UNIVERSITE DU QUEBEC

MÉMOIRE PRESENTÉ À

L'UNIVERSITE DU QUEBEC A CHICOUTIMI

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

RUHLMANN CARINE

ÉTUDE DU PROBLEME DE JOB SHOP AVEC UN CONVOYEUR

3 mai 2007

Résumé

La densité des échanges commerciaux, ainsi que l'intensification de la concurrence qui a suivi, a conduit les entreprises à rationaliser leurs activités, particulièrement celles nécessitant des activités d'ordonnancement.

La théorie de l'ordonnancement est une discipline bien établie de l'optimisation combinatoire. Son champ d'investigation concerne les problèmes d'allocation, dans le temps, d'un ensemble limité de ressources par un ensemble de tâches, afin d'optimiser un ou plusieurs critères donnés. Sa popularité vient du fait qu'une multitude de situations peut être ramenée à cette problématique d'ordonnancement. Cela est dû, en grande partie, à la richesse de l'interprétation que peuvent avoir les termes ressources et tâches. Nous pouvons citer, entre autres, des applications dans l'industrie (réalisation de produits sur des machines), la santé (confection d'horaires), l'informatique (exécution de processus).

Dans ce mémoire de maîtrise, nous nous intéressons spécifiquement aux problèmes d'ordonnancement d'ateliers de production. Ainsi, notre étude porte sur l'ordonnancement de n tâches (jobs) sur m ressources (machines) dans un environnement de type job shop. Dans le modèle de job shop, chaque tâche doit passer sur l'ensemble des machines, à chaque fois pendant un temps connu à l'avance, et selon également un ordre donné. Le critère, que nous avons choisi pour évaluer la qualité d'une solution, est celui du makespan (la durée totale d'accomplissement des n tâches). Pour rester proche de la réalité industrielle, notre modèle incorpore un convoyeur chargé de transporter les tâches semi-finies d'une machine à une autre. Ce modèle peut être illustré par l'exemple d'une entreprise d'assemblage d'ordinateurs. Les machines assemblent divers éléments (cartes mères, disques durs, barrettes mémoires, etc) dans un boîtier. Un convoyeur déplace le boîtier entre les différentes machines. Suivant les spécifications de chaque ordinateur, chaque boîtier suit un chemin particulier. En effet, si un client souhaite acheter un boîtier contenant uniquement la carte-mère et l'alimentation, ce boîtier ne passera que sur deux machines.

Nous nous sommes restreint au problème de job shop à deux machines et un seul convoyeur. Notons que nous supposons que les deux machines possèdent des espaces de stockage de taille illimitée pour recevoir les travaux semi-finis. Notre but était au départ de trouver un algorithme polynomial pour résoudre ce problème. Or, il s'est avéré que même avec ce modèle restreint et simplifié, le problème est NP-difficile. Pour le résoudre, nous nous sommes alors tournés vers l'approche heuristique. Néanmoins, nous avons pu trouver des cas particuliers où ce problème est résoluble en un temps polynomial.

Notre démarche a été d'abord d'introduire brièvement les problèmes de la théorie de l'ordonnancement ainsi que quelques concepts de la NP-complétude,

avant d'aborder les différentes approches algorithmiques de résolution des problèmes d'ordonnancement. Dans une seconde étape, la littérature relative à cette problématique a été passée en revue. Ensuite, nous avons décrit plus en détail notre modèle de job shop ainsi que son fonctionnement. Nous avons discuté de l'influence du convoyeur sur la minimisation du critère du makespan. Nous avons également proposé une borne inférieure pour ce même critère. Enfin, nous avons discuté et proposé deux approches de résolution approchée. La première est constructive : trois algorithmes, basés sur des règles de priorité, ont été conçus. Les deux premières règles sont fondées sur l'appariement des travaux et la troisième est la règle bien connue de Jackson que nous avons modifiée ; ces règles ont une complexité temporelle en $O(n \log n)$. La seconde approche de résolution est itérative : l'algorithme de recherche avec tabous a été notre choix. Cette méthode étend à n travaux la méthode classique de résolution graphique à deux travaux. Finalement, nous avons entrepris une étude comparative de l'ensemble des algorithmes de résolution proposés. Les deux approches de résolution ont été simulées par un programme Java sur des instances générées de manière aléatoire à partir d'une distribution uniforme. Ces instances sont de tailles n = 10, 20, 50 et 200. Les temps d'exécution et de transport sont compris entre 0 et 50 unités de temps. La borne inférieure a été utilisée pour évaluer la qualité des solutions générées par chacune de ces heuristiques. Cette étude a montré que, parmi les algorithmes basés sur les règles de priorité, celui de Jackson donne des solutions de meilleure qualité. L'algorithme de recherche avec tabous donne, en moyenne, de meilleures solutions que les algorithmes basés sur les règles de priorité. Toutefois, les temps de calculs de cette approche sont de loin plus importants que ceux générés par les règles de priorité, surtout lorsque la taille des problèmes devient de plus en plus grande.

Nous concluons notre travail par la suggestion de nouvelles pistes à explorer pour des recherches futures.

Remerciement

Je souhaite tout d'abord, remercier mon directeur de recherche M Djamal Rebaïne, qui grâce à son soutien et ses idées judicieuses, m'a permis de mener à bien ce travail de recherche.

Je souhaite également remercier Jean-François Michaud et Yannick Eurin qui m'ont toujours soutenue et qui par leurs remarques m'ont permis d'avoir une vue plus objective de mon travail.

J'aimerais remercier Stephen Whitney avec qui j'ai apprécié de travailler, ainsi que le directeur du département Jean Rouette, et également Richard Bouchard, Johanne Labrie et le personnel de l'UQAC.

Table des matières

Liste des tableaux

Liste des figures

INTRODUCTION

Du fait de la concurrence, les industries, pour survivre, se doivent d'être toujours plus compétitives. Cette compétitivité passe par la réduction des coûts et un gain de productivité. Les industries se sont donc tournées vers la recherche et le développement dans le but d'améliorer leurs processus et notamment ceux de fabrication. Ainsi, la branche de l'optimisation combinatoire, qu'est la théorie de l'ordonnancement, de par son large champ d'investigation, permet de répondre à certaines des problématiques industrielles. En effet, cette théorie concerne les problèmes d'allocation, dans le temps, d'un ensemble limité de ressources par un ensemble de tâches [23]. La richesse d'interprétation des termes « ressource » et « tâche » permet l'application pratique à une multitude de cas.

Dans le présent mémoire, nous nous sommes intéressés aux problèmes dits d'ordonnancement d'atelier. Et plus particulièrement à l'ordonnancement de n tâches (jobs) sur m ressources (machines) dans un environnement de type atelier à cheminement multiple, aussi appelé « job shop » [8, 40] . De plus, pour rendre notre étude encore plus proche des cas réels, et pour répondre à la demande de gestion de la qualité totale [84], où les contraintes d'échéance sont essentielles, nous avons introduit un système de transport des tâches aussi appelé dans une partie de la littérature [33, 74, 86] robot ou convoyeur. Ces systèmes de transport des tâches sont également impliqués dans le chargement et le déchargement des machines dans ces ateliers. Un atelier à cheminement multiple avec transport est typiquement une usine de fabrication de semi-conducteurs ou un terminal portuaire, par exemple.

L'atelier qui va être étudié est constitué de deux stations et d'un convoyeur devant exécuter un ensemble de n tâches. Une station est composée d'une machine et de deux espaces de stockage. Ces espaces sont nommés respectivement, stock d'entrée pour l'espace d'attente des tâches devant être exécutées par la machine, et stock de sortie l'espace où les tâches venant d'être exécutées attendent le convoyeur

pour être transporté. Notre objectif est la construction d'un ordonnancement minimisant le temps total d'accomplissement (makespan), qui est le temps mis par la dernière tâche à sortir de l'atelier [66]. Le problème étant NP-difficile, nous abordons sa résolution d'une manière approchée, en proposant des règles de priorité ainsi qu'une méthode de recherche avec tabous.

Dès lors, notre démarche consistera à étudier l'influence du convoyeur sur le temps d'accomplissement total, autrement dit, de voir comment elle peut être atténuée, voire éliminée. De même, nous verrons jusqu'où peut être appliqué l'algorithme de Johnson [39] qui permet de déterminer en un temps polynomial un ordonnancement optimal pour un atelier à cheminement unique (un atelier à cheminement multiple est une généralisation de celui à cheminement unique) à deux machines.

Ainsi, au Chapitre 1, nous présentons une vue d'ensemble de la théorie de l'ordonnancement qui est un cadre permettant la description des différents modèles de problèmes d'ordonnancement. La classification de la difficulté de résolution d'un problème va nous amener à présenter les méthodes générales de résolution.

Dans le Chapitre 2, nous décrirons de façon générale l'atelier à cheminement multiple : modélisation, complexité et principales méthodes de résolution. Ensuite, les opérations de transport sont introduites, tout d'abord de façon globale puis spécifiquement dans le cadre d'un atelier à cheminement multiple. Finalement, nos objectifs de recherche seront présentés.

Dans le Chapitre 3, l'atelier à cheminement multiple à deux machines et un convoyeur est présenté en détail. Son étude commence par décrire certaines propriétés particulières de cet atelier. L'influence du convoyeur est alors examinée et une borne inférieure pour le temps d'accomplissement total est proposée.

Dans les Chapitres 4 et 5, deux méthodes de résolution approchées, ainsi que des résultats de simulation, sont présentés et discutés. La première de ces méthodes est une résolution par règles de priorité, où les tâches sont ordonnées suivant un ordre donné. La seconde méthode est une métaheuristique, la recherche avec tabous.

Finalement, la conclusion générale de ce mémoire est présentée pour discuter du bilan du travail effectué, ainsi que de nouvelles voies de recherches qui pourraient en découler.

CHAPITRE 1

INTRODUCTION À LA THÉORIE DE L'ORDONNANCEMENT

1.1 Introduction

L'ordonnancement est un champ d'investigation qui a connu un essor important ces dernières années [52], tant par les nombreux problèmes identifiés que par l'utilisation et le développement de techniques de résolutions. En effet, l'activité d'ordonnancement apparaît dans des disciplines aussi diverses que l'organisation du travail, l'allocation dynamique de ressources dans les systèmes d'exploitation des ordinateurs [19] ou la création d'emploi du temps [73].

L'ordonnancement est un processus de décisions [66], c'est-à-dire un enchaînement d'événements permettant d'arriver à un but souhaité. À ce processus est associé un environnement, qui est le milieu dans lequel se déroule le problème d'ordonnancement. Par exemple, dans le domaine de l'industrie, le processus de décision peut être l'organisation de la fabrication d'un produit et le but souhaité est de minimiser le temps total de fabrication. Son environnement est l'usine, le personnel et les machines. Pour résumer, l'ordonnancement est l'allocation de ressources au fil du temps dans le but de réaliser un ensemble de tâches en optimisant une ou plusieurs fonctions objectifs [3].

L'ordonnancement, comme le stipule le titre de ce chapitre, est aussi une théorie, c'est-à-dire un ensemble de principes, de modèles et de techniques permettant l'étude et la résolution du problème d'ordonnancement [23]. Cette théorie consiste donc en la recherche de modèles mathématiques et la mise au point de méthodes de résolution efficaces[1]. Elle constitue donc l'ensemble des méthodes

---

[1] L'efficacité est définie par rapport au temps d'exécution de l'algorithme résolvant le problème en question. Une définition sera donnée plus loin.

et techniques rationnelles d'analyse et de synthèse des phénomènes d'organisation utilisable pour élaborer de meilleures décisions.

Ce présent chapitre présente, dans un premier temps, les notions fondamentales de la théorie de l'ordonnancement en commençant par définir ce qu'est un problème, ses caractéristiques, notation et solution. Dans un second temps, nous nous intéressons à la complexité des problèmes d'ordonnancement et plus généralement, à la théorie de la complexité. Finalement, les différentes approches de résolution d'un problème sont présentées.

1.2 Problèmes d'ordonnancement

Les problèmes d'ordonnancement sont, de nature, très variés. Dans une entreprise manufacturière, par exemple, qui fabrique des voitures, il est crucial que les pièces soient assemblées de façon à minimiser le temps de fabrication. Pour cela, le gestionnaire doit déterminer l'ordre de passage des différentes pièces dans les différentes machines. Cet exemple est typiquement un problème d'ordonnancement.

1.2.1 Définition générale des problèmes d'ordonnancement.

Les éléments essentiels d'un problème d'ordonnancement sont les ressources, les activités, les contraintes et les objectifs. Les ressources sont les moyens humains ou matériels mis à disposition pour réaliser les activités demandées. Elles peuvent être de plusieurs types comme des lieux ou des processeurs, mais dans cette étude, pour coller à la réalité industrielle, les ressources sont considérées comme un ensemble de machines. De plus, nous supposons que cet ensemble est disponible au temps $t = 0$. Le terme machine est pris dans un sens global ; il inclut les outils et accessoires. Dans l'exemple précèdent, les ressources sont les machines-outils qui composent la chaîne de montage des véhicules.

