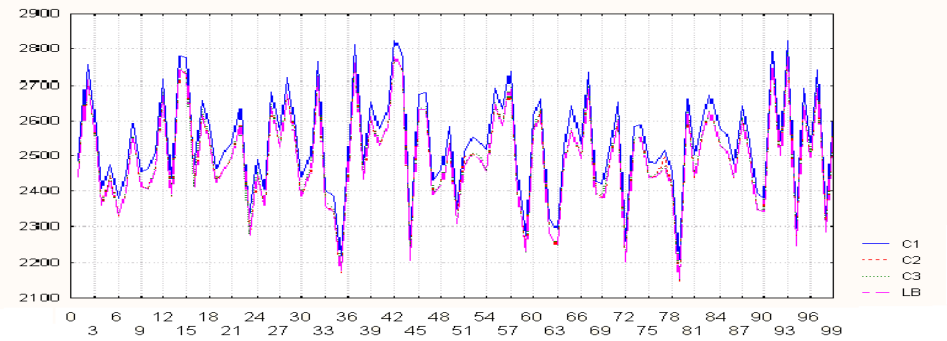

FIGURE 2.6 – Comparaison entre le makespan de l'heuristique H_i et la borne inférieure LB ($Classe : \ Cl_2, \ n = 100$)

Le Tableau 2.5 résume les résultats des calculs expérimentaux pour résoudre les instances de grande taille. Pour chaque combinaison de paramètres, les moyennes et les maximums des déviations, respectivement Avg.ER et Max.ER, ont été utilisés pour évaluer la performance des trois heuristiques. Le temps CPU moyen et maximal requis pour exécuter chaque heuristique (calculé en millisecondes) sont désignés par Avg.Time and Max.Time respectivement. Les trois heuristiques sont exécutées sans aucune limite de temps.

D'après les résultats de calculs des déviations de performance du Tableau 2.5, nous pouvons observer que les petites différences existent entre la borne inférieure et le makespan de l'heuristique. Cela révèle qu'ils sont proches de la solution optimale. Nous notons que les trois heuristiques sont des heuristiques constructives fondées sur des règles de priorité. Par conséquent, elles peuvent trouver des solutions quasi-optimales en un temps CPU très court. Cependant, l'heuristique H_2 est plus performante que les deux autres dans la plupart des cas par l'obtention de la meilleure déviation moyenne dans un temps CPU raisonnable.

Nous pouvons voir également d'après les différents graphes que les déviations ont tendance à diminuer à mesure que le nombre de tâches augmente. L'une des raisons peut être que la borne inférieure augmente en augmentant le nombre de tâches n, mais la différence entre le makespan donnée par les trois heuristiques et la borne inférieure

ne peut pas augmenter en proportion de la taille de n. Ainsi, la performance des trois heuristiques est améliorée considérablement lorsque n augmente. Cette amélioration est importante pour les deux heuristiques. En général, les résultats obtenus pour les différents tests révèlent que l'heuristique H_2 est plus appropriée pour résoudre les instances de moyenne et grande taille.

2.8 Conclusion

Dans ce chapitre, nous avons étudié un problème de flow shop à deux étages avec des machines dédiées au premier étage et une machine commune au deuxième étage. L'objectif est de trouver un ordonnancement de production qui minimise le makespan. Deux modèles de programmation mixte en nombres entiers ont été proposés pour la résolution du problème étudié.

Nous avons également établi de nouvelles bornes inférieures. Comme les variables et les contraintes des modèles proposés augmentent considérablement lorsque le nombre de tâches augmente, et dans le but de résoudre les instances de grande taille, trois heuristiques ont été proposées et leurs performances ont été analysées. Des expérimentations numériques ont été effectuées pour montrer la performance des heuristiques proposées. Ces résultats sont bons d'un point de vue pratique (des solutions de bonne qualité et un temps CPU très raisonnable).

Chapitre 3

Recherche avec tabous multi-objectif

3.1 Introduction

Les problèmes d'ordonnancement avec plusieurs agents ont attiré beaucoup d'attention dans la dernière décennie, depuis les premiers travaux de Baker et Smith [3] et Agnetic et al. [1]. Une abondante littérature a discuté du problème en une seule machine, machines parallèles et flow shop. Cheng et al. [14] étudient la NP-complétude du problème d'ordonnancement multi-agent d'une seule machine, dans lequel l'objectif de chaque agent est de minimiser le nombre total pondéré des tâches tardives. Cheng et al. [15] ont présenté le modèle de problème d'ordonnancement multi-agent sur une seule machine. Wu et al. [76] présentent un algorithme de Branch and bound et deux heuristiques pour un problème d'ordonnancement à une seule machine et deux agents. Cheng et al. [12, 13] développent un algorithme de branch-and-bound et un algorithme de recuit simulé pour le problème d'ordonnancement à une machine et deux agents, d'abord avec des considérations d'apprentissage ensuite avec des dates de disponibilité, respectivement. Lee et al. [53] proposent trois algorithmes génétiques pour le problème à une seule machine à deux agents avec des dates de disponibilité. L'objectif est de minimiser le retard total des tâches du premier agent étant donné que le retard maximum des tâches du second agent ne dépasse pas une borne supérieure. Yin et al. [81] considèrent plusieurs problèmes d'ordonnancement à une seule machine à deux agents avec des dates d'échéances et fournissent des algorithmes en temps polynomial. Wu et al. [77] proposent un algorithme de Branch-and-bound et une recherche tabou pour un problème d'ordonnancement à une seule machine avec détérioration des tâches.

En ce qui concerne les problèmes d'ordonnancement multi-agent à machines parallèles, Li et Yuan [57] considère le problème d'optimisation avec des machines parallèles par batch

illimitées à deux agents et ont fourni un algorithme pseudo-polynomial. Fan et al.[27] étudient la NP-complétude et la solvabilité polynomiale du problème d'ordonnancement à machines parallèles délimitées avec deux agents concurrents et des objectifs différents. Zhao et Lu [82] présentent un FPTAS (fully polynomial time approximation schemes) pour un problème d'ordonnancement sur machines parallèles identiques à deux agents.

L'ordonnancement multi-agent dans un environnement flow shop est rarement pris en considération dans la littérature. Lee et al. [51] étudient le problème d'ordonnancement à deux machines flow shop avec deux agents dont l'objectif est de minimiser le temps total d'achèvement des tâches de premier agent et la contrainte de non retard des tâches de deuxième agent. Lee et al. [52] considèrent le même problème et ont développé un algorithme de Branch-and-Bound et un algorithme du récuit simulé pour minimiser le retard total des tâches du premier agent avec la contrainte de non retard des tâches du second agent. Mor et Mosheiov [64] proposent des algorithmes polynomiaux de résolution pour le même modèle avec deux agents.

Les premiers travaux portants sur l'utilisation de la Recherche Tabou pour l'optimisation multi-objective opéraient par transformation vers le mono-objectif. Des approches ultérieures, étendent les principes de la Recherche Tabou afin de produire une bonne approximation de la frontière Pareto. Dans [38] on se base sur une approche lexicographique. La recherche tabou est utilisée pour résoudre le problème de formation de cellules de fabrication dans les ateliers. La méthode est appliquée pour optimiser une séquence de sous-problèmes mono-objectif sous contraintes. Les objectifs sont traités de façon séquentielle suivant l'ordre d'importance des critères. Le premier sous-problème résolu consiste à optimiser la fonction objectif la plus prioritaire. Le deuxième problème revient alors à optimiser la deuxième fonction objectif la plus importante. Ceci en maintenant satisfaite une contrainte supplémentaire qui consiste à ne pas détériorer, avec une certaine marge, la valeur obtenue pour la première fonction. Les autres critères sont ainsi traités en réitérant ce procédé. D'autres versions multi-objectives de la recherche tabou ont été proposées dans [37, 4, 7, 36]

Dans ce chapitre, le problème d'ordonnancement du flow shop robotique avec deux agents est étudié. L'objectif est de minimiser le makespan global des tâches et le makespan des tâches du premier agent. L'objectif principal de ce chapitre est de proposer une approche pour formaliser la recherche de solutions du compromis à l'aide d'une métaheuristique. Un exemple d'application à un algorithme de recherche tabou est donné pour la résolution du problème du flow shop robotique bi-objectif à deux agents.

3.2 Problème multi-agent

Dans ces problèmes, on considère des partitions disjointes de l'ensemble des travaux, et chaque partition a sa propre fonction objectif à optimiser. Comme la version étendue du problème d'ordonnancement classique, les problèmes d'ordonnancement avec plusieurs agents ont quelques différentes caractéristiques : chaque agent a son propre ensemble de tâches et souhaite minimiser une fonction objectif qui dépend de la date de réalisation de ses tâches ; tous les agents en concurrence sur l'utilisation des ressources de processus communs. Ces caractéristiques conduisent à un nouvel objectif, qui est de trouver un ordonnancement qui minimise une combinaison de fonctions objectifs des agents ou satisfait les exigences de chaque agent pour sa propre fonction objectif.