Les activités sont les tâches (jobs) que l'atelier doit exécuter. Une tâche se définit comme un ensemble d'opérations. Une opération est le passage d'une tâche *j* sur une machine *i* et elle est notée *(i, j)* ou O_{ij}. La durée de traitement ou temps d'exécution d'une opération est son temps de passage sur une machine donnée ; c'est-à-dire le temps entre le moment où l'opération débute sur une machine et le moment où elle la quitte. Une opération est une entité élémentaire qui est décrite par :

- sa durée de traitement $p_{ij,}$
- sa date de début $t_{ij,}$
- sa date de fin d'exécution.

Le terme date désigne l'indication de l'époque où, un événement s'est produit ou va se produire. Cela peut donc être un horaire, une date ou un événement. Notons qu'une tâche est dite terminée si toutes ces opérations ont été exécutées. Toujours dans l'exemple précèdent, une tâche est la pose des portes. Les opérations composant cette tâche sont alors : saisir la porte, placer la porte dans les charnières, placer les boulons, viser les boulons, vérifier que la porte est bien fixée.

Les contraintes représentent les limitations imposées par l'environnement. Elles peuvent par exemple porter sur les opérations. Si l'exécution d'une opération en cours sur une machine ne peut être interrompue, alors le modèle est dit non préemptif. Sinon, l'opération en cours d'exécution peut être arrêtée au profit d'une autre jugée plus urgente, le modèle est dit alors préemptif. Les contraintes peuvent aussi être d'ordre temporel, des temps d'exécution identiques par exemple, ou porter sur les machines, qui ne sont pas toujours disponibles, et en général ne peuvent exécuter qu'une opération à la fois. De plus, l'environnement de travail peut comporter des contraintes de déplacement, comme l'emploi de convoyeurs. Une

tâche qui doit être exécutée avant une autre implique une contrainte, dite de précédence. Dans le modèle de chaîne de montage d'une porte, un exemple de contrainte de précédence est qu'il faut poser la porte avant de la visser.

Le ou les objectifs sont les critères que l'on souhaite optimiser. Ils seront décrits plus précisément par la suite.

1.2.2 Différents modèles

Pour permettre la classification des problèmes d'ordonnancement, ceux-ci ont été séparés en deux grands modèles suivant les tâches à exécuter [9] :

- les modèles dits statiques : le nombre de tâches à exécuter et leurs dates de disponibilité sont connus à l'avance et ne peuvent être modifiés par la suite,
- les modèles dynamiques : les tâches arrivent de façon aléatoire. Leurs nombres, ainsi que leurs dates de disponibilité ne sont pas connus à l'avance.

Pour ces modèles, deux autres précisions sont apportées. En effet, si les temps d'exécution sont fixés à l'avance alors le modèle est déterministe sinon il est stochastique.

Dans ce mémoire, nous allons restreindre notre étude au modèle statique et déterministe, car le problème de job shop que nous souhaitons étudier fait partie de ce modèle. Celui-ci est lui-même divisé suivant le nombre d'opérations des tâches.

1.2.2.1 Modèle statique à une opération

Ce modèle est lui-même scindé suivant le nombre de machines employées. Si ce nombre se borne à une unique machine alors ces problèmes appartiennent à la classe des problèmes à une machine (« single machine problems »). En revanche, si

l'environnement possède m machines en parallèle alors ces problèmes appartiennent à la classe des problèmes de machines parallèles (« parallel machine »).

1.2.2.1.1 Modèle à une machine

Son environnement est composé d'une seule machine, $m = 1$, qui est disponible d'une manière continue. Les n tâches à ordonnancer ne comprennent qu'une seule opération à effectuer sur la machine. Dans ce cas, l'opération est assimilée à sa tâche.

C'est un modèle simple, mais il possède des propriétés uniques. De plus, il est utile dans l'étude de modèles plus complexes, car il fournit des algorithmes et heuristiques de base. Souvent, les modèles plus complexes sont décomposables en sous-problèmes contenant le modèle à une machine. Par ailleurs, dans la pratique il existe des problèmes n'utilisant qu'une seule machine, comme par exemple une pointeuse dans une usine.

1.2.2.1.2 Modèle à machines parallèles

C'est une généralisation du modèle à une machine. Dans ce cas, m machines sont placées en parallèle, pour effectuer les différentes tâches simultanément. Les tâches n'ont toujours qu'une opération et le problème central est de trouver une partition de ces tâches sur les machines. Dans ce modèle, trois cas sont à distinguer selon la vitesse des machines [52] :

- si la vitesse des m machines est identique, alors ce problème est dit à machines identiques (« identical machines problems »), noté P,
- s'il y a un coefficient de proportionnalité entre la vitesse des machines, alors ce problème est dit à machines uniformes (« uniform machines problems »), noté Q,

– si toutes les machines ont des vitesses indépendantes alors ce problème est dit à machines indépendantes ou non reliées (« unrelated machines problems »), noté *R*.

Un exemple de machines parallèles est le passage aux caisses dans un supermarché. Dans ce cas, les machines sont les caisses et les tâches sont les clients avec leurs courses qui attendent. Selon la vitesse des caissières, nous pouvons parler des trois modèles cités ci-dessus.

1.2.2.2 Modèle statique à plus d'une opération

Cette classe contient les problèmes ayant plus d'une machine et dont les tâches, pour être considérer terminées, doivent passer par plusieurs machines. Ces tâches sont donc composées de plusieurs opérations. Cette classe de problèmes est appelée problèmes d'atelier (« shop problems »). On en distingue trois modèles de base [23].

- Les problèmes d'atelier à cheminement libre (« open shop ») : chaque tâche doit passer par toutes les machines. De plus, l'ordre de passage des opérations sur les machines n'est pas connu au départ. Un exemple de problème d'atelier à cheminement libre est le problème de faire ses courses. Les tâches sont les clients du magasin et les machines sont les rayons. Les clients doivent aller dans les rayons leur fournissant les articles dont ils ont besoin. L'ordre dans lequel les clients obtiennent les articles est sans importance.

- Les problèmes d'atelier à cheminement unique (flow shop) : dans ces ateliers, les machines sont en série, c'est-à-dire à la suite des unes des autres et l'ordre de passage sur les machines est le même pour toutes les tâches. Ce chemin est donc une donnée du problème. Il faut donc trouver l'ordre d'exécution des opérations des tâches. Un exemple typique d'atelier à cheminement unique est une chaîne de

montage. Il est à noter un cas particulier : si l'ordre de passage des opérations sur chaque machine est identique, alors on se trouve face à un problème d'atelier à cheminement unique, dit de permutation.

- Les problèmes d'atelier à cheminement multiple (« job shop ») : l'atelier à cheminement multiple reprend les hypothèses de celui à cheminement unique, mais chaque tâche a son propre chemin à travers les machines. De plus, dans ce modèle, une tâche peut passer plusieurs fois sur une même machine [23]. Ce phénomène est appelé recirculation. Un exemple est la consultation d'un patient à l'hôpital. Les patients sont les tâches et les praticiens sont les machines. Un patient peut très bien être ausculté par un médecin puis aller voir le radiologue pour passer une radio et ensuite revenir voir le même médecin pour la suite de sa consultation.

Notons qu'il est aisé de créer d'autres modèles d'ordonnancement en combinant les modèles qui viennent d'être présentés. Par exemple, dans un atelier à cheminement unique, une machine, appelée dans ce cas station, peut être remplacée par un ensemble de machines mises en parallèles : ce modèle sera alors appelé atelier à cheminement unique hybride [70].

Par la suite, par soucis de lisibilité et puisque ce sont les termes en usage dans le domaine, seuls les termes job shop, flowshop et openshop seront employés.

1.2.3 Critères d'optimisation

Les critères à optimiser sont exprimés sous forme d'une fonction appelée fonction objectif : c'est le but que l'on cherche à atteindre en résolvant un problème d'ordonnancement. La fonction objectif est une fonction de minimisation, ou de maximisation d'un ou plusieurs critères. Ces critères sont nombreux ; ci-dessous, une liste non exhaustive :

- Le temps d'accomplissement total (appelé communément makespan), noté C_{max}, représente le temps d'accomplissement de la dernière tâche terminée,
- Le retard maximum, noté L_{max}, représente la violation la plus importante des dates d'échéance de toutes les tâches. Une date d'échéance est la date à laquelle la tâche doit être terminée,
- Le temps de retard total pondéré, noté $\sum w_j T_j$ où le terme w_j est le poids d'une tâche j et T_j est le temps de retard de j. Le poids d'une tâche dénote son importance par rapport aux autres tâches relativement à un ou plusieurs critères particuliers. Dans ce cas, c'est le temps de retard total en fonction de l'importance des tâches qui en général est à minimiser,
- le temps d'accomplissement pondéré de toutes les tâches $\sum w_j C_j$. Souvent, cette somme est appelée « *flow time* ».

Par la suite, il ne sera essentiellement question que du makespan comme fonction objectif. Cette fonction étant le critère le plus souvent utilisé pour évaluer le coût d'un ordonnancement [37].

1.2.4 Notation

Pour identifier facilement et de façon concise un problème d'ordonnancement, une notation a été introduite [10, 30]. Un problème est décrit par le triplet $\alpha \mid \beta \mid \gamma$. Le paramètre α représente l'environnement machine, il peut être décomposé en $\alpha = \alpha_1 \alpha_2$ où α_1 représente le type de machine ; par exemple, P indique des machines fonctionnant en parallèle et α_2 est le nombre de machines de l'environnement. Le paramètre β représente l'ensemble des contraintes et caractéristiques du processus. Enfin, le paramètre γ désigne la (les) fonction(s) objectif(s) à résoudre, sauf mention, ce sont des minimisations.

Ci-dessous, quelques exemples de notation :

- $1|r_j, prmp|\sum w_j C_j$: dénote un problème à une machine dont les tâches ne sont disponibles qu'à la date r_j. Lors de l'exécution d'une tâche, la préemption notée *prmp* est possible. Le but est de minimiser la somme des temps d'accomplissements pondéré ou $\sum w_j C_j$,
- $P3||L_{max}$: représente le problème à trois machines parallèles identiques, noté *P3,* et sa fonction objectif est la minimisation du retard maximum L_{max}. Il n'y a pas de contraintes particulières,
- $F_m|prmu|C_{max}$: représente le problème général de flowshop, noté F_m, de permutation *prmu* ayant pour objectif de minimiser le makepan C_{max},
- $J2||T_{max}$: représente le problème de job shop à deux machines, noté *J2*, dont l'objectif est la minimisation du retard total.

1.2.5 Représentation d'une solution

Comme il est dit dans [23], résoudre un problème d'ordonnancement, c'est trouver une adéquation entre une tâche à effectuer, et les moyens disponibles pour sa réalisation. Cela consiste donc à fixer les dates de début t_{ij} de chaque opération O_{ij} et leur attribuer des ressources, de façon à ce que les contraintes soient respectées. Pour cela, il faut déterminer l'ordre de passage des tâches sur chaque machine. Cet ordre de passage est appelé séquence. Une séquence est une permutation de n tâches ou, autrement dit, l'ordre dans lequel les n tâches sont allouées à une machine. Un ordonnancement est défini grâce aux séquences de tâches, et leurs dates de début.

En général, un ordonnancement est représenté sous forme d'un diagramme, appelé diagramme de Gantt. Il permet de visualiser, en fonction du temps, l'exécution des différentes opérations. Il a été développé par Gantt, dans les

années 1910. Dans un diagramme de Gantt, les différentes opérations sont représentées en lignes et en colonnes se trouvent les machines. Le temps d'exécution d'une opération est modélisé par une barre horizontale dont l'extrémité gauche est située sur la date prévue de démarrage et l'extrémité droite sur la date prévue de fin de réalisation. Dans les cas où les opérations s'enchaînent de façon séquentielle et qu'il y a contrainte de précédence, les relations d'antériorité sont modélisées par une flèche partant de l'opération en amont vers l'opération en aval.

Par exemple, si un gestionnaire souhaite représenter l'ordonnancement de la pose d'une porte de voiture dans le cadre de la chaîne de montage déjà mentionnée. Comme décrit dans le Tableau 1, la tâche de pose d'une porte est composée de cinq opérations. Puisque dans cet exemple, il n'y a qu'une tâche, i est la machine sur laquelle l'opération j devra se réaliser.

Opérations	**Notation O_{ij}**	**Durée p_{ij}**	**Date de fin d'exécution t_{ij} (minutes)**
Saisir la porte	O_{11}	p_{11} = 1 minute	t_{11} = 1
Placer la porte dans les charnières	O_{12}	p_{12} = 2 minutes	t_{12} = 3
Placer les boulons	O_{21}	p_{21} = 2 minutes	t_{21} = 5
Viser les boulons	O_{22}	P_{22} = 0.5 minute	t_{22} = 5.5
Vérifier que la porte est bien fixée	O_{13}	p_{13} = 1 minute	t_{13} = 6.5

Tableau 1 : Description des opérations de la pose d'une porte

Les dates de fin d'exécution sont calculées comme suit : $t_{i(j+1)} = t_{ij} + p_{ij}$ car les opérations sont supposées se dérouler sans interruption. Pour réaliser ces opérations, il faut deux machines, notées M_1 et M_2. Elles sont disponibles au temps $t = 0$. Les

opérations : saisir la porte notée O_{11}, placer la porte dans les charnières, notée O_{12}, et vérifier que la porte est bien fixée, notée O_{13}, s'effectuent sur la machine M_1. Quant aux opérations, placer les boulons, notée O_{21}, et les visser, notée O_{22}, c'est la machine M_2 qui s'en charge. Comme le montre la Figure 1, on obtient le diagramme de Gantt suivant :

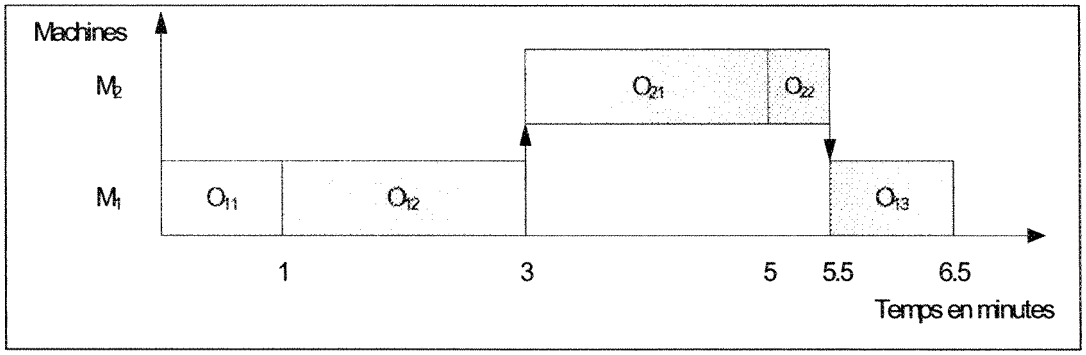

Figure 1 : Diagramme de Gantt

Grâce au diagramme de Gantt, si le gestionnaire de la chaîne de montage souhaite savoir combien de temps prend la pose d'une porte, il lui suffit de lire le diagramme et il obtiendra un temps d'accomplissement de *6.5* minutes. Ceci n'est qu'un exemple très simple et il est fort vraisemblable que dans la réalité les deux machines travaillent sur d'autres véhicules, au lieu de rester inoccupées comme on le voit ici.

1.2.6 Caractéristiques des ordonnancements

Comme l'a mentionné Mellor [56], il existe de nombreux critères, bien souvent conflictuels, qui permettent l'évaluation de la qualité d'un ordonnancement. Ces critères d'évaluation sont rangés dans deux classes : les critères réguliers et les non réguliers.

Définition 1 : Un critère R est régulier s'il est une fonction croissante des dates de temps d'accomplissement de chaque tâche, c'est-à-dire si

$$C_1 \leq C_1', C_2 \leq C_2', ..., C_n \leq C_n' \Rightarrow R(C_1, C_2, ..., C_n) \leq R(C_1', C_2', ..., C_n')$$

C_j et C_j' étant les dates de temps d'accomplissement d'une tâche j dans deux ordonnancements.

Il est facile de montrer que le makespan est un critère régulier (voir par exemple French [24]).

Grâce à cette définition, s'il existe deux ordonnancements S et S' telles que toutes les tâches de S se terminent avant celles de S' alors, pour un critère régulier, il est possible de dire que S est au moins aussi bon que S'. Cette évaluation n'étant pas toujours vérifiée si le critère n'est pas régulier.

Définition 2: Un ordonnancement « semi-actif » est un ordonnancement dans lequel aucune tâche ne peut commencer plus tôt sans changement de l'ordre de passage aux machines, ni violation des contraintes.

Dans le cadre de la minimisation d'un critère régulier, il est possible grâce au théorème suivant [24] de limiter la recherche à un ordonnancement semi-actif.

Théorème 1 : Pour minimiser un critère régulier, il suffit de se limiter aux ordonnancements semi-actifs.

Concernant d'autres types d'ordonnancement comme les ordonnancements actifs ou sans délai, le lecteur peut se référer à Pinedo [66].

1.3 Brève introduction à la théorie de la complexité

L'expérience montre que certains problèmes d'ordonnancement sont résolus en des temps records et ce, même pour un grand nombre de données traitées. Alors que pour d'autres, les seules résolutions existantes ne peuvent traiter, dans un temps raisonnable, qu'un nombre restreint de données. La théorie de la complexité (à ne pas confondre avec la complexité d'un algorithme) a donc été établie pour catégoriser la difficulté des problèmes. Elle a été développée à partir des travaux de Cook [17] et de Karp [17, 41]. Elle est un outil important dans la théorie de l'ordonnancement.

1.3.1 Efficacité d'un algorithme

Une méthode de résolution est appelée algorithme. Il peut être défini de la manière suivante. « Un *algorithme* de résolution d'un problème *P* donné est une procédure décomposable en opérations élémentaires, transformant une chaîne de caractères représentant les données de n'importe quel exemple du problème *P* en une chaîne de caractères représentant les résultats de *P* » [72].

Une fois qu'un algorithme de résolution est trouvé, il faut déterminer son efficacité. Cela revient à suivre l'évolution du nombre d'opérations de base en fonction de la quantité de données à traiter. La complexité temporelle d'un algorithme (dans le pire cas) est le temps mis pour transformer l'ensemble de données de taille n en un ensemble de résultats. Elle est représentée par la fonction $T(n)$. Pour simplifier cette introduction, on ne s'intéressera qu'au comportement asymptotique de la complexité temporelle d'un algorithme décrite par la notation O.

Définition 3: $T(n) = O(g(n))$ s'il existe c et n_0 des constantes positives telles que $T(n) \leq c.g(n)$ pour tout $n \geq n_0$ [75].

Définition 4 : Un problème est facile s'il existe un algorithme de complexité temporelle polynomiale $T(n) = O(n^k)$ où k une constante pour le résoudre. C'est-à-dire si $T(n)$ est borné par une fonction polynômiale.

Pour mettre en évidence l'importance des complexités polynomiales, prenons l'exemple d'un programme de n opérations sur un ordinateur ayant un processeur exécutant 10^6 instructions par seconde. Si le programme est de complexité $T(n)$ alors il va exécuter $T(n)$ opérations, donc la durée estimée du programme sera calculée par $d = \frac{T(n)}{1000000}$ secondes.

T(n)	n = 10	n = 100	n = 1000
n	0,00001	0,0001	0,001
n^2	0,0001	0,01	1
n^3	0,001	1	1000,00
2^n	0,001024	$1,27.10^{24}$	$1,07.10^{295}$
3^n	0,059049	$5,15.10^{41}$	∞

Tableau 2 : Comparaison de complexités en secondes

Le Tableau 2, ci-dessus, montre la durée estimée pour un programme de 10, 100 et 1000 opérations. Cette comparaison atteste que même pour des instances de problème de taille moyenne les algorithmes de complexité dite exponentielle (non polynomiaux) ne donnent plus de résultats dans un temps raisonnable. De plus, les problèmes de complexité polynomiale bénéficient des progrès technologiques. En effet, si l'on construit une machine 1000 fois plus rapide que celles actuelles, elle pourra résoudra un problème de taille $1000n$ si le problème est résolu par un algorithme en $O(n)$ et seulement de taille $n+10$ si le problème est résolu par un algorithme en $O(2^n)$. Ainsi, les algorithmes polynomiaux sont dits efficaces.

Jusqu'ici nous n'étions pas intéressés au codage des données. Un algorithme est de complexité pseudo-polynomiale si sa complexité temporelle est polynomiale pour un codage unaire. Un codage est unaire si par exemple *3* est codé par *<111>*.

Pour un problème donné, si aucun algorithme efficace n'est trouvé, alors se pose la question de savoir s'il n'en existe pas du tout ou si aucun n'a encore été découvert. La théorie de la complexité ne répond pas à cette question, mais permet une certaine classification de la difficulté de résolution des problèmes.

1.3.2 Classes de complexité

Le classement des problèmes passe par l'utilisation de problèmes de décision. Cet outil permet le rangement des problèmes suivant deux grandes classes : la classe P et la classe NP.

1.3.2.1 Problème de décision

Précisons d'abord que « Devant la difficulté de classer les problèmes tels quels, la théorie de la complexité se limite à la seule étude des problèmes de décisions » [25].

Ainsi, pour chaque problème étudié, un problème de décision lui est associé. Celui-ci, aussi appelé problème de reconnaissance, est un problème dont la solution ne peut n'être que OUI ou NON (ou VRAI/FAUX). Pour un problème d'optimisation, dont la fonction objectif est la minimisation, est associé un problème de décision défini par la question : existe-t-il une solution telle que son évaluation est majorée par un nombre donné ? Dans le cas d'un problème de maximisation, l'évaluation de la solution sera alors minorée. Par exemple, dans un problème de type job shop $J_m \mid\mid C_{max}$, donc un problème ayant une fonction objectif à minimiser,

le problème de décision associé est : existe-t-il un ordonnancement pour un job shop où le makespan est inférieur à une certaine valeur z.

Ce qui est importe lors de la définition d'un problème de décision c'est qu'elle permette de savoir si une solution existe. Si celle-ci existe, et est obtenue grâce à un algorithme polynomial alors il existera un algorithme polynomial résolvant le problème d'optimisation associé. De même, si un algorithme polynomial est connu pour un problème d'optimisation alors il existe aussi un algorithme polynomial pour son problème de décision. Cette propriété nous permet de classer les problèmes grâce à leurs problèmes de décision. Ainsi, ce seront les problèmes de décisions qui seront contenus dans les différentes classes, mais par abus de langage, tel problème d'optimisation combinatoire sera dit appartenant à telle classe.

1.3.2.2 Classes P et NP

La première classe de complexité que nous présentons est la classe P ; P pour Polynomiale. Elle contient tous les problèmes dits « faciles » [72]. Pour être plus précis, la classe P contient tous les problèmes de décision pour lesquels il est possible de trouver une réponse oui/non en un nombre d'étapes, borné polynomialement en fonction de la taille n du problème considéré. En d'autres termes, cette classe contient tous les problèmes résolubles par des algorithmes efficaces. Le problème $J2||C_{max}$, par exemple, appartient à la classe P.

Il existe des problèmes pour lesquels aucun algorithme polynomial n'est connu, la classe NP a donc été créée pour les classer. Cette classe a un nom trompeur, NP ne correspond pas à Non Polynomial, mais à Polynomial Non déterministe ou « Nondeterministic Polynomial time » en anglais. Elle contient les problèmes de décision pour lesquels, il est possible de leurs associer un ensemble de

solutions potentielles tel qu'on puisse vérifier en un temps polynomial que chaque solution satisfait la question posée.

La classe NP contient donc des problèmes plus « difficiles » que la classe P. Mais même dans la classe NP, il y a des problèmes encore plus difficiles ; par soucis de clarté, la NP-complétude a été créée.

1.3.3 Classe des problèmes NP-complets

Cette classe (NPC) concerne les problèmes les plus difficiles de la classe NP. Elle est basée sur la notion de réduction. En effet, un problème de décision est dit NP-complet si tout problème de la classe NP peut se réduire polynomialement à lui.

1.3.3.1 Concept de réduction

La réduction convertit un problème en un autre dans le but de pouvoir utiliser la solution du second problème pour résoudre le problème initial. Pour bien comprendre ce qu'est la réduction, prenons un exemple fort simple. Vous arrivez dans une ville inconnue et vous devez vous rendre à un endroit précis. Vous cherchez votre chemin ; dans ce but vous allez chercher une carte de la ville. Le problème de chercher son chemin a donc été réduit à celui de trouver une carte de la ville.

Dans les cas de problème d'optimisation, souvent un problème est un cas particulier, équivalent, ou plus général d'un autre problème. Dans cette configuration, un algorithme fonctionnant bien pour un problème peut aussi donner des résultats pour un autre problème après quelques modifications.

La réduction sert à résoudre un problème à l'aide d'un second problème, mais permet aussi de montrer la difficulté de résolution d'un problème en le transformant en un autre dont la complexité est déjà connue.

Soient *A* et *B* deux problèmes de décision quelconques. *A* est le problème à résoudre et *B* le problème qui possède un algorithme de résolution en temps polynomial. *A* est dit réductible à *B* si et seulement s'il existe une fonction *f*, de complexité polynomiale, qui transforme les instances de *A* en celles de *B* de telle manière que la réponse pour *A* soit oui si et seulement si la réponse pour *B* est oui.