Classification des problèmes multi-agents : On considère K agents. Chaque agent a son propre ensemble de tâches à traiter N^k, $(k = 1, ..., K)$. La complexité de ces problèmes dépendent de la structure de l'intersection des sous ensembles de tâches N^k $(k = 1, ..., K)$. De ce fait, une classification des problèmes d'ordonnancement multi-agents est donnée dans la littérature en 2014 par Agnetis basée sur la structure de la relation entre les sous-ensembles N^k. On décrit brièvement les différents scénarios de cette classification.

- **Agents compétitifs :** Dans ce cas, les K agents n'ont pas de tâches en commun c'est à dire que toutes les tâches de chaque ensemble N_k appartiennent exclusivement à un seul agent. Ce qui veut dire que $N^h \cap N^k = \emptyset$, $\forall h, k$ tel que $h \neq k$. Dans cette situation, les agents sont en concurrence entre eux dans l'utilisation des ressources du système. La notation de ce scénario dans le champs β est "CO".
- **Agents intérférants :** Dans ce cas, ce scénario concerne les problèmes admettant $K - 1$ agents locaux et un agent global associé à la totalité des travaux. La relation d'ensemble liant ces agents est la suivante : $N^K \subset N^{K-1} \subset .... \subset N^1 = N$. Les agents locaux sont en concurrence comme dans le cas "Compétition", ils sont aussi en conflit avec l'agent global, même dans le cas où leurs critères sont de même nature. La notation de ce scénario dans le champs β est "IN".
- **Agents à ensembles nondisjoints :** C'est le cas général, dans lequel une tâche peut appartenir à plusieurs ensembles de différents agents. Dans le cas de deux agents A et B (K=2), une tâche dans $N^A \cap N^B$ ne peut avoir qu'un seul temps de traitement quel que soit l'agent auquel elle appartient. Pour les autres paramètres comme les dates échues, ça dépends de l'agent auquel elle appartient. La notation de ce scénario dans le champs β est ND.
- **Agents concurrents avec un agent global :** Ce scénario est défini comme suit.

On considère K agents dont les sous-ensembles de tâches associés sont deux à deux disjoints. Chaque agent vise à minimiser une fonction objectif dont la valeur dépend uniquement de ses propres tâches. En plus, il y a un agent global qui vise la minimisation d'une et d'une seule fonction objectif, dont la valeur dépend de toutes les tâches à ordonnancer. La relation entre les ensembles de ces agents est la suivante. $\bigsqcup_{k=1}^{K} N^k \subset N$, et $N^h \cap N^k = \emptyset, \forall h \neq k$. La notation de ce scénario dans le champs β est "CO-GA".

– **Agents multicritères :** C'est le cas d'ordonnancement multicritère classique c'est à dire $N^1 = N^2 = ... = N^1 = N$. La notation de ce scénario dans le champs β est "MU". Et dans le cas de deux critères, on note "BI".

La Figure 3.1 résume ces différents scénarios dans le cas de deux agents A et B.

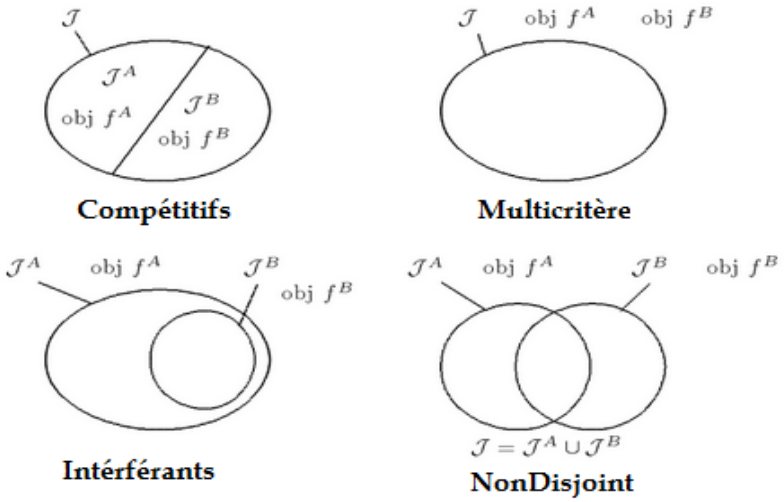

FIGURE 3.1 – Différents scénarios pour un problème d'ordonnancement à deux agents

3.3 Optimisation multi-objective et dominance

Nous présentons ici une brève introduction à l'optimisation multicritère. Un problème d'optimisation se définit comme la recherche d'un optimum, i.e., un minimum ou un maximum d'une fonction donnée. On peut aussi trouver des problèmes d'optimisation pour lesquels les variables de la fonction à optimiser sont contraintes d'évoluer dans une certaine partie de l'espace de recherche. Dans ce cas, on a une forme particulière de ce que l'on appelle un problème d'optimisation sous contraintes [22].

Cependant, lorsque l'on modélise un problème, on cherche souvent à satisfaire plusieurs objectifs. Par exemple, on veut un système performant et on veut aussi que ce

système consomme peu. Dans ce cas, on parle de problème d'optimisation multi-objectif (ou problème d'optimisation multicritère).

Soient $f^1(.), f^1(.), ..., f^k(.)$ des fonctions objectifs à minimiser.

La formulation générale d'un problème multi-objectif est comme suit :

$$\begin{cases} Minimiser \quad F(x) = (f^1(.), f^2(.), ..., f^k(.)) \\ \qquad\quad sous \quad x \in \Omega \end{cases}$$

$x = (X_1, X_2, ..., X_n)$ représente le vecteur des variables de décision. L'espace Ω représente l'ensemble des solutions réalisables et qui peut être défini par un ensemble de contraintes vérifiées par chacune des solutions.

La principale difficulté que l'on rencontre en optimisation mono-objectif vient du fait que modéliser un problème sous la forme d'une équation unique, représentant la réalité du problème, peut être une tâche difficile. Avoir comme but de se ramener à une seule fonction objectif peut aussi biaiser la modélisation. Par contre, l'optimisation multi-objective autorise ces degrés de liberté qui manquaient en optimisation mono-objectif. La recherche ne nous donnera plus une solution unique mais une multitude de solutions. Ces solutions sont appelées solutions de Pareto et l'ensemble des solutions que l'on obtient à la fin de la recherche est la surface de compromis. C'est après avoir trouvé les solutions du problème multi-objectif que d'autres difficultés surviennent : il faut sélectionner une solution dans cet ensemble. La solution qui sera choisie par l'utilisateur va refléter les compromis opérés par le décideur vis-à-vis des différentes fonctions objectifs. Ainsi est introduite la notion de "compromis" ou encore de "non-dominance". Un concept intéressant, qui nous permettra de définir les solutions obtenues. En effet, les solutions que l'on obtient lorsque l'on a résolu le problème sont des solutions de compromis. Elles minimisent un certain nombre d'objectifs tout en dégradant les performances sur d'autres objectifs. Plus de détails sont proposés dans la section ci-après.

3.3.1 Concept de dominance

Historiquement, le premier à évoquer les situations impliquant plusieurs critères en conflit est Vilfredo Pareto en 1896 [67]. Il n'est alors pas question d'optimalité mais d'ophélimité. Ce mot tire de la racine grecque ophelimos, et en terme de multicritère, une solution jouit d'une ophélimité maximale lorsque toute amélioration selon un critère engendre une détérioration selon au moins un autre critère. Pour Vilfredo Pareto, une solution est dite Pareto optimale si elle jouit d'une ophélimité maximale. L'optimalité de Pareto est basée sur la notion de dominance.

En ordonnancement, le concept de dominance d'un sous-ensemble de solutions est

important afin de limiter la complexité algorithmique liée à la recherche d'une solution optimale au sein d'un grand ensemble de solutions.

Lorsque nous avons résolu notre problème d'optimisation multi-objectif, nous avons obtenu une multitude de solutions. Seul un nombre restreint de ces solutions va nous intéresser. Pour qu'une solution soit intéressante, il faut qu'il existe une relation de dominance entre la solution considérée et les autres solutions, dans le sens suivant et en supposant que tous les objectifs sont à minimiser :

Définition 2. *la relation de dominance*

On dit que le vecteur $\overrightarrow{x_1}$ domine le vecteur $\overrightarrow{x_2}$ si :

- *$\overrightarrow{x_1}$ est au moins aussi bon que $\overrightarrow{x_2}$ pour tous les objectifs, et,*
- *$\overrightarrow{x_1}$ est strictement meilleur que $\overrightarrow{x_2}$ pour au moins un objectif.*

Les solutions qui dominent les autres mais ne se dominent pas entre elles sont appelées solutions optimales au sens de Pareto (ou solutions non dominées).

3.3.2 Degré de dominance

Définition 3. *(Dominance faible de Pareto)*

Soient deux solutions x et y, on dit que x domine faiblement (au sens faible) y, si et seulement si, $\forall k \in \{1, ..., K\}$ on a : $f^k(x) \leq f^k(y)$ tel que $\exists\ k \in \{1, ..., K\}\ f^k(x) < f^k(y)$.

Définition 4. *(Dominance stricte de Pareto) Soient deux solutions x et y, on dit que x domine strictement (au sens strict) y, si et seulement si, $\forall k \in \{1, ..., K\}\ f^k(x) < f^k(y)$.*

Donc une solution est appelée Pareto-optimale ou efficace si elle n'est dominée par aucune autre solution. Si E désigne l'ensemble des solutions efficaces, le décideur est logiquement invité à décider parmi les solutions de E dans le cas d'une problématique de choix.

Il faut cependant observer que la détermination de l'ensemble E, à laquelle se limite d'ailleurs un certain nombre de méthodes multicritère, ne résout en général pas le problème.

En effet, d'une part E contient souvent un nombre élevé de solutions (le cas n'est d'ailleurs pas rare où toutes les solutions sont efficaces) et d'autre part les solutions efficaces sont souvent de nature opposée : quand une solution est bonne sur un critère, elle est général moins bonne sur d'autres.

Nous sommes donc confrontés à une situation fort différente du cas mono-critère dans lequel une solution optimale s'impose (ou éventuellement plusieurs solutions optimales indifférentes ou alternatives).

Il importe donc de ne pas confondre les notions d'optimalité (cas mono-critère) et de Pareto-optimalité (cas multicritère). Dans ce dernier cas, décider sur E reste un problème délicat : les solutions efficaces, bien qu'incomparables, ne sont pas indifférentes.

3.3.3 Front de Pareto et points de référence

Toute solution de l'ensemble Pareto peut être considérée comme optimale, dans un certain sens, puisque aucune amélioration ne peut être faite sur un objectif sans dégrader la valeur d'un autre objectif. Ces solutions forment le front Pareto.

Dans le cas d'un problème bi-objectif (deux objectifs à minimiser par exemple), les solutions efficaces peuvent être identifiées visuellement dans l'espace objectif, comme étant celles pour lesquelles le rectangle inférieur gauche formé par la solution et le point $(0\,;\,0)$ est vide de solution réalisable (voir Figure 3.2).

En vue d'avoir certains points de références permettant de discuter de l'intérêt des solutions trouvées, des points particuliers ont été définis dans l'espace objectif. Ces points peuvent représenter des solutions réalisables ou non [25].

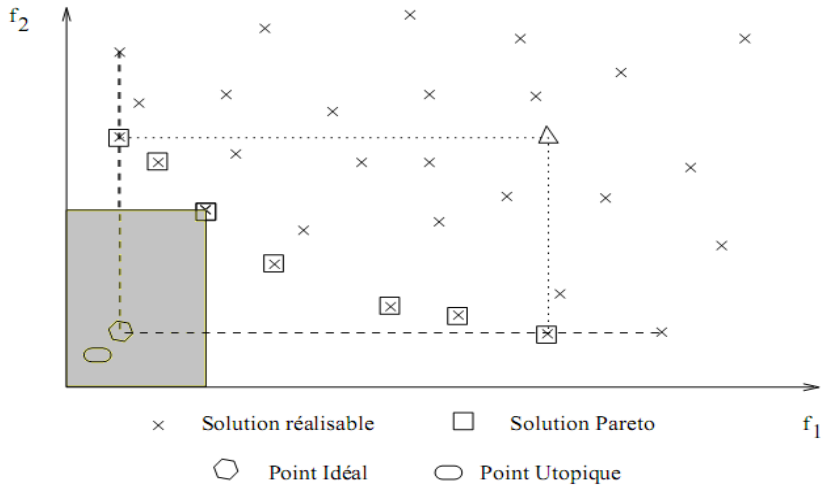

FIGURE 3.2 – Points de référence

Une solution de Pareto est "supportée" si et seulement si elle appartient à l'enveloppe convexe des solution non dominées.

Le point idéal Z^* est le point qui a comme valeur pour chaque objectif la valeur optimale de l'objectif considéré.

$$Z_k^* = \min_{x \in \Omega} f^k(x), \ \forall k = 1, ..., K.$$

Ce point ne correspond pas à une solution réalisable car les objectifs sont contradictoires.

Le point utopique Z^U peut être défini de la façon suivante :

$$Z^U = Z^* - \epsilon U$$

tel que $\epsilon > 0$ et U est le vecteur unitaire ($U = (1, ..., 1) \in R^K$).

3.4 Approches multicritère

L'objectif des approches multicritère est de proposer aux décideurs une ou plusieurs solutions Pareto optimales. Il existe un nombre important de méthodes et nous avons classé celles-ci en cinq groupes :

- Les méthodes scalaires,
- Les méthodes interactives,
- Les méthodes floues,
- Les méthodes exploitant une métaheuristique,
- Les méthodes d'aide à la décision.

Ces approches peuvent être classées en trois catégories comme suit :

- Les approches *à posteriori* : Dans ces méthodes, l'utilisateur choisit une solution de compromis en examinant toutes les solutions extraites par la méthode d'optimisation. Les méthodes de cette famille fournissent, à la fin de l'optimisation, une surface de compromis.
- Les approches *à préférence progressive* : Dans ces méthodes, l'utilisateur affine son choix de compromis au fur et à mesure du déroulement de l'optimisation. On retrouve dans cette famille les méthodes interactives.
- Les approches *à priori* : Dans ces méthodes, l'utilisateur définit le compromis qu'il désire réaliser (il fait part de ses préférences) avant de lancer la méthode d'optimisation. On retrouve dans cette famille la plupart des méthodes par agrégation (où les fonctions objectifs sont fusionnées en une seule par exemple).

La deuxième classification divise les méthodes suivant leur façon de traiter les fonctions objectifs. On distingue les méthodes agrégées, les méthodes basées sur l'équilibre de Pareto et les méthodes non agrégées et non Pareto, que nous détaillons dans la suite [8].

3.4.1 Méthodes agrégées

Les méthodes agrégées transforment un problème multi-objectif en un problème mono-objectif, en regroupant les critères à optimiser dans une unique fonction objectif. Les

méthodes d'agrégation les plus simples utilisent une fonction de mise à l'échelle de chaque critère, afin de pouvoir les additionner (modèle additif) ou bien les multiplier (modèle multiplicatif). La méthode d'agrégation la plus connue et la plus employée est la moyenne pondérée. Cette méthode consiste à additionner tous les objectifs en affectant un cœfficient de poids. Ce cœfficient traduit l'importance relative que le décideur attribue à l'objectif :

$$F(x) = \sum_{i=1}^{K} w_i c^i x$$

où les poids w_i sont tels que $\sum_{i=1}^{K} w_i = 1$.

Quoi que largement utilisée, cette méthode présente toutefois quelques problèmes. D'un côté, le décideur doit déterminer *a priori* les poids de chaque critère. Ceci repose sur la connaissance du problème par le décideur. D'un autre côté, il doit pouvoir exprimer l'interaction entre les différents critères. Ce problème est d'autant plus délicat qu'ils existent plusieurs types d'interaction. De plus, dans le cas où le front de Pareto est non convexe, certaines solutions peuvent ne pas être accessibles en utilisant cette méthode. Il existe également d'autres méthodes agrégées. Nous pouvons citer à titre d'exemple le Goal programming, le min-max et la méthode ϵ-contrainte [8].

3.4.2 Méthodes basées sur l'équilibre de Pareto

Ces méthodes reposent sur le postulat de V. Pareto : "Il existe un équilibre tel que l'on ne peut pas améliorer un critère sans détériorer au moins un des autres critères". La notion de solution optimale unique dans l'optimisation mono-objectif disparaît pour les problèmes d'optimisation multi-objectif au profit de la notion d'ensemble de solutions Pareto optimales, selon le critère de dominance au sens de Pareto. On appellera front de Pareto (ou surface de compromis) du problème, l'ensemble des points de l'espace de recherche tel qu'il n'existe aucun point qui est strictement meilleur que les autres sur tous les critères simultanément. Il s'agit de l'ensemble des meilleurs compromis réalisables entre les objectifs contradictoires. L'objectif de l'optimisation va être d'identifier cet ensemble de compromis optimaux entre les critères. Notez cependant que l'on n'est pas toujours dans une situation aussi régulière et que le front de Pareto peut être concave, discontinu,... etc. Les solutions placées sur le front de Pareto ne peuvent pas être comparées, aucune n'étant systématiquement meilleure que les autres sur tous les objectifs. C'est le décideur qui aura pour rôle de choisir la solution à retenir.