Une réduction est donc une transformation qui à une instance du problème *A* fait correspondre une instance du problème *B* qui a la même réponse. De plus, une oui-instance est une instance d'un problème de décision dont la réponse est oui. Dans le cas d'une réduction polynomiale, comme le montre la Figure 2, la transformation, notée *f*, transforme chaque oui-instance *i* de *A* en une oui-instance de *f(i)* de *B*. *f* étant polynomiale [75].

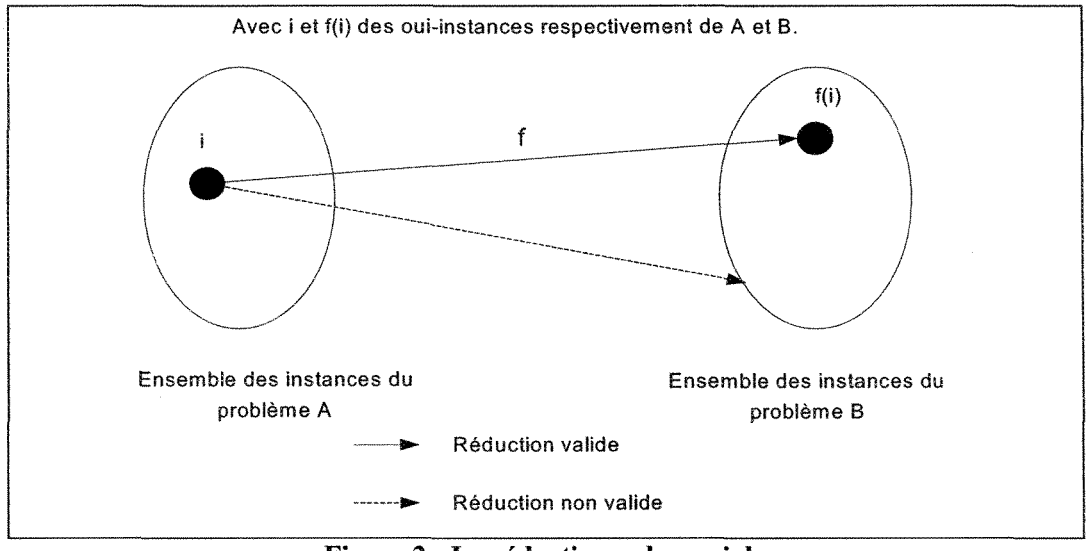

Figure 2 : La réduction polynomiale

Notation : la réduction polynomiale de A en B est notée $A \propto B$.

L'intérêt de la réduction est le suivant : si un problème A se réduit en temps polynomial à B et si B est résoluble par un algorithme polynomial alors il existe aussi un algorithme polynomial pour A. Cette propriété est transitive, si $A \propto B$ et $B \propto C$ alors $A \propto C$. De même, s'il n'existe pas d'algorithme polynomial pour A alors il n'existera pas non plus d'algorithme en temps polynomial pour B, puisque A est un cas particulier du problème B.

1.3.3.2 Existence des problèmes NP-complets

Le concept de réduction permet de montrer l'existence de la NP-complétude d'un problème. Il est clair que la classe P est incluse dans celle de NP. Or, la théorie de la complexité s'intéresse à l'étude de la classification de problème et les frontières existantes entre ces différentes classes. Sa problématique centrale, la plus connue, est de savoir si P = NP. En d'autres termes, s'il est toujours facile de valider une solution donnée, est-il aussi facile de trouver une solution ? L'attente de la plupart des informaticiens et mathématiciens est que cette égalité soit fausse. Mais il n'existe pas encore de preuve démontrant que l'assertion « P = NP » soit fausse.

Comme mentionné précédemment, un problème de décision est dit NP-complet si tout problème de la classe NP peut se réduire polynomialement à lui. Cette définition est très forte, car s'il est possible de trouver un algorithme en temps polynomial pour un seul problème NP-complet, alors de par le concept de réduction, tous les problèmes de NP seront résolus en un temps polynomial, *i.e* P = NP. Mais existe-t-il, ne serait-ce qu'un seul problème NP-complet ? Cook, en 1971, grâce au problème SAT [17], a assuré l'existence de la NP-complétude.

La Figure 3 montre l'ensemble des différentes classes [72].

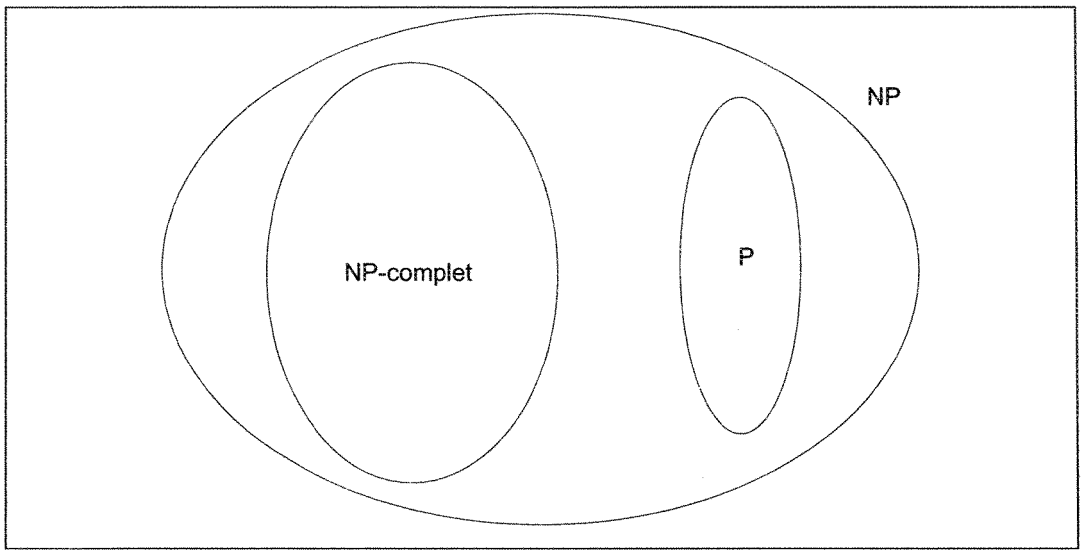

Figure 3 : Les différentes classes

Il est à noter l'existence d'une « appellation » spécifique concernant certains problèmes d'optimisation combinatoire. En effet, ceux-ci sont nommés NP-difficiles, si les problèmes de décision qui leur sont associés sont NP-complets.

Certains auteurs font une distinction entre les problèmes faiblement NP-difficiles et les problèmes fortement NP-difficile. Ainsi, pour les problèmes faiblement NP-difficiles, des algorithmes pseudo-polynomiaux (polynomiaux en fonction de la taille unaire des données) les résolvants, sont connus. Les différentes classes, déjà présentées, ne sont que les principales dans la théorie de la complexité, il est laissé au lecteur le loisir de se référer à [11, 53], s'il souhaite plus de détails.

1.3.4 Hiérarchie de complexité pour les problèmes d'ordonnancement

Suite à la hiérarchie de la complexité, et puisque les problèmes d'ordonnancement sont aussi des problèmes d'optimisation, la notion de réduction

introduite plus tôt peut leur être appliquée. Une hiérarchisation des problèmes d'ordonnancement est donc possible.

Par exemple, le problème a une machine ayant pour objectif de minimiser la somme des temps d'accomplissement $1||\sum C_j$ est un cas particulier du problème à une machine dont l'objectif est de minimiser la somme pondérée des temps d'accomplissement $1||\sum w_j C_j$. $1||\sum C_j$ se réduit donc à $1||\sum w_j C_j$. Un algorithme résolvant $1||\sum w_j C_j$ peut alors, être appliqué au problème $1||\sum C_j$.

Plusieurs hiérarchies de complexité sont possibles suivant la ou les caractéristiques étudiées. Une hiérarchie utile à connaître est celle générale des problèmes ayant des configurations machines différentes dans des environnements identiques, présentée par la Figure 4. 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 [66] .

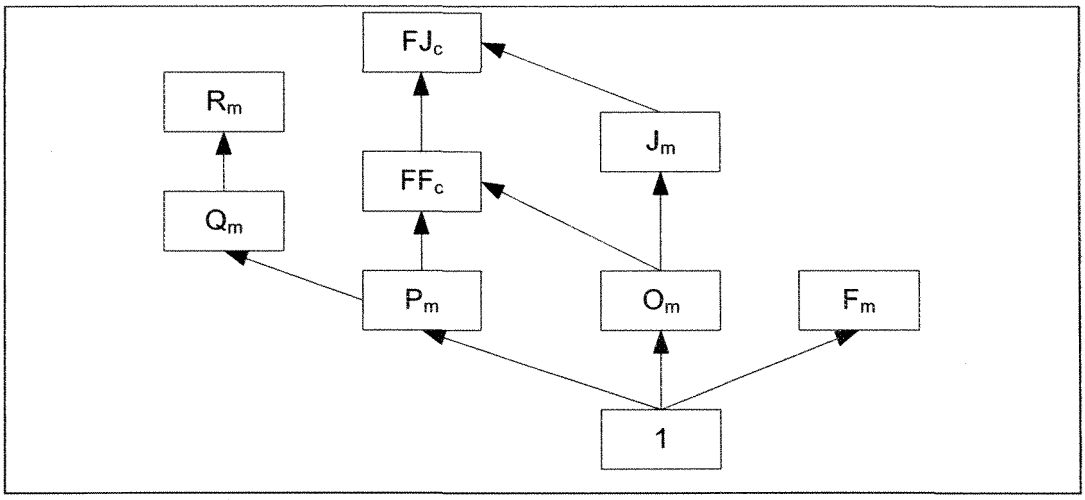

Figure 4 : Hiérarchie de la complexité en fonction de l'environnement machine

La Figure 5, ci-dessous, montre la hiérarchie de complexités de quelques problèmes ayant pour point commun leur fonction objectif (minimisation du makespan).

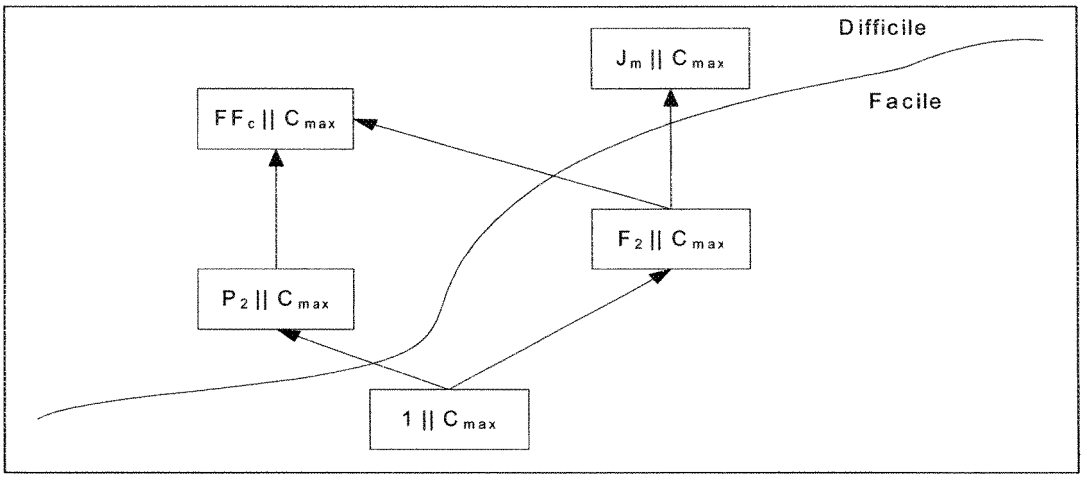

Figure 5 : Hiérarchie de complexité des problèmes ayant le makespan comme fonction objectif

Dans cette situation, le problème de flowshop à deux machines, qui est facile, pourra par ses méthodes de résolution servir à résoudre le problème de job shop à m machines. D'autres hiérarchies existent, pour de plus amples informations, le lecteur peut se référer à Pinedo [66] et Timkovsly [80].

1.4 Approches de résolution

L'étude de la complexité des problèmes d'ordonnancement conduit au choix de leurs méthodes de résolution. Il est donc essentiel de savoir si un problème est « facile » ou « difficile ». Si un problème semble NP-complet, il est nécessaire d'en établir la preuve. La liste suivante présente les étapes usuelles de l'établissement de cette preuve :

- transformer le problème d'optimisation en problème de décision P,

- montrer que P en dans NP,
- choisir un problème P' étant NP-complet, permettant de faire la réduction,
- construire une fonction f de P' vers P de telle sorte que I est une oui-instance de P' si et seulement si $f(I)$ est une oui-instance de P,
- montrer que f est une réduction polynomiale.

Il existe de nombreuses techniques pour prouver la NP-complétude. Certains types de preuves apparaissent assez régulièrement comme la restriction, le remplacement local et la construction de composants. Pour plus de détails, le lecteur peut se référer à la Section 3.2 de Garey et Johnson [25].