De nombreux algorithmes génétiques ont été développés pour résoudre ce type de problème. Parmi les plus significatifs d'entre eux figurent : l'algorithme Multi Objective

Genetic Algorithm (MOGA), l'algorithme Niched Pareto Genetic Algorithm (NPGA), utilisant une sélection par tournoi, basée principalement sur la dominance de Pareto, l'algorithme Non Dominated Sorting Genetic Algorithm (NSGA) et enfin l'algorithme NSGA-II qui est sans doute l'algorithme le plus populaire [8].

3.4.3 Méthodes non agrégées et non Pareto

En général, les méthodes non agrégées et non Pareto possèdent un processus de recherche qui traite séparément les objectifs. Elle n'utilisent donc aucun des deux principes énoncés précédemment. L'algorithme génétique à évaluation vectorielle (VEGA : Vector Evaluated Genetic Algorithm), en est un exemple. La seule différence avec un algorithme génétique simple est la manière dont s'effectue la sélection. Si nous avons m objectifs et une population initiale de n individus, alors la population est divisée en m sous-populations. La population est ensuite reconstruite et l'on applique les opérateurs génétiques de croisement et de mutation.

3.5 Recherche tabou multi-objectif

La recherche tabou multi-objectif est une méthode qui cherche à simuler une forme de *mémoire*. Pour ce faire, les dernières solutions explorées, ou plus précisément les dernières transformations effectuées pour obtenir ces solutions, sont conservées dans une liste de tabous afin de ne pas reprendre, avant un certain temps, une direction de recherche déjà empruntée. Ceci permet d'éviter un cyclage et de diriger le processus de recherche vers d'autres régions de l'ensemble des solutions. Avant de présenter notre méthode de recherche tabou, on explique les différents voisinages utilisés dans notre méthode. Sachant qu'une solution ici est représentée par une séquence de tâches (on adopte le même codage de la solution que dans le chapitre 4).

3.5.1 Structures de voisinages utilisées

Dans notre méthode tabou, l'ensemble des voisins de la solution courante est choisi aléatoirement pour chaque itération. Trois types de voisinages sont utilisés pour définir un voisin.

- en premier, deux tâches i et j sont permutées dans la séquence ($swap$) ;
- en second, une tâche i est insérée à une différente place k ($move$) ;
- en troisième, à chaque itération un choix est fait entre les deux.

La Figure suivante illustrent les différentes structures.

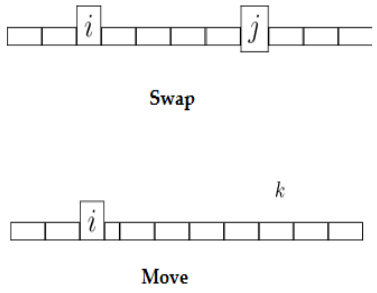

FIGURE 3.3 – Types de voisinages utilisés

3.5.2 Algorithme de la recherche avec tabous multi-objectif

Après avoir fixer les paramètres de la recherche avec tabous à savoir la taille de la liste tabou et la taille de l'ensemble de voisinage, on commence par générer une solution initiale S_0 (au début S_0 contient une seule solution non dominée X_0) et on initialise l'ensemble des points potentiellement efficaces S_{best} qui contient la solution courante S_0. On initialise également l'ensemble courant des solutions $S_{current}$ qui contient bien sûr la solution courante S_0 comme le montre cet organigramme.

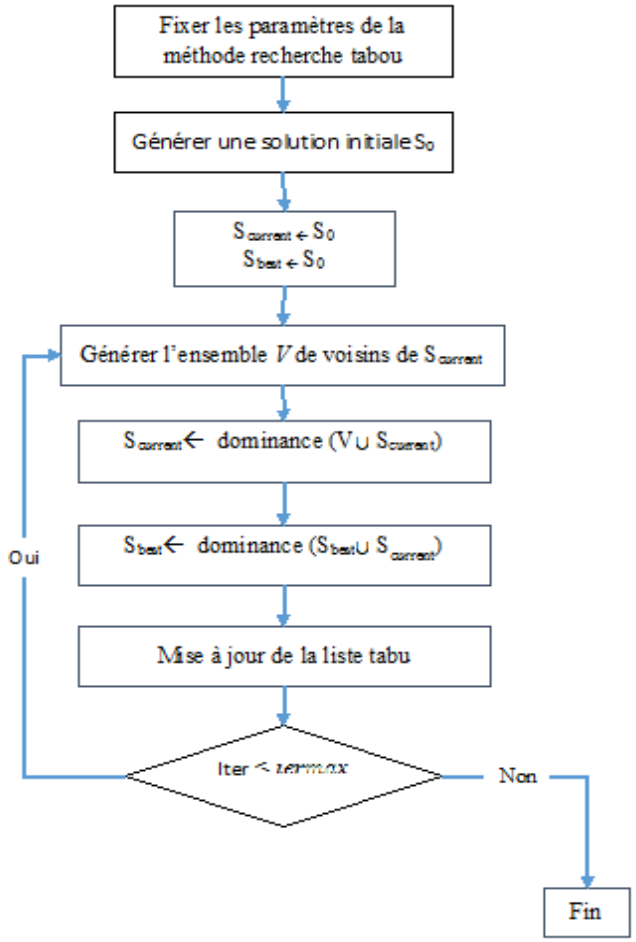

FIGURE 3.4 – Organigramme de la recherche tabou

En seconde étape, on génère un ensemble de points V dans le voisinage de la solution courante. Mais plutôt d'accepter chaque point Y de V s'il est meilleur que la solution courante en un objectif, nous l'acceptons maintenant si ce n'est pas dominé par l'un des points quelconques de l'ensemble courant $S_{current}$. Dans ce cas, on ajoute tout point Y qui n'est pas dominé à l'ensemble $S_{current}$ et on jette tous les points dominés de $S_{current}$. En d'autres termes, on cherche les solutions non dominées dans l'ensemble $V \bigcup S_{current}$ et on mis à jour l'ensemble des points non dominés S_{best}. On mis à jour également la liste Tabou en sauvegardant les mouvements utilisés pour avoir $S_{current}$ et on répète le procédé depuis le début en générant un autre ensemble V de $S_{current}$ et on refait les mêmes étapes jusqu'à ce que le critère d'arrêt soit vérifié (nombre maximum d'itérations est atteint). L'algorithme suivant donne les étapes de la recherche tabou :

Algorithm 8 Algorithme de Recherche Tabou

1: initialiser : $Size \leftarrow \frac{N}{2}$, $\mid V \mid = 500$, $Iter_{max} = 1000$
2: Construction d'une solution initiale S_0
3: Initialisation à vide de la liste tabou
4: $iter = 1$ *compteur d'itération*
5: $S_{current} = S_{best} = S_0$ S_{best} *est l'ensemble contenant les solutions non dominées*
6: **Tant Que** $(iter \leq Iter_{max})$
7: *Neignbors search* Générer un ensemble V de voisins de $S_{current}$. V *contient les voisins de tout les points dans* $S_{current}$
8: $S_{current} = dominance(V \bigcup S_{current})$ *Mise à jour de l'ensemble courant*
9: $S_{best} = dominance(S_{current} \bigcup S_{best})$ *Mise à jour de l'ensemble des solutions non dominées*
10: *update memory* Mise à jour de la liste tabou
11: $iter \leftarrow iter + 1$.
12: **Fin Tant Que**

3.6 Résolution par la recherche avec tabous multi-objectif

Nous avons appliqué la méthode de recherche tabou multi-objectif pour résoudre le problème $TF2(D2,1)|\upsilon = 1, c = 1|C^{gl}_{max}, C^{1}_{max}$.

Le problème à résoudre consiste à trouver le meilleur compromis, entre le critère global C^{gl}_{max} appliqué aux travaux de N et le critère C^{1}_{max} appliqué aux travaux de N_1 (celui de l'agent 1). Suivant la notation à trois champs des problèmes d'ordonnancement, le problème est noté $TF2(D2,1)|\upsilon = 1, c = 1|C^{gl}_{max}, C^{1}_{max}$. Ce problème est NP-difficile par déduction de la complexité du problème mono-critère $TF2(D2,1)|\upsilon = 1, c = 1|C_{max}$.

3.7 Evaluation numérique

Dans cette section, nous présentons les résultats de la méthode de Recherche Tabou multi-objectif appliquée au problème $TF2(D2,1)|\upsilon = 1, c = 1|C^{gl}_{max}, C^{1}_{max}$ en minimisant le makespan global C^{gl}_{max} et le makespan du premier agent C^{1}_{max}. Des essais numériques ont été réalisés afin d'évaluer la performance de l'approche Recherche Tabou Multi-objectif. La performance de la métaheuristique utilisée a été vérifiée à l'aide des 50 problèmes de

taille $n = 10$ à $n = 200$ tâches.

Pour les instances de différente taille, les temps de traitement p_i^s $(\forall i \in N, s \in \{1, 2\})$ sont générés à partir d'une distribution uniforme discrète sur l'intervalle $[1, 100]$ et le temps de transport est généré selon la loi uniforme dans l'intervalle $[1, 30]$. Le nombre choisi de tâches pour chaque agent est $n_1 = n_2 = 0.5 \cdot n$. Pour chaque combinaison de paramètres $(n,\ t)$, nous avons généré au hasard 10 instances. Les tableaux 3.1 et 3.2 représentent les résultats obtenus pour des instances de petite et grande taille.

problèmes	n	Solutions obtenues
Problème 1	10	(578, 303) (639, 298)
Problème 2	10	(517, 350) (533, 345) (565, 312)
Problème 3	10	(584, 338) (586, 290)
Problème 4	10	(538, 538) (542, 314) (553,269)
Problème 5	10	(479, 334) (484, 334) (495, 329)
Problème 6	10	(564, 365) (575, 296)
Problème 7	10	(472,472) (483, 351) (493, 351) (501, 304) (526, 292) (529, 280) (558, 277) (579, 266)
Problème 8	10	(466,403) (474, 392) (483, 390) (488, 383) (495,381)
Problème 9	10	(533,380) (552,374)
Problème 10	10	(580, 580) (588, 280)
Problème 11	20	(1014,643) (1015, 633)
Problème 12	20	(1086,578)
Problème 13	20	(1065,643)
Problème 14	20	(1203,644)
Problème 15	20	(1180,1180) (1183, 640) (1201, 612)
Problème 16	20	(917, 560) (919, 558) (920, 557) (922, 555) (934,548) (940,543) (942, 541) (943, 540) (945, 538) (946,537) (948,535) (966,31)
Problème 17	20	(1096, 608) (1105,570)
Problème 18	20	(758, 707) (760, 705) (762, 703) (764, 701) (768, 697) (769, 678) (771, 676) (772, 674) (774, 672) (776, 670) (779, 668) (781, 665) (782, 664) (783, 663) (785, 661) (791, 649)
Problème 19	20	(1179, 872) (1180,870) (1182,833) (1183, 828) (1184, 806) (1187, 768) (1187, 775) (1209, 721) (1212, 708) (1241, 648) (1285, 598)
Problème 20	20	(911,532) (919,498)

TABLE 3.1 – Expérimentations numériques sur des instances de petite taille.

problèmes	n	Solutions obtenues
Problème 21	50	(2425, 1417) (2426, 1415) (2428, 1413) (2430, 1411) (2432, 1409)
Problème 22	50	(2653, 1497)
Problème 23	50	(2923, 1459) (2924, 1398)
Problème 24	50	(2569, 1432) (2570, 1431) (2571, 1430) (2572, 1429) (2573, 1428) (2574, 1427) (2575, 1426) (2576, 1425) (2577, 1424) (2578, 1423) (2579, 1422) (2580, 1421) (2581, 1420) (2582, 1419) (25
Problème 25	50	(2373, 1315) (2375 , 1313) (2376,1312) (2377 , 1311) (2378 , 1310) (2379 , 1309) (2380 , 1308) (2381 , 1307) (2382 , 1306) (2383 , 1305) (2384 , 1304) (2385 , 1303) (2386 , 1302) (2387 ,
Problème 26	50	(2374, 1474) (2375, 1473) (2376, 1472) (2377, 1471) (2378 , 1470) (2379 , 1469) (2380, 1468) (2381, 1467) (2382, 1466) (2383, 1465) (2384, 1464) (2385, 1463) (2386, 1462) (2387, 1461) (
Problème 27	50	(2776, 1511)
Problème 28	50	(2410, 1265) (2411, 1264) (2412, 1263) (2413, 1262) (2414, 1261) (2415, 1260) (2416, 1259) (2417, 1258) (2418, 1257) (2419, 1256) (2420, 1255) (2421, 1254) (2422, 1253) (2423, 1252) (24
Problème 29	50	(2597, 1395) (2603, 1364)
Problème 30	50	(2254, 1396) (2255, 1395) (2256, 1394) (2257, 1393)
Problème 31	100	(5146, 2844) (5147, 2843) (5148, 2842) (5149, 2841) (5150, 2840) (5151, 2839) (5152, 2838) (5153, 2837) (5154, 2836) (5155, 2835) (5156, 2834) (5157, 2833) (5158, 2832) (5159, 2831) (51
Problème 32	100	(5450, 2660) (5451, 2641) (5452, 2598)
Problème 33	100	(4614, 2924) (4615, 2923) (4616, 2922) (4617, 2921) (4618, 2920) (4619, 2919) (4620, 2918) (4621, 2917) (4622, 2916) (4623, 2915) (4624, 2914) (4625, 2913) (4626, 2912) (4627, 2911) (46
Problème 34	100	(5577, 5577) (5578, 2777)
Problème 35	100	(4978, 2530) (4983, 2485)
Problème 36	100	(4464, 2623)
Problème 37	100	(4659, 2504) (4660, 2503) (4661, 2502) (4662, 2501) (4663, 2500) (4664, 2499) (4665, 2498) (4666, 2497) (4667, 2496) (4668, 2495) (4669, 2494) (4670, 2493) (4671, 2492) (4672, 2491) (46
Problème 38	100	(5155, 2820) (5156, 2819) (5157, 2818) (5158, 2817) (5159, 2816) (5160, 2815) (5161, 2814) (5162, 2813) (5163, 2812) (5164, 2811) (5165, 2810) (5166, 2809) (5167, 2808) (5168, 2807) (51
Problème 39	100	(5574, 5574) (5576, 2774)
Problème 40	100	(5087, 2748) (5091, 2730) (5093, 2678)
Problème 41	200	(11574, 11574) (11575, 5781)
Problème 42	200	(10162, 5349) (10163, 5350)
Problème 43	200	(9741, 5345) (9742, 5344) (9743, 5343)
Problème 44	200	(9767, 5099) (9768, 5098) (9769, 5097) (9770, 5096) (9771, 5095) (9772, 5094) (9773, 5093) (9774, 5092) (9775, 5091) (9776, 5090) (9777, 5089) (9778, 5088) (9779, 5087) (9780, 5086) (97
Problème 45	200	(9699, 5017) (9700, 5016) (9701, 5015) (9702, 5014) (9703, 5013) (9704, 4996) (9707, 4992) (9708, 4991) (9709, 4990) (9710, 4989) (9711, 4988) (9712, 4987) (9713, 4986) (9714, 4985) (97
Problème 46	200	(10166, 5307)
Problème 47	200	(11973, 5973)
Problème 48	200	(10775, 5377)
Problème 49	200	(10187,5249)
Problème 50	200	(9830, 5047) (9831, 5046) (9832, 5045) (9833, 5044) (9834, 5043) (9835, 5042) (9836, 5041)

TABLE 3.2 – Expérimentations numériques sur des instances de grande taille.

Nous calculons dans les deux tableaux 3.3 et 3.4 les paramètres suivants :

– $\mid S \mid$: représente les cardinalités des fronts de pareto approchés.
– $\mid C^{gl}_{max} \mid$: représente les valeurs moyennes du makespan global C^{gl}_{max}.
– $\mid C^{1}_{max} \mid$: représente les valeurs moyennes du makespan du premier agent C^{1}_{max}.
– CPU : représente les temps de calcul pour chaque problème.

problèmes	n	$\mid S \mid$	CPU	$\mid C^{gl}_{max} \mid$	$\mid C^{1}_{max} \mid$
Problème 1	10	2	3.963	608,5	300,5
Problème 2	10	3	5,959	538,333	335,666
Problème 3	10	2	3,884	585	314
Problème 4	10	3	6,116	544,333	373,666
Problème 5	10	3	5,662	486	332,333
Problème 6	10	2	3,930	569,5	330,5
Problème 7	10	8	16,427	517,625	324,125
Problème 8	10	5	9,765	481,2	389,8
Problème 9	10	2	4,071	542,5	377
Problème 10	10	2	3,961	584	430
Moyenne		2.4	6,374	545,699	350,759
Problème 11	20	2	9,859	1014,5	638
Problème 12	20	1	4,618	1086	578
Problème 13	20	1	4,585	1065	643
Problème 14	20	1	4,586	1203	644
Problème 15	20	3	14,164	1188	810,666
Problème 16	20	12	57,750	936,833	545,25
Problème 17	20	2	9,390	1100,5	589
Problème 18	20	16	74,694	773,437	678,937
Problème 19	20	11	52,681	1202,636	766,090
Problème 20	20	2	9,296	915	515
Moyenne		5.1	24,162	1048,4906	640,7943

TABLE 3.3 – Résultats des instances de petite taille.