La notion de séquence implique que les problèmes d'ordonnancements sont fortement combinatoires. En effet, le nombre de solutions dans le pire des cas est borné par $(n!)^m$ avec n le nombre de tâches et m le nombre de machines. Ce qui, même pour de petites instances comme cinq tâches et trois machines, entraîne un nombre astronomique de solutions, ici *1728000.* Face à cette « explosion combinatoire », il est nécessaire d'utiliser des méthodes de résolutions efficaces dans le but d'avoir des solutions en des temps raisonnables. La Figure 6 résume la méthodologie de résolution d'un problème d'optimisation combinatoire donné [9].

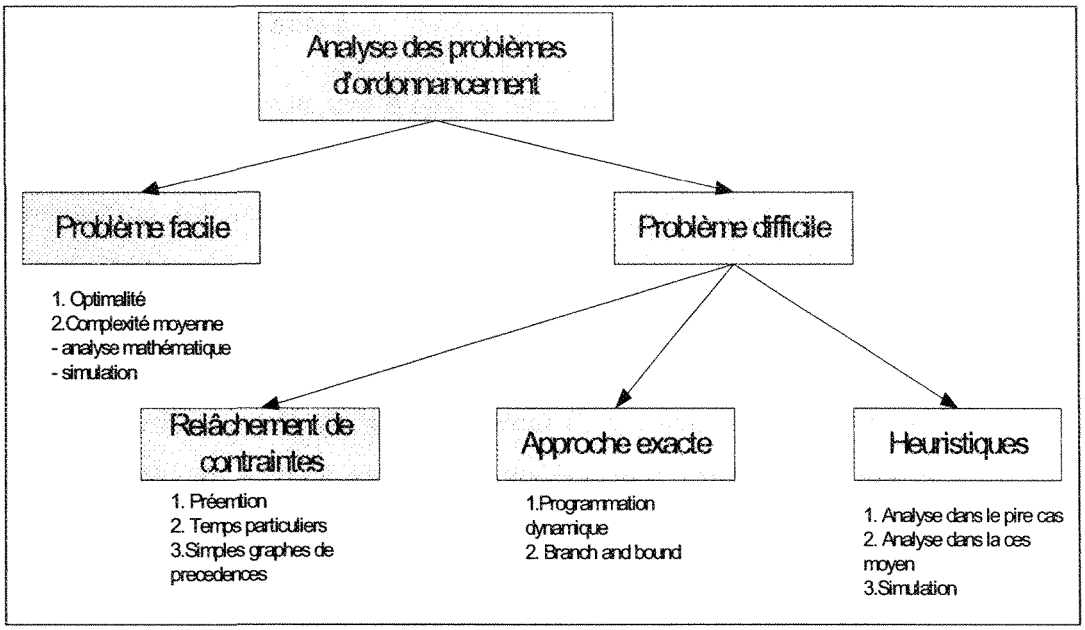

Figure 6 : Analyse des problèmes d'ordonnancement

1.4.1 Problème facile

Plusieurs techniques existent pour montrer qu'un problème donné est facile à résoudre. L'essentiel est d'exhiber un algorithme polynomial. Nous pouvons citer comme méthode de résolution, les algorithmes constructifs, la modélisation à l'aide de la programmation linéaire et la programmation dynamique.

1.4.2 Problème difficile

Quand le problème étudié NP-difficile, deux approches sont utilisées pour la résolution de cette classe de problèmes.

1.4.2.1 Méthodes exactes

Ces méthodes permettent d'obtenir une solution optimale. Deux méthodes seront vues, la méthode de séparation et évaluation progressive (SEP) et la programmation dynamique.

1.4.2.1.1 Séparation et évaluation ou « branch and bound ».

Cette méthode consiste en une évaluation partielle et progressive des solutions du problème [23]. De fait, au cours de l'évaluation des solutions, celles qui ne mènent pas à la solution recherchée par rapport à certaines contraintes, sont éliminées. Dans ce dessein, une fonction permettant de mettre une borne sur certaines solutions pour les exclure ou les maintenir comme solutions partielles est utilisée. Comme son nom l'indique, la méthode est composée de deux procédures.

Procédure de séparation : le problème est scindé en sous problèmes, eux-mêmes scindés en sous problèmes et ainsi de suite récursivement... Suite à ce découpage, par convention, un arbre est construit, appelé arbre d'évaluation, où les nœuds sont les sous problèmes et les arcs issus d'un même sommet représentent la décomposition d'un problème en sous problèmes de plus petites tailles. Par exemple, soit P^0 le problème initial qui consiste à vouloir construire la séquence de n tâches sur une machine. Pour commencer, P^0 est scindé en n sous problèmes P_1^1, .., P_n^1 où P_j^i est le sous problème de niveau i où la tâche j est assignée à la première position et où la sous séquence $2,..,n$ reste à assigner. Ces sous-problèmes seront, eux aussi, scindés. Ainsi, le sous-problème P_1^1 sera scindé en $P_{11}^2,..,P_{n1}^2$.

L'arborescence générale, présentée par la Figure 7, sera alors construite.

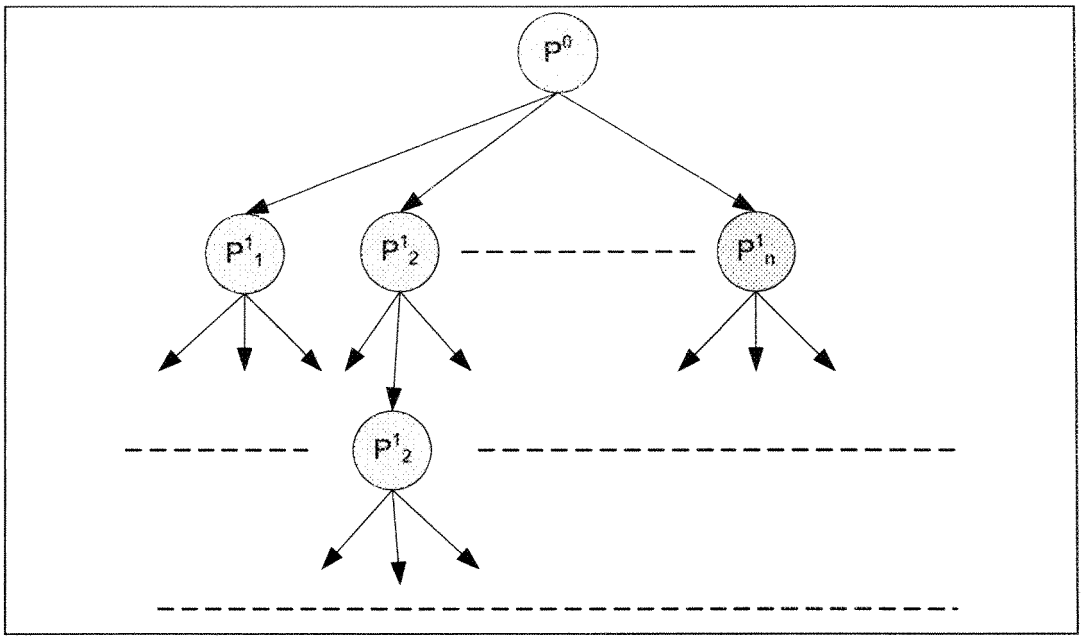

Figure 7 : Arborescence de séparation et d'évaluation

La construction de l'arbre total donnerait *n!* sous problèmes au niveau *n* et, chacun de ses *n!* sous problèmes correspondrait à une solution du problème racine P^0. L'arbre constitue donc l'énumération de toutes les séquences possibles des *n* tâches sur une machine. Pour ne pas avoir besoin de construire la totalité de l'arbre, mais pour tout de même trouver une solution au problème, la procédure d'évaluation est utilisée.

Procédure d'évaluation : cette procédure permet de répondre à la question : comment être sûre que la solution optimale ne peut être l'un des descendants d'un nœud particulier de l'arbre ?

La réponse a été donnée indépendamment par Land et Doig [46] et par Murty, Karel et Little [60]. En effet, il est toujours possible de trouver une solution réalisable par exemple en utilisant une heuristique. Donc, s'il est possible de calculer une « borne » constituée de la meilleure solution possible du sous arbre de

racine le nœud considéré, alors la valeur de la fonction objectif de ce nœud peut être comparée avec la borne. Ainsi si la valeur de la fonction objectif est inférieure, pour une minimisation (respectivement supérieur pour une maximisation), alors dans le sous-arbre considéré il n'y a pas de meilleure solution. L'évaluation de cette branche peut alors être arrêtée.

De plus, l'exploration de l'arbre s'arrête également lorsque l'une des conditions suivantes est satisfaite :

- le sous problème considéré n'a pas de solution réalisable,
- il n'y a plus de sous problèmes à sonder.

Une technique possible pour l'évaluation est de calculer la borne initiale grâce à une heuristique. Ensuite, lors de la création d'un nœud de l'arbre, évaluer la solution et retenir sa valeur si celle-ci est inférieure dans le cas d'une minimisation (supérieure dans le cas d'une maximisation) à la borne actuelle. Cette borne est alors mise à jour avec la nouvelle valeur minimale (ou maximale).

Une fois que tout l'arbre est séparé et évalué alors la solution optimale est facilement déduite. La méthode de séparation et d'évaluation progressive ne résout pas tous les problèmes, mais c'est un moyen de décomposer un problème en éléments permettant de le résoudre, il en est de même pour la programmation dynamique.

1.4.2.1.2 Programmation dynamique

Cette méthode a été élaborée à partir des travaux de Bellman dans les années cinquante [6, 7]. Elle peut être décrite comme récursive ou multi-niveau. En effet, les problèmes d'optimisation sont considérés, dans ce cas, comme des processus de décision multi-niveaux.

La programmation dynamique repose sur le principe d'optimalité, dit de Bellman : « Toute politique optimale est composée de sous politiques optimales ». C'est-à-dire que les solutions de petits sous problèmes servent à résoudre un problème plus important. Les sous problèmes étant les subdivisions du problème considéré. La solution du problème recherchée est donc dépendante des solutions des sous problèmes. Une particularité de la programmation dynamique, comme le montre la Figure 8, est qu'un sous problème peut être utilisé dans la solution de deux autres sous problèmes différents. De plus, cette méthode est ascendante, elle part des plus petits sous problèmes et remonte vers les problèmes plus importants, afin d'éviter la duplication de leur résolution.

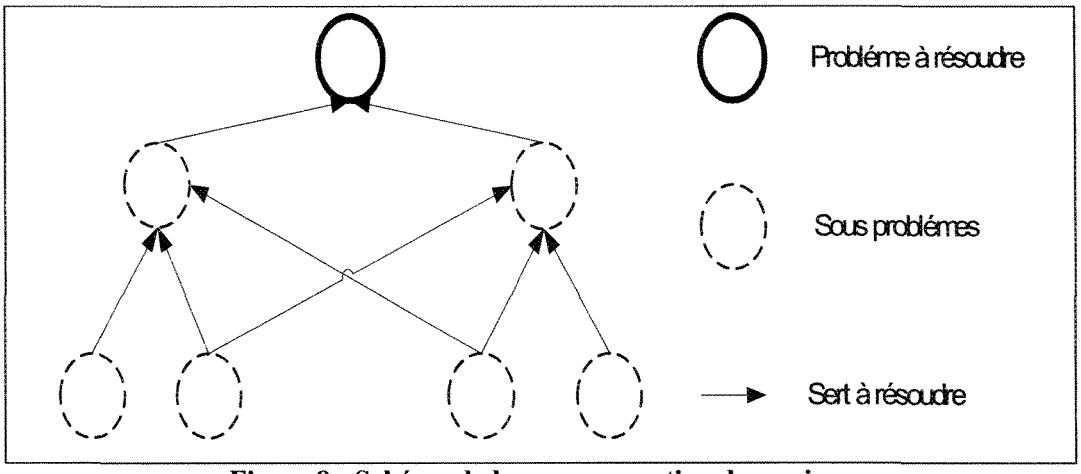

Figure 8 : Schéma de la programmation dynamique

Pour appliquer la programmation dynamique à un problème donné, il suffit de trouver un moyen de le subdiviser en problèmes de plus petites tailles puis une expression récursive permettant de calculer la valeur optimale d'une solution à une étape en fonction des solutions des sous problèmes de l'étape précédente est recherchée.

Après l'obtention de l'expression récursive, une table mémoire est crée dans laquelle chaque case correspond à un sous problème. La table est initialisée par les conditions initiales de l'expression récursive puis elle est remplie en se servant de l'expression. Le dernier calcul de la table ne donne que la valeur optimale de la solution, elle ne donne pas la solution elle-même. Pour reconstruire cette solution, la table est parcourue dans le sens inverse de sa construction.

Comme la méthode de séparation et d'évaluation, la programmation dynamique est une méthode d'énumération. Même si ce type de méthode diminue grandement le nombre de solutions générées pour des problèmes de grandes tailles ou complexes, elles engendrent souvent des complexités exponentielles [24]. Ces méthodes ne sont donc pas toujours exploitables, et seront alors remplacées par des méthodes approchées. Toutefois, dans certains cas, cette approche génère des solutions polynomiales.

1.4.2.2 Méthodes approchées

La majorité des problèmes d'ordonnancement sont NP-difficiles et leurs résolutions par des méthodes exactes ne sont, alors, pas possibles dans un contexte de temps de calcul et de mémoire limité. Dans ce cas, une solution non optimale, mais de bonne qualité et ayant un temps d'exécution raisonnable, appelée solution approchée, est recherchée. Une analyse des résultats devra être entreprise pour évaluer la distance séparant la solution obtenue et la solution optimale.

La relaxation et les heuristiques sont deux grands types de méthodes fournissant des solutions approchées.

1.4.2.2.1 Relaxation

Il est possible pour passer d'un problème difficile à un problème facile de relaxer certaines de ses contraintes. Cette relaxation consiste à simplifier le problème en agissant sur ses contraintes. Pour cela, soit une ou plusieurs contraintes sont éliminées, soit elles sont rendues moins contraignantes. La solution du problème relâché sera alors une « bonne » approximation du problème initial.

Voici une liste non exhaustive des relaxations les plus communes concernant les problèmes d'ordonnancement [29] :

- permettre la préemption de tâches,
- modifier les temps d'exécution des tâches en les rendant unitaires ou appartenant à un certain domaine d'intervalle de temps,
- modifier la précédence des tâches, en la rendant plus simple.

1.4.2.2.2 Heuristiques

Une méthode heuristique est une procédure exploitant au mieux la structure du problème considéré, dans le but de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible [62]. C'est une méthode, souvent simple, guidée généralement par l'intuition et qui fournit relativement rapidement, une bonne solution à des problèmes difficiles. On distingue deux classes d'heuristiques [68] : la première citée génère une solution tandis que la deuxième approche en génère un ensemble.

- l'approche constructive : typiquement ce sont les algorithmes de type glouton ou vorace,
- l'approche améliorative ou itérative : les métaheuristiques.

L'approche constructive :

Cette approche démarre d'une solution vide et insère à chaque étape une composante du problème considéré dans la solution partielle jusqu'à obtention d'une solution complète admissible. Les règles de priorité sont constructives. En effet, elles construisent une solution en appliquant un critère de priorité aux tâches.

L'approche itérative :

Cette approche consiste à construire une solution initiale et de procéder à des améliorations à travers un processus itératif. Les métaheuristiques constituent la partie la plus connue de cette approche. Elles utilisent différentes approches de recherche de solution approchée, qui permettent de les classifier. Quatre de ces approches de recherche sont décrites brièvement dans les paragraphes suivants.

- L'approche par la recherche locale

La recherche locale consiste à faire des mouvements d'une solution vers une autre dans son voisinage selon un certain nombre de lois prédéterminées [67]. En résumé, les méthodes de recherche locales partent d'une solution arbitraire puis vont par transformations de la solution, appelée mouvements, successivement aller vers la solution optimale. Comme le montre la Figure 9, cette approche tend donc à trouver l'optimum global (G) sans tomber dans le piège des optima locaux (L par exemple).

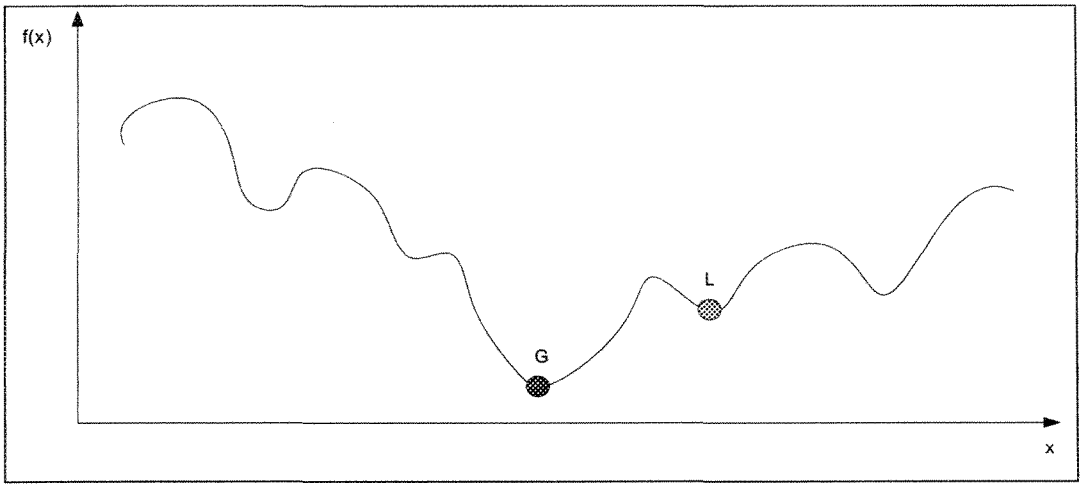

Figure 9 : Optima locaux et global

Voici deux métaheuristiques basées sur la recherche locale.

Le recuit simulé

Cette méthode est basée sur la thermodynamique, elle a été présentée par Métropolis *et al.* [57]. C'est une méthode de recherche locale qui suit le principe suivant. Le système est soumis à une haute température puis refroidit progressivement pour atteindre un état d'équilibre d'énergie minimale. Cet état d'équilibre correspond à la solution optimale. Pour plus de détails, le lecteur peut se référer à l'article de E. W. Eglese [22].

La méthode avec tabous

Crée par Glover [27], elle applique aussi une méthode itérative, mais ajoute une mémoire ou liste tabou qui lui permet de ne pas rester bloqué sur un optimum local. En effet, une information sur les solutions, venant d'être trouvées, est gardée en mémoire, ce qui empêche de rester bloqué sur une solution de moindre qualité ou de revenir à des solutions déjà étudiées.

- L'approche évolutive

Ces méthodes s'inspirent du monde vivant. Elles manipulent un ensemble de solutions appelé population. Le processus est cyclique, composé d'une phase de coopération puis d'une phase d'adaptation individuelle. Les algorithmes génétiques et les colonies de fourmis sont évolutifs.

Les algorithmes génétiques

Ces algorithmes travaillent sur une population d'individus [77]. A chaque individu est assigné une valeur, ou *fitness,* qui représente son bénéfice par rapport au problème posé. L'algorithme fait évoluer la population vers la solution recherchée en croisant les individus entre eux et en effectuant sur certains des mutations. Les mutations permettant de garder une population suffisamment hétérogène.

Les colonies de fourmis

Comme son nom l'indique, cette métaheuristique s'inspire du comportement des fourmis [21]. Elle est basée sur le principe suivant : les fourmis communiquent entre elles grâce à des phéromones qu'elles déposent sur les pistes allant de leurs nids vers la nourriture. Ces phéromones leur permettent de trouver le plus court chemin, car statistiquement il sera le plus emprunté.

- L'approche hybride

Cette approche combine deux méthodes de recherche pour améliorer les solutions. Par exemple, il est possible de mélanger algorithmes génétiques et méthode tabou.

1.5 Conclusion

Dans ce chapitre, nous avons vu qu'un problème d'ordonnancement est la recherche de l'agencement de ressources dans le but de satisfaire des contraintes exprimées sous forme d'une fonction mathématique. Le problème est donc caractérisé par ses ressources ou machines et par ses activités ou tâches. Le passage d'une tâche sur une machine est une opération qui possède une durée ou temps d'exécution. Les contraintes devant être satisfaites sont les limitations dues à l'environnement et la fonction mathématique, appelée fonction objectif, est une équation soit de minimisation ou de maximisation. Une notation compacte des problèmes d'ordonnancement a été introduite, par le triplet $\alpha \mid \beta \mid \gamma$ dénotent pour α l'environnement machine, β les contraintes et enfin γ la fonction objectif. Les solutions de ces problèmes (ordonnancements) sont représentées par un diagramme de Gantt. Ce diagramme matérialise les opérations au cours du temps. Ces définitions permettent de construire un modèle de base, qui peut être étendu pour donner des modèles plus complexes comme les ateliers à cheminement unique ou multiple.

Une fois la modélisation effectuée, il est intéressant d'étudier la complexité du problème. En effet, les problèmes d'ordonnancement font partie des problèmes qui ont la particularité d'avoir des ensembles de solutions trop large pour être énumérés en un temps raisonnable. La théorie de la complexité a été créée pour classifier de tels problèmes. Dans cette théorie les deux principales classes sont la classe P et la classe NP. La classe P contient les problèmes « faciles » ou résolubles grâce à un algorithme en temps polynomial. Et la classe NP contient les problèmes dits « difficiles ». Un problème difficile signifie qu'il existe au moins une instance pour laquelle la recherche de solution génère des temps d'exécution prohibitifs.

Cette théorie n'est pas figée, car il reste à l'étude l'existence de la frontière entre la classe P et NP, donc entre les problèmes faciles et difficiles.

L'étude de la complexité conduit au choix de la méthode de résolution. De fait, si le problème est facile alors un algorithme spécifique est exhibé. Dans le cas d'un problème NP-complet, deux types de méthodes peuvent être appliqués. Les méthodes fournissant une solution optimale telle la programmation dynamique, sont applicables à des problèmes de taille raisonnable. Les méthodes, ne fournissant qu'une solution approchée, telles les métaheuristiques sont appliquées lorsque le problème est de trop grande taille ou trop complexe.

CHAPITRE 2

PRÉSENTATION DE L'ATELIER À CHEMINEMENT MULTIPLE

2.1 Introduction

Dans ce chapitre, nous présentons les problèmes d'atelier à cheminement multiple, communément appelé sous le terme anglais « job shop ». Ces problèmes sont les plus généraux parmi ceux de la famille de l'ordonnancement [37]. Rappelons que dans ce modèle, chaque tâche possède un ordre spécifique d'exécution à travers les machines. Ce type d'atelier fait partie des problèmes d'ordonnancement les plus étudiés [8].

Dans un premier temps, le job shop est présenté de façon générale. Puis nous verrons ses deux modélisations les plus courantes [8] : la modélisation linéaire mixte et le graphe disjonctif. Après avoir exposé les principaux résultats sur la complexité du job shop, nous présentons trois méthodes de résolution proposées dans la littérature. Les deux premières portent sur des job shop particuliers. Le premier est à deux machines dont les tâches possèdent au plus deux opérations, résolu par l'algorithme de Jackson [36]. Le second est constitué de m machines où seules deux tâches sont à exécutés et est résolu par une résolution graphique [2]. La dernière est la recherche avec tabous appliquée au job shop général [20, 63, 85].

Dans un deuxième temps, nous nous intéressons à la prise en compte des opérations de transport dans les problèmes d'ordonnancement. Nous commençons par une présentation générale de cette prise en compte (définition, extension de notation et classification) puis son influence sur les problèmes de job shop. Pour terminer cette revue de littérature, nous présenterons nos objectifs de recherche.

2.2 Problèmes de job shop

Les problèmes de job shop se rencontrent, par exemple, dans l'industrie métallurgique lorsqu'un atelier fabrique des pièces ayant des caractéristiques différentes, mais nécessitant néanmoins des traitements dont les séquences sont connues à l'avance [23].

Du fait de la grande variation autour de ce problème, il n'a pas été possible pour nous de lui trouver dans la littérature une formulation unique. Dans le présent mémoire, une formulation, la plus générale possible, est donc présentée.

2.2.1 Présentation

Il est communément admis que c'est le livre « Industrial Scheduling » de Muth et Thompson [61] qui a été le premier à regrouper tous les résultats autour de ce sujet et a servi de base aux recherches qui ont suivies.

Un job shop est constitué d'un ensemble fini, noté M, de m machines différentes qui doit exécuter un ensemble J, lui aussi fini, de n tâches. Chaque tâche, notée j_i avec $i \in \{1,..., n\}$, est constituée d'une gamme opératoire, aussi appelée suite linéaire, de n_i opérations. Cette séquence ne dépend que de la tâche et peut donc varier d'une tâche à l'autre. Elle correspond à l'ordre de passage prédéterminé sur chaque machine. Ainsi, une opération O_{ij} (i machine, j tâche) de temps d'exécution p_{ij} sera nommée première opération si celle-ci est la première de la séquence formant la tâche. Au total, il y a donc $\sum_{i=1}^{n} n_i$ opérations à exécuter dans l'atelier. Chaque machine, notée M_k avec $k \in \{1,..., m\}$, est spécialisée et ne peut effectuer qu'un seul type d'opération.

Les contraintes intrinsèques au job shop sont les suivantes :

- les machines sont indépendantes les unes des autres ; elles n'utilisent pas d'outils en commun par exemple,
- une machine est disponible pendant tout l'ordonnancement, même les pannes éventuelles sont prises en compte,
- une machine ne peut effectuer qu'une seule opération à la fois,
- la préemption n'est pas permise,
- deux opérations d'une même tâche ne peuvent être exécutées simultanément,
- les tâches sont indépendantes les unes des autres.