problèmes	n	$\mid S \mid$	CPU	$\mid C^{gl}_{max} \mid$	$\mid C^{1}_{max} \mid$
Problème 21	50	5	102,819	2428,2	1413
Problème 22	50	1	21,371	2653	1497
Problème 23	50	2	40,794	2923,5	1428,5
Problème 24	50	30	611,289	2585,2	1415,8
Problème 25	50	36	759,411	2394,583	1277,944
Problème 26	50	24	451,654	2385,5	1462,5
Problème 27	50	1	20,468	2776	1511
Problème 28	50	21	429,361	2420,476	1254,904
Problème 29	50	2	40,373	2600	1379,5
Problème 30	50	4	83,554	2255,5	1394,5
Moyenne		12.6	256,110	2542,196	1403,465
Problème 31	100	22	1004,382	5156,545	2833,454
Problème 32	100	3	127,904	5451	2633
Problème 33	100	38	1753,951	4632,5	2905,5
Problème 34	100	2	93,194	5577,5	4177
Problème 35	100	2	98,935	4980,5	2507,5
Problème 36	100	1	53,789	4464	2623
Problème 37	100	22	1024,911	4669,5	2493,5
Problème 38	100	20	931,403	5164,5	2810,5
Problème 39	100	2	87,299	5575	4174
Problème 40	100	3	127,904	5090,333	2718,666
Moyenne		11.5	530,367	5076,138	2987,612
Problème 41	200	2	127,874	11574,5	8677,5
Problème 42	200	2	122,898	10162,5	5349,5
Problème 43	200	3	209,432	9742	5344
Problème 44	200	31	1737,444	9782	5084
Problème 45	200	44	2729,175	9723,272	4974,840
Problème 46	200	1	64,365	10166	5307
Problème 47	200	1	114,379	11973	5973
Problème 48	200	1	118,374	10775	5377
Problème 49	200	1	63,540	10187	5249
Problème 50	200	7	396,710	9833	5044
Moyenne		9.3	568,419	10391,827	5637,984

TABLE 3.4 – Résultats des instances de grande taille.

D'après le Tableau 3.3 et le Tableau 3.4, on remarque que lorsque le nombre de travaux $n = 10$, le nombre de solutions ne dépasse pas en moyenne sur toutes les instances 3 solutions. La taille moyenne de front de Pareto approché augmente en fonction du nombre de tâches. En d'autres termes, En général, nous obtenons à chaque fois plus de solutions approchées lorsque le nombre de tâches augmente. La même remarque pour le temps de calcul qui augmente en fonction du nombre de tâches.

Nous avons généré un nuage de points pour le front de Pareto. Une illustration graphique d'exemples de front de Pareto pour deux instances de différente taille est donné dans les deux figures suivantes :

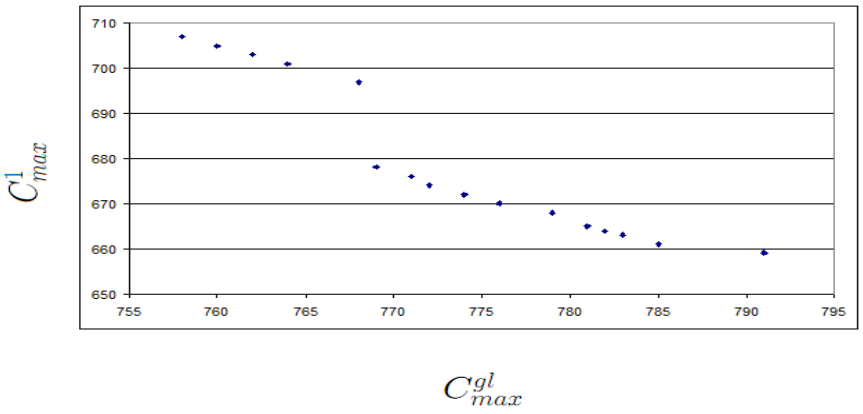

FIGURE 3.5 – Exemple de front de Pareto généré aléatoirement pour une instance avec $n = 20$ et $n_1 = 10$

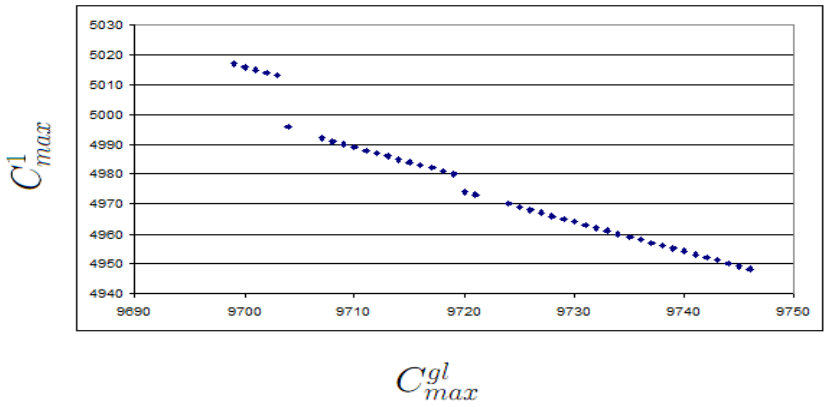

FIGURE 3.6 – Exemple de front de Pareto généré aléatoirement pour une instance avec $n = 200$ et $n_1 = 100$

D'après ces deux figures, on peut observer, les points sont répartis de manière presque linéaire sur le plan. On voit que les solutions ont des valeurs du critère global de C^{gl}_{max} importantes par rapport au valeurs du critère du premier agent C^1_{max}. Ceci est évident vu la globalité du critère C^{gl}_{max} (makespan de tous les travaux). Par contre la répartition des solutions montre que les valeurs du critère global de C^{gl}_{max} sont proportionnelles en fonction des valeurs du critère du premier agent C^1_{max}.

3.8 Conclusion

Dans ce chapitre, tout d'abord, nous avons rappelé les concepts de base des problèmes multicritère et un survol des méthodes utilisées pour résoudre les problèmes d'optimisation multicritère. Après un état de l'art des problèmes multi-agent et leur classification, nous avons présenté notre méthode de recherche tabou multi-objectif pour la résolution d'un problème d'ordonnancement de travaux avec deux agents dont l'un est global. Les deux critères considérés sont la minimisation du makespan de tous les travaux et la minimisation du makespan du premier agent. Nous avons expliqué le principe et les étapes de la méthode ainsi que les différents types de voisinage utilisés. Dans la partie expérimentation, des tests ont été effectués sur plusieurs instances générées aléatoirement. Ce travail n'entre pas dans le même cadre mono-objectif que les travaux des chapitres 2, 3 et 4 précédents, et peut être considéré comme une première contribution dans le contexte des problèmes d'ordonnancement multi-agent dans un environnement flow shop dans lesquels les temps

du transport sont pris en compte.

Conclusion générale

Cette thèse avait pour objet la résolution du problème d'ordonnancement de flow shop robotique avec des machines dédiées qui est un problème d'optimisation. La première partie traite le cas mono-objectif du problème. Cette partie concerne les quatre premiers chapitres. En effet, dans le premier chapitre, Nous avons présenté, dans un premier temps, quelques concepts de base du problèmes d'ordonnancement et aussi un état de l'art sur les méthodes de la littérature permettant de résoudre les problèmes d'optimisation.

Une première contribution est dans le deuxième chapitre qui concerne l'étude du problème de flow shop robotique à deux étages avec deux machines dédiées au premier étage et une machine commune au deuxième étage. Nous avons traité le cas à capacité unitaire du robot et le cas général avec capacité limitée à plusieurs tâches. Nous proposons deux modèles mathématiques en variables mixtes. Ces deux modèles sont testés sous le solveur Cplex sur plusieurs schémas de génération. Une comparaison théorique et expérimentale est effectuée entre ces deux modèles. Des bornes inférieures ont été établies pour la fonction objectif. Ces bornes sont utilisées comme référence dans la partie expérimentation.

Résoudre des problèmes d'optimisation est un point clé dans l'amélioration constante de la productivité des entreprises. Quand les méthodes traditionnelles échouent, il devient alors naturel de se tourner vers des techniques de résolution approchée. La conception des heuristiques pour la résolution du problème général. Les différents tests effectués révèlent l'efficacité et la performance des heuristiques utilisées en terme de la qualité de la solution et le temps de calcul d'exécution.

Une autre étude s'impose dans le troisième chapitre concerne la complexité du problème. Nous avons démontré les cas NP-difficiles et nous avons proposé des algorithmes de résolution en temps polynomial pour quelques cas particulies. On cite essentiellement les cas avec capacité de robot à deux tâches pour lequel un algorithme dynamique est proposé avec un exemple d'application. Le problème à capacité unitaire du robot est analysé dans

le cas des temps de traitement identiques sur le premier étage puis sur le deuxième étage. Enfin, un cas général avec une capacité quelconque du robot et des temps de traitement identiques sur le deuxième étage a été démontré NP-difficile.