Rappelons qu'il est possible qu'une tâche puisse visiter plusieurs fois la même machine [23]. Ce phénomène est appelé « recirculation » ou gamme bouclante. Toutefois, il est fréquent de trouver des formulations plus restrictives [65]. Par exemple, si les gammes bouclantes ne sont pas permises, chaque tâche est alors composée de m opérations et chaque machine doit effectuer n tâches. Le nombre total d'opérations est alors $m*n$. De plus, si $m=n$ alors le problème est dit carré. Une autre restriction possible est que la $j^{ème}$ opération doit être exécutée par la $j^{ème}$ machine, ce qui revient à traiter un problème de flowshop.

L'ordre de passage sur les machines étant fixé, le but est de trouver l'ordonnancement de toutes les opérations sur chaque machine. La plupart des recherches effectuées sur ce type d'atelier ont comme fonction objectif le makespan [37].

2.2.2 Modélisation

Le job shop peut être modélisé de différentes façons, mais les deux plus communes sont la modélisation programmation linéaire mixte et celle par graphe

disjonctif. Notons que d'autres modélisations du job shop ont été proposées. Pour plus de détails, consulter [24].

2.2.2.1 Modélisation linéaire mixte

La modélisation linéaire mixte a été l'une des premières méthodes utilisées dans la littérature [54]. Celle que nous présentons, ci-dessous, est celle décrite dans [1].

Minimiser t_n

Sous les contraintes suivantes:

$$t_i - t_j \geq p_i,\ (i,j) \in A \qquad (1)$$

$$t_i \geq 0,\ i \in N \qquad (2)$$

$$\left(t_j - t_i \geq p_i\right) ou \left(t_i - t_j \geq p_j\right),\ (i,j) \in E_k, k \in M \qquad (3)$$

Rappelons que M est l'ensemble des machines. N est l'ensemble des opérations (incluant deux opérations supplémentaires fictives, les opérations de départ et de fin de l'atelier) et A l'ensemble de paires *(i, j)* d'opérations marquant les relations de précédence de la séquence des machines pour une tâche. E_k est l'ensemble des paires d'opérations devant être exécuté sur la machine k. Finalement, t_i est la date de départ de l'opération.

La contrainte (1) assure que la séquence des opérations de chaque tâche correspond à l'ordre prédéterminé. La contrainte (3) garantit que les machines n'exécutent qu'une tâche à la fois. Et la contrainte (2) assure la terminaison de toutes les tâches. Trouver un ordonnancement respectant ces contraintes est alors une solution réalisable du problème de job shop.

2.2.2.2 Modélisation par graphe disjonctif

Bien que, le diagramme de Gantt soit la méthode traditionnelle pour visualiser un ordonnancement, Blazewicz *et al.* [8] ont montré que la modélisation par graphe disjonctif est maintenant prévalente. Cet outil de modélisation élégant a, pour la première fois, été présenté, en 1964, par Roy et Sussmann [69].

L'atelier est représenté sous la forme d'un graphe $G = (S, C, D)$ où S est l'ensemble des nœuds du graphe. Cet ensemble de nœuds représente l'ensemble des opérations de toutes les tâches, auxquelles sont ajoutées deux nœuds fictifs s et p. Ces deux nœuds représentant respectivement le début et la fin de l'ordonnancement. C est l'ensemble des arcs appelés arcs conjonctifs qui symbolisent les chemins de toutes les tâches. Ces arcs indiquent donc les contraintes de précédence entre les opérations. Ainsi s'il existe un arc allant de l'opération O_{11} à l'opération O_{12} alors l'opération O_{11} est le prédécesseur immédiat de O_{12}. Finalement, D est l'ensemble des arcs disjonctifs marquant les contraintes de ressource. Ces arcs sont, au départ, non dirigés, ils relient les opérations devant passer sur la même machine. Le choix d'une direction assure que deux opérations partageant la même ressource ne sont pas exécutées simultanément. Il est à noter que les arcs de D reliant toutes les opérations partageant une même machine forment une clique de disjonctions (ou sous graphe complet) de doubles arcs. Puisqu'il y a m machines, alors l'ensemble d'arcs disjonctifs forment m cliques. Chaque arc du graphe est pondéré avec le temps d'exécution de l'opération origine de l'arc. Les arcs partant du nœud source s sont étiquetés par un temps nul.

Un ordonnancement réalisable correspond à la sélection d'un arc disjonctif dans chaque paire formant les cliques pour aboutir à un graphe acyclique. La sélection des arcs dans une clique doit donc être acyclique. C'est l'arc sélectionné

qui donne l'ordre de passage sur la machine. La durée totale d'un ordonnancement est obtenue, après la création du graphe acyclique, en recherchant le plus long chemin entre le nœud initial et le nœud final. Ce chemin est alors appelé chemin critique.

Exemple 1: Cette modélisation est illustrée sur un problème $J4||C_{max}$ à trois tâches. Le Tableau 3 donne les temps d'exécutions des opérations et les séquences de machine pour chaque tâche.

Tâches	Séquence machines	Temps d'exécution
1	M1, M2, M3	$p_{11} = 9$, $p_{21} = 8$, $p_{31} = 4$
2	M1, M2, M4	$P_{12} = 5$, $p_{22} = 6$, $p_{42} = 3$
3	M3, M1, M2	$P_{33} = 10$, $p_{13} = 4$, $p_{23} = 9$

Tableau 3 : Données du problème $J4 \mid\mid C_{max}$

La Figure 10, ci-dessous, donne le graphe disjonctif associé à cet atelier.

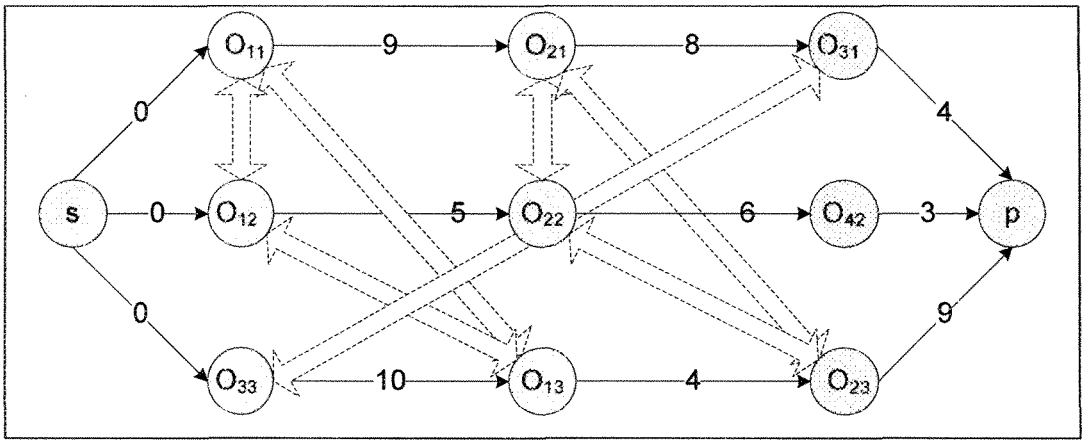

Figure 10 : Graphe disjonctif

Les flèches solides représentent les arcs conjonctifs et les doubles flèches les arcs disjonctifs. Par soucis de lisibilité, la pondération des doubles flèches n'est pas

indiquée. La Figure 11, ci-dessous, trace le graphe après orientation des arcs disjonctifs.

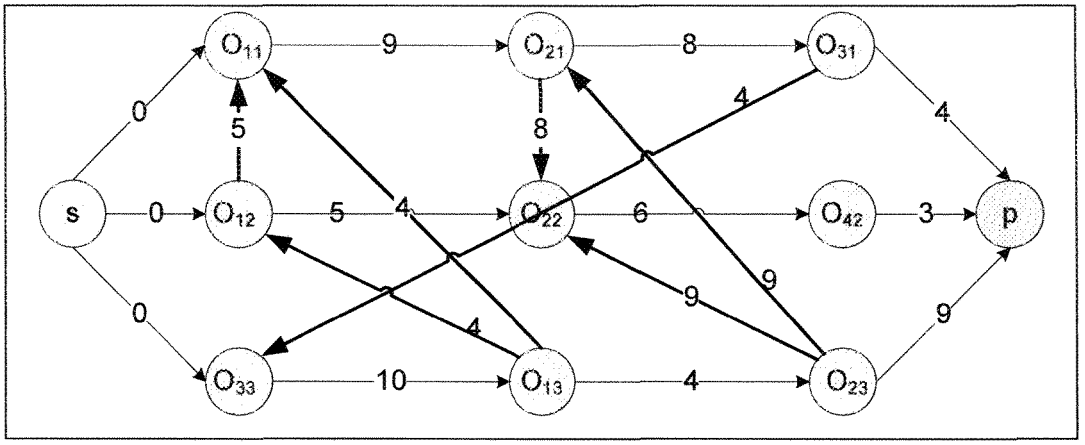

Figure 11 : Graphe après sélection des arcs disjonctifs

Le choix de l'orientation des arcs n'est pas quelconque, car le but est de minimiser le temps d'accomplissement total. Pour cela, il suffit de trouver le chemin le plus long de *s* à *p* du graphe. Dans cet exemple, un tel chemin est : *s*, O_{12}, O_{22}, O_{42}, *p* qui a pour longueur *14* unités de temps.

2.2.3 Complexité

Les problèmes de job shop sont en général NP-complets [26], même si l'atelier est simple. En effet, Lenstra et Rinnooy Kan [51] ont montré que les ateliers possédant plus de trois machines, ou un nombre de tâches supérieur ou égal à trois, sont NP-difficiles même si la préemption est permise. De même, pour les problèmes à deux machines, dès qu'il y a recirculation, ils deviennent fortement NP-difficiles. En revanche, si la fonction objectif est la minimisation du makespan, alors il existe quelques cas particuliers résolubles en temps polynomiaux :

- un atelier composé de deux machines dont chaque tâche comprend au plus deux opérations, Jackson [36],

- un atelier comprenant deux tâches sur *m* machines, Akers [2], Brucker [12],
- un atelier à deux machines dont les tâches ont des opérations unitaires, Kubiak *et al.* [43],
- un atelier à deux machines dont les tâches ont des opérations unitaires avec des dates de disponibilité, Timkovsky [81],
- un atelier à deux machines dont le nombre de tâches est fixe, Brucker [13].

Malheureusement, même de petites modifications à ces ateliers les font basculer dans la classe des problèmes NP-difficiles. Par exemple, l'ajout d'une troisième machine même pour des ateliers ayant des tâches unitaires devient NP-difficile. Le Tableau 4 résume les principales complexités en fonction des caractéristiques de l'atelier.

Type d'atelier	Complexité	Référence
J2 | $n_i \leq 2$ | C_{max}	*O(nlogn)*	Jackson, 1956 [36]
J2 | $p_{ij} = 1$| C_{max}	*O(nlog(nr))*	Kubiak, 1995 [43]
J2 | $p_{ij} = 1$; r_i | C_{max}	$O(n^2)$	Timkovsky, 1997 [81]
J2 | n = k | C_{max}	$O(r^{2k})$	Brucker, 1994 [13]
J2 | $p_{ij} \in \{1,2\}$ | C_{max}	NP-difficile	Lenstra et Rinnooy Kan, 1979 [51]
J2 | chains ; $p_{ij} = 1$| C_{max}	NP-difficile	Timkovsky, 1985 [79]
J2 | prmtn| C_{max}	NP-difficile	Lenstra et Rinnooy Kan, 1979 [51]
J2 | n = 3 ; prmtn| C_{max}	NP-difficile	Brucker *et al.*, 1996 [16]
J3 | $p_{ij} = 1$| C_{max}	NP-difficile	Lenstra et Rinnooy Kan, 1979 [16]
J3 | n = 3| C_{max}	NP-difficile	Sotskov et Shakhlevich, 1995 [76]

Tableau 4 : Récapitulatif des principales complexités d'un job shop

Pour plus de détails sur la complexité des problèmes de job shop, consulter Lenstra *et al.* [50, 51], Gonzalez et Sahni [28] et Sotskov et Shakhlevich [76].

2.2.4 Méthodes de résolution

Dans cette partie, nous n'aborderons pas les méthodes de résolution telles que la méthode de séparation et d'évaluation progressive [14, 45] ou l'heuristique « shifting bottleneck » [1, 4]. Nous nous limiterons à décrire trois méthodes de résolution particulières du problème de job shop. Les deux premières sont les méthodes classiques de l'algorithme de Jackson [36] et de la résolution graphique [2, 12]. La troisième porte sur la recherche avec tabous appliquée au job shop [20, 63, 85]. La présentation de ces seules méthodes s'explique par le fait que ce sont elles dont nous nous servons par la suite.

2.2.4.1 Job shop à deux machines et algorithme de Jackson

Cet algorithme proposé par Jackson [36], repose sur la réduction du problème $J2||C_{max}$ dont les tâches ont au plus deux opérations, au problème de flowshop à deux machines $F2||C_{max}$. Tout d'abord, présentons l'algorithme de Johnson qui résout le problème $F2||C_{max}$.