Les métaheuristiques jouent, aujourd'hui, un rôle primordial dans la résolution des problèmes d'optimisation. Ces techniques sont devenues, en quelques années, des outils incontournables et performants. Parmi ces méthodes, la recherche à voisinages variables qui est une méthode récente basée sur la performance des méthodes de descente. La méthode propose simplement d'utiliser plusieurs voisinages successifs quand on se trouve bloqué dans un minimum local. La RVV est une méthode intéresse de plus en plus de chercheurs et est promise à un bel avenir. Le nombre de papiers utilisant cette méthode est en très forte augmentation dans les congrès.

Le point crucial dans une RVV (VNS), c'est bien évidement la constitution des voisinages de plus en plus grands et inclus les uns dans les autres. Une bonne structure de voisinage conduit généralement à de bons résultats ou au moins à une recherche efficace. Nous avons donc appliquée cette méthode à notre problème en proposant plusieurs variantes de RVV avec différentes structures de voisinage. Nous avons également comparé cette méthode avec des algorithmes génétiques. Les résultats de la partie expérimentation montrent que la RVV et les AG sont très adaptés à notre problème en fournissant des meilleures solutions en un temps de calcul très raisonnable.

A propos des méthodes de résolution pour l'optimisation combinatoire multi-objectif, différentes perspectives sont envisageables. Dans la deuxième partie de ce these, on propose une méthode de recherche tabou multi-objectif et on l'applique à notre problème avec capacité unitaire du robot tout en minimisant deux objectifs dont l'un est global.

Il existe plusieurs extensions possibles à ce travail de recherche. Analyser différents modèles liés aux caractéristiques des espaces de stockage et le nombre de robots. Il serait également intéressant de développer d'autres méthodes exactes comme le branch and bound. Etudier d'autres fonctions objectifs. Également, une autre direction de recherche est de prendre en compte l'aspect stochastique du problème présenté (par exemple le temps de traitement, temps de transport) et d'introduire des contraintes de disponibilité.

Bibliographie

[1] Allesandro Agnetis, Pitu B Mirchandani, Dario Pacciarelli, and Andrea Pacifici. Scheduling problems with two competing agents. *Operations Research*, 52(2) :229–242, 2004.

[2] Fardin Ahmadizar and Parmis Shahmaleki. Group shop scheduling with sequence-dependent setup and transportation times. *Applied Mathematical Modelling*, 2014.

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

[4] Vincent Barichard and Jin-Kao Hao. Genetic tabu search for the multi-objective knapsack problem. *Tsinghua Science and Technology*, 8(1) :8–13, 2003.

[5] J Behnamian, SMT Fatemi Ghomi, F Jolai, and O Amirtaheri. Realistic two-stage flowshop batch scheduling problems with transportation capacity and times. *Applied Mathematical Modelling*, 36(2) :723–735, 2012.

[6] R Bellman. Dynamic programming. *Princeton University Press*, 1957.

[7] F Ben Abdelaziz and S Krichen. A tabu search heuristic for multiple objective knapsack problems. Technical report, Ructor Research Report, RR 28-97, 1997.

[8] Ilhem Boussaid. *Improvement of metaheuristics for continuous optimization.* PhD thesis, Université Paris-Est, 2013.

[9] P. Brucker and S. Knust. Complexity results for flow-shop and open-shop scheduling problems with transportation delays. *Annals of Operations Research 129*, pages 81–106, 2004.

[10] Grissele Centeno and Robert L Armacost. Minimizing makespan on parallel machines with release time and machine eligibility restrictions. *International Journal of Production Research*, 42(6) :1243–1256, 2004.

[11] J. Cheng and H. Kise. Optimal scheduling for automated two-machine manufactxring systems with intermediate operations. *Transactions of the Institute of Systems, Control and Information Engineers of Japan*, 8 :65–71, 1995.

[12] TC Edwin Cheng, Shuenn-Ren Cheng, Wen-Hung Wu, Peng-Hsiang Hsu, and Chin-Chia Wu. A two-agent single-machine scheduling problem with truncated sum-of-processing-times-based learning considerations. *Computers & Industrial Engineering*, 60(4) :534–541, 2011.

[13] TC Edwin Cheng, Yu-Hsiang Chung, Shan-Ci Liao, and Wen-Chiung Lee. Two-agent singe-machine scheduling with release times to minimize the total weighted completion time. *Computers & Operations Research*, 40(1) :353–361, 2013.

[14] TC Edwin Cheng, CT Ng, and JJ Yuan. Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs. *Theoretical Computer Science*, 362(1) :273–281, 2006.

[15] TC Edwin Cheng, CT Ng, and JJ Yuan. Multi-agent scheduling on a single machine with max-form criteria. *European Journal of Operational Research*, 188(2) :603–609, 2008.

[16] Nacira Chikhi, Moncef Abbas, Abdelghani Bekrar, Rachid Benmansour, and Said Hanafi. On the complexity of robotic flow shop with transportation constraints. In *ROADEF - 15ème congrès annuel de la Société française de recherche opérationnelle et d'aide à la décision*, Bordeaux, France, 2014.

[17] Nacira Chikhi, Moncef Abbas, Rachid Benmansour, Abdelghani Bekrar, and Saïd Hanafi. A two-stage flow shop scheduling problem with transportation considerations. *A Quartely journal of operations research "4OR"*, 13(4) :381–402, 2015.

[18] Nacira Chikhi, Moncef Abbas, Rachid Benmansour, and Saïd Hanafi. New complexity results on scheduling problem in a robotic cell. *DOI : http ://dx.doi.org/10.1051/ro/2016053 RAIRO*, 2016.

[19] Nacira Chikhi, Moncef Abbas, Saïd Hanafi, and Nenad Mladenović. A general variable neighborhood search for a scheduling problem in a robotic cell. In *11th edition of the Metaheuristics International Conference (MIC)*, 2015.

[20] Nacira Chikhi, Rachid Benmansour, Abdelghani Bekrar, Saïd Hanafi, and Moncef Abbas. A case study of a two-stage flow shop with dedicated machines and a single robot. In *CoDIT*. IEEE, 2014.

[21] H. Papadimitriou Christos and Mihalis Yannakakis. On bounded rationality and computational complexity. Technical report, Indiana University, 1994.

[22] Yann Collette and Patrick Siarry. *Optimisation multiobjectif : Algorithmes*. Editions Eyrolles, 2011.

[23] Wagner Emanoel Costa, Marco César Goldbarg, and Elizabeth G Goldbarg. Hybridizing vns and path-relinking on a particle swarm framework to minimize total flowtime. *Expert Systems with Applications*, 39(18) :13118–13126, 2012.

[24] Wagner Emanoel Costa, Marco César Goldbarg, and Elizabeth G Goldbarg. New vns heuristic for total flowtime flowshop scheduling problem. *Expert Systems with Applications*, 39(9) :8149–8161, 2012.

[25] C. Dhaenens. *Multi-objective combinatorial Optimization : contribution of cooperative methods and application to knowledge discovery*. PhD thesis, Université des Sciences et Technologie de Lille - Lille I, 2005.

[26] Atabak Elmi and Seyda Topaloglu. A scheduling problem in blocking hybrid flow shop robotic cells with multiple robots. *Computers & Operations Research*, 40(10) :2543–2555, 2013.

[27] BQ Fan, TC Edwin Cheng, SS Li, and Q Feng. Bounded parallel-batching scheduling with two competing agents. *Journal of Scheduling*, 16(3) :261–271, 2013.

[28] Clarisse Flipo-Dhaenens. *Optimisation of a production and distribution network*. PhD thesis, Institut National Polytechnique de Grenoble - INPG, 1998.

[29] M. R. Garey, D. S. Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2) :117–129, 1976.

[30] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5) :533–549, 1986.

[31] Fred Glover and Saıd Hanafi. Tabu search and finite convergence. *Discrete Applied Mathematics*, 119(1) :3–36, 2002.

[32] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of discrete Mathematics*, 5 :287–326, 1979.

[33] A. Gunasekaran, Martikainen, and P. Yli-Olli. Flexible manufacturing systems : An investigation for research and applications. *European Journal of Operational Research*, 66(1) :1–26, 1993.

[34] Jatinder ND Gupta. Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, pages 359–364, 1988.

[35] Mohamed Karim Hajji, Hatem Hadda, and Najoua Dridi. Une heuristique pour le flow shop hybride à deux étages avec machines dédiées. *17ème congrès ROADEF*, 2016.

[36] Michael Pilegaard Hansen. Tabu search for multiobjective optimization : Mots. In *Proceedings of the 13th International Conference on Multiple Criteria Decision Making*, pages 574–586. Citeseer, 1997.

[37] Alain Hertz. Tabu search for large scale timetabling problems. *European journal of operational research*, 54(1) :39–47, 1991.

[38] Alain Hertz, Brigitte Jaumard, CC Ribeiro, and WP Formosinho Filho. A multi-criteria tabu search approach to cell formation problems in group technology with multiple objectives. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, 28(3) :303–328, 1994.

[39] Johannes Adzer Hoogeveen, Jan Karel Lenstra, and Bart Veltman. Preemptive scheduling in a two-stage multiprocessor flow shop is np-hard. *European Journal of Operational Research*, 89(1) :172–175, 1996.

[40] Johann Hurink and Sigrid Knust. Makespan minimization for flow-shop problems with transportation times and a single robot. *Discrete Applied Mathematics*, 112(1) :199–216, 2001.

[41] Hark-Chin Hwang, Soo Y Chang, and Kangbok Lee. Parallel machine scheduling under a grade of service provision. *Computers & Operations Research*, 31(12) :2055–2061, 2004.

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

[43] Jihène Kaabi-Harrath. *Contribution to maintenance activity scheduling in production system.* PhD thesis, Université de Franche-Comté, 2004.

[44] Mohammed Kharbeche. *Exact and heuristic methods for variants of the permutation flow shop problems.* PhD thesis, University of Tunis, 2011.

[45] H. Kise, K. Kohno, T. Shioyama, and T. Kushiyama. Optimal scheduling for an automated two-machine manufacturing system. *Journal of Japanese Society of Mechanical Engineers*, 57 :121–127, 1992.

[46] H. Kise, T. Shioyama, and T. Ibaraki. Automated two-machine flowshop scheduling : A solvable case. *HE Transactions*, 23(1) :10, 1991.

[47] Hiroshi Kise. On an automated two-machine flowshop scheduling problem with infinite buffer. *Journal of the Operations Research Society of Japan*, 34(3) :354–361, 1991.

[48] Oumar Kone. *Nouvelles approches pour la résolution du problème d'ordonnancement de projet à moyens limités*. PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2009.

[49] Mohand Larabi. *Job-shop with transport : its modelling and optimisation*. PhD thesis, Université Blaise Pascal - Clermont-Ferrand II, 2010.

[50] C Y Lee and Z L Chen. Machine scheduling with transportation considerations. *J. sched*, 4(3) :24, 2001.

[51] Wen-Chiung Lee, Shiuan-Kang Chen, Cheng-Wei Chen, and Chin-Chia Wu. A two-machine flowshop problem with two agents. *Computers & Operations Research*, 38(1) :98–104, 2011.

[52] Wen-Chiung Lee, Shiuan-kang Chen, and Chin-Chia Wu. Branch-and-bound and simulated annealing algorithms for a two-agent scheduling problem. *Expert Systems with Applications*, 37(9) :6594–6601, 2010.

[53] Wen-Chiung Lee, Yu-Hsiang Chung, and Mei-Chia Hu. Genetic algorithms for a two-agent single-machine problem with release time. *Applied Soft Computing*, 12(11) :3580–3589, 2012.

[54] Deming Lei and Tao Wang. An effective neighborhood search algorithm for scheduling a flow shop of batch processing machines. *Computers & Industrial Engineering*, 61(3) :739–743, 2011.

[55] E. Levner, K. Kogan, and I. Levin. Scheduling a two-machine robotic cell : A solvable case. *Mathematics of Industrial Systems*, 1, 1995.

[56] Eugene Levner, Konstantin Kogan, and Ilya Levin. Scheduling a two-machine robotic cell : A solvable case. *Annals of Operations Research*, 57(1) :217–232, 1995.

[57] Shisheng Li and Jinjiang Yuan. Unbounded parallel-batching scheduling with two competitive agents. *Journal of Scheduling*, 15(5) :629–640, 2012.

[58] Bertrand MT Lin. The strong np-hardness of two-stage flowshop scheduling with a common second-stage machine. *Computers & operations research*, 26(7) :695–698, 1999.

[59] Yen-Cheng Liu, Kuei-Tang Fang, and Bertrand Lin. A branch-and-bound algorithm for makespan minimization in differentiation flow shops. *Engineering Optimization*, 45(12) :1397–1408, 2013.

[60] Taïcir Loukil, Jacques Teghem, and Daniel Tuyttens. Solving multi-objective production scheduling problems using metaheuristics. *European journal of operational research*, 161(1) :42–61, 2005.

[61] Parkash Lal Maggu, Ghanshiam Das, and Ravinder Kumar. On equivalent-job for job-block in 2× n sequencing problem with transportation-times. *Journal of the Operations Research Society of Japan*, 24(2) :136–146, 1981.

[62] R M'Hallah. Minimizing total earliness and tardiness on a permutation flow shop using vns and mip. *Computers & Industrial Engineering*, 75 :142–156, 2014.

[63] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11) :1097–1100, 1997.

[64] Baruch Mor and Gur 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.

[65] Ceyda O$\widehat{g}$uz, Bertrand MT Lin, and TC Edwin Cheng. Two-stage flowshop scheduling with a common second-stage machine. *Computers & operations research*, 24(12) :1169–1174, 1997.

[66] SS. Panwalkar. Scheduling of a two-machine flowshop with travel time between machines. *Journal of the Operational Research Society*, pages 609–613, 1991.

[67] V. Pareto. Cours d'economie politique. *Rouge and Cie, Lausanne, Switzerland*, 1, 1896.

[68] M. Pinedo. *Scheduling : theory, algorithms and systems*. PhD thesis, Prentice Hall Series, New Jersey, 1995.

[69] Fouad Riane, Abdelhakim Artiba, and Salah E Elmaghraby. Sequencing a hybrid two-stage flowshop with dedicated machines. *International journal of production research*, 40(17) :4353–4380, 2002.

[70] Marc Sevaux. *Métaheuristiques : Stratéegies pour l'optimisation de la production de biens et de services*. PhD thesis, Université de Valenciennes, 2004.

[71] H. I. Stern and G. Vitner. Scheduling parts in a combined production-transportation work cell. *Journal of the Operational Research Sociaty*, 41 :625–632, 1990.

[72] T. Stützle. Applying iterated local search to the permutation flow shop problem. *FG Intellektik, TU Darmstadt, Darmstadt, Germany*, 1998.

[73] Lixin Tang and Peng Liu. Two-machine flowshop scheduling problems involving a batching machine with transportation or deterioration consideration. *Applied Mathematical Modelling*, 33(2) :1187–1199, 2009.

[74] Ekunda Lukata Ulungu and Jacques Teghem. Multi-objective combinatorial optimization problems : A survey. *Journal of Multi-Criteria Decision Analysis*, 3(2) :83–104, 1994.

[75] Shijin Wang and Ming Liu. A heuristic method for two-stage hybrid flow shop with dedicated machines. *Computers & Operations Research*, 40(1) :438–450, 2013.

[76] Chin-Chia Wu, Shih-Ke Huang, and Wen-Chiung Lee. Two-agent scheduling with learning consideration. *Computers & Industrial Engineering*, 61(4) :1324–1335, 2011.

[77] Wen-Hsiang Wu, Jianyou Xu, Wen-Hung Wu, Yunqiang Yin, I-Fan Cheng, and Chin-Chia Wu. A tabu method for a two-agent single-machine scheduling with deterioration jobs. *Computers & Operations Research*, 40(8) :2116–2127, 2013.

[78] Jaehwan Yang. A two-stage hybrid flow shop with dedicated machines at the first stage. *Computers & operations research*, 40(12) :2836–2843, 2013.

[79] Jaehwan Yang. Minimizing total completion time in a two-stage hybrid flow shop with dedicated machines at the first stage. *Computers & Operations Research*, 58(0) :1–8, 2015.

[80] Frank S Yao, Mei Zhao, and Hui Zhang. Two-stage hybrid flow shop scheduling with dynamic job arrivals. *Computers & Operations Research*, 39(7) :1701–1712, 2012.

[81] Yunqiang Yin, Shuenn-Ren Cheng, TCE Cheng, Chin-Chia Wu, and Wen-Hsiang Wu. Two-agent single-machine scheduling with assignable due dates. *Applied Mathematics and Computation*, 219(4) :1674–1685, 2012.

[82] Kejun Zhao and Xiwen Lu. Approximation schemes for two-agent scheduling on parallel machines. *Theoretical Computer Science*, 468 :114–121, 2013.

[83] Wei-ya Zhong and Long-hua Lv. Hybrid flowshop scheduling with interstage job transportation. *Journal of the Operations Research Society of China*, 2(1) :109–121, 2014.

[84] Hongli Zhu. A two stage scheduling with transportation and batching. *Information Processing Letters*, 112(19) :728–731, 2012.

[85] GI Zobolas, Christos D Tarantilis, and George Ioannou. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, 36(4) :1249–1267, 2009.