Algorithme de Johnson :

Le problème $F2||C_{max}$ a été résolu de façon optimale par Johnson [38]. Cet algorithme, et ses variantes, est un moyen rapide d'optimiser l'ordonnancement de processus simples. En effet, il est de facile de voir que la complexité de cet algorithme est en $O(n \log n)$. Les résultats obtenus par Johnson sont devenus des classiques dans la théorie de l'ordonnancement [66].

Description du problème :

Les deux machines du flowshop sont nommées A et B, et doivent exécuter n tâches. Chaque tâche doit tout d'abord passer sur la machine A puis sur la machine B et est caractérisé par un temps d'exécution t_{Aj} sur la machine A et un temps d'exécution t_{Bj} sur B. Le problème est de trouver le bon ordonnancement des tâches pour avoir un temps d'accomplissement minimal. La résolution est basée sur la règle appelée règle de Johnson :

Théorème 2 : Une tâche i précède une tâche j dans une séquence optimale si

$$\min\left\{t_{Ai}, t_{Bj}\right\} \leq \min\left\{t_{Bi}, t_{Aj}\right\}$$

À partir de cette règle, il est facile de générer un algorithme optimal comme suit :

Algorithme de Johnson

1. Tant qu'il reste des tâches non placées dans la séquence,
   1.1. Trouver la tâche pour lequel $\min_j\left\{t_{Aj}, t_{Bj}\right\}$
   1.2. Si le temps d'exécution minimale de la tâche est celui de la machine de départ alors cette tâche est mise au début de la séquence.
   1.3. Si le temps d'exécution minimale de la tâche est celui de la seconde machine alors cette tâche est mise à la fin de la séquence.
2. FinTantQue

Présentons maintenant l'algorithme de Jackson, essentiellement basé sur celui de Johnson.

Algorithme de Jackson :

Cet algorithme s'applique à un job shop à deux machines sur lesquelles doivent être exécutés n tâches, qui ont au plus deux opérations. Tout d'abord, ces

tâches sont scindées en deux ensembles suivant leurs machines de départ. Ainsi, N_{AB} est l'ensemble des tâches dont la première opération doit s'exécuter sur la machine A et N_{BA} l'ensemble des tâches dont la première opération doit s'exécuter sur B. Si le nombre de tâches de N_{AB} est plus grand ou égale que celui de N_{BA} alors l'algorithme de Johnson est appliqué sur l'ensemble N_{AB}. L'ensemble des tâches de N_{BA} est quand à lui ordonnancé de façon quelconque. Si N_{BA} est plus grand que N_{AB}, alors ce cas est symétrique.

Théorème 3 : L'algorithme de Jackson est optimal pour le cas de job shop à deux machines.

Preuve : Comme le montre la Figure 12, supposons que $N_{AB} \geq N_{BA}$. (Le cas $N_{AB} < N_{BA}$ est symétrique). Dans ce cas, la machine A ne s'arrête qu'à la fin de l'exécution des tâches qui lui sont dédiées et cela sans temps morts.

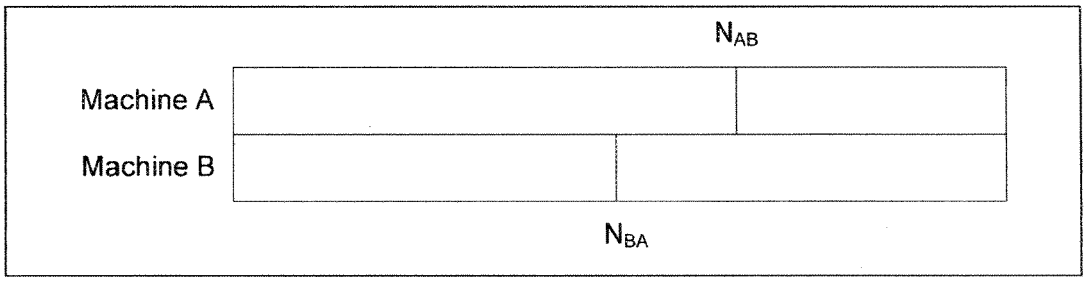

Figure 12 : Etat initial de l'atelier avant l'algorithme de Jackson

En effet, les tâches de N_{BA} ne peuvent commencer leurs exécutions sur la machine A que lorsque les tâches de N_{AB} sont terminées sur celle-ci. Autrement dit, la machine A s'arrête au temps $c_1 = \sum_{i=1}^{n} p_{Aj_i}$, qui est une borne inférieure.

Les tâches de N_{AB} sont ordonnancées suivant la règle de Johnson. Leur temps d'accomplissement c_2 est donc optimal. Dans tous les cas, le makespan de

l'ensemble est clairement le maximum entre c_1 et c_2. Par conséquent, l'algorithme de Jackson est optimal.

Exemple 2: Prenons un atelier $J2||C_{max}$ de neuf tâches ayant chacune au plus deux opérations. Le Tableau 5 résume les temps d'exécution des tâches sur les machines A et B.

Tâches j	j_1	j_2	j_3	j_4	j_5	j_6	j_7	j_8	j_9
Temps d'exécution t_{Aj}	3	5	1	6	7	2	-	3	7
Temps d'exécution t_{Bj}	6	2	2	6	5	4	5	4	2

Tableau 5 : Temps d'exécution sur les machines *A* et *B*

Les tâches sont réparties comme suit sur les deux machines, $N_{AB} = \{j_1, j_2, j_3, j_4, j_5\}$ et $N_{BA} = \{j_6, j_7, j_8, j_9\}$. La tâche j_7 n'est constituée que d'une seule opération qui s'exécute sur la machine B. Appliquons maintenant l'algorithme de Jackson. C'est l'ensemble N_{AB} qui est plus grand que N_{BA}, la séquence R_{AB} est donc créée en appliquant l'algorithme de Johnson sur N_{AB}. Son exécution est montrée dans le Tableau 6, ci-dessous.

Liste des tâches à ordonnées	Temps d'exécution le plus faible	Ordonnancement
j_1, j_2, j_3, j_4, j_5	t_{A3}	j_3 _ _ _ _
j_1, j_2, j_4, j_5	t_{B2}	j_3 _ _ _ j_2
j_1, j_4, j_5	t_{A1}	$j_3\, j_1$ _ _ j_2
j_4, j_5	t_{B5}	$j_3\, j_1$ _ $j_5\, j_2$
j_4		$j_3\, j_1\, j_4\, j_5\, j_2$

Tableau 6 : Exécution de l'algorithme de Johnson sur N_{AB}

L'ordonnancement optimal est donc $R_{AB} : j_3, j_1, j_4, j_5, j_2$. Il reste à appliquer à ordonnancer de façon quelconque l'ensemble N_{BA}. Si par exemple, on laisse les

tâches dans l'ordre alphanumérique, on obtient la séquence suivante, $R_{BA} : j_6, j_7, j_8, j_9$.

L'ordonnancement final sur les deux machines est alors le suivant. Sur A, $j_3, j_1, j_4, j_5, j_2, j_6, j_8, j_9$ et sur B, $j_6, j_7, j_8, j_9, j_3, j_1, j_4, j_5, j_2$.

2.2.4.2 Job shop à deux tâches, résolution graphique

Le job shop, considéré dans ce travail, possède n machines, mais ne doit exécuter que deux tâches. Akers [2] a proposé une méthode de résolution graphique permettant la résolution d'un tel atelier. Cette méthode a été reprise par Brucker [12] pour créer un algorithme efficace. Nous décrirons, tout d'abord, la transformation en représentation graphique du problème à deux tâches puis sa résolution.

Supposons que les deux tâches j_i, avec $i = \{1, 2\}$, devant être exécutées par l'atelier, sont composés de n_i opérations. Ces deux tâches ont chacune un temps d'accomplissement total noté C_1 et C_2 respectivement. Le but est alors de minimiser le temps d'accomplissement total de l'atelier, composé de ces deux temps, noté $f(C_1, C_2)$.

Ce problème est alors représenté sous forme d'un plan à deux dimensions où chaque tâche est représentée par un axe fractionné en n_i intervalles ordonnés suivant une séquence déterminée. Chaque intervalle correspond à une opération O_{ij} et sa longueur correspond à son temps d'exécution p_{ij}. Deux intervalles O_{mj_1} et O_{mj_2} forment un « obstacle » si ces deux opérations partagent la même ressource, c'est-à-dire la même machine durant une même période. De plus, la dernière intersection des tâches est appelée « obstacle final » F, ou encore $F = \left(\sum_{k=1}^{n_1} p_{1k}, \sum_{k=1}^{n_2} p_{2k} \right)$.

Par exemple, un atelier a trois machines, nommées *A*, *B* et *C*, doit exécuter deux tâches j_1 et j_2 possédant chacune trois opérations ayant les caractéristiques données par le Tableau 7 suivant.

Opérations	Machines	Temps d'exécution
O_{A1}	A	3
O_{B1}	B	2
O_{C1}	C	4
O_{C2}	C	2
O_{A2}	A	1
O_{B2}	B	2

Tableau 7 : Données de l'exemple.

Le graphique de la Figure 13 suivante, est alors obtenu.

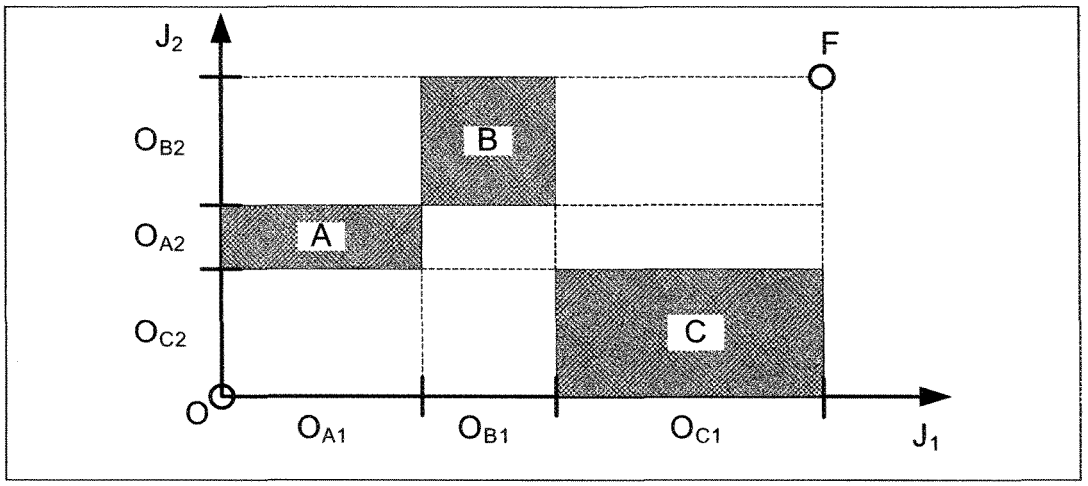

Figure 13 : Représentation graphique de j_1 et j_2

Une solution réalisable du problème correspond au chemin partant de l'origine *O* et allant jusqu'au point final *F*. Selon Akers, ce chemin est composé des segments horizontaux (seulement si une opération de j_1 est exécutée), des segments verticaux (seulement si une opération de j_2 est exécutée) et de segments diagonaux

lorsque les opérations de j_1 et j_2 sont exécutées ensemble. Le chemin doit éviter l'intérieur des obstacles, car les opérations correspondantes ne peuvent être exécutées simultanément sur la même machine. Il est à noter que la longueur d'un segment diagonale est égale à la longueur de ses projections sur les deux axes, correspondant au temps passé pour l'exécution simultanée des deux opérations.

Le makespan est alors la longueur du chemin et sera calculé ainsi :

$$C_{max} = \sum(\textit{longueurs des segments verticaux}) + \sum(\textit{longueurs des segments horizontaux}) + \frac{1}{\sqrt{2}}\sum(\textit{longueurs des segments diagonaux}).$$

Pour trouver le plus court chemin parmi toutes les solutions réalisables, au lieu de les passer toutes en revue, une solution est de transformer le problème du plus court chemin dans le plan en la recherche du plus court chemin dans un graphe acyclique *N*. Cette méthode a été décrite par Brucker [12]. Tout d'abord, le graphe acyclique $N = (V, E, d)$ est créé, où l'ensemble des nœuds V est composé de l'origine O de l'obstacle final F et de certains coins Nord-Ouest et Sud-Est des obstacles. Les coins sont sélectionnés en recherchant les successeurs des nœuds déjà trouvés de V.

Comme le montre la Figure 14, ci-dessous, chaque nœud v_i de V a au plus deux successeurs obtenus en traçant une diagonale imaginaire jusqu'à ce qu'elle touche un obstacle, les successeurs seront alors les coins Nord-Ouest et Sud-Est de cet obstacle. Si la diagonale ne touche pas d'obstacle et atteint une des limites du plan alors le nœud F sera l'unique successeur de v_i.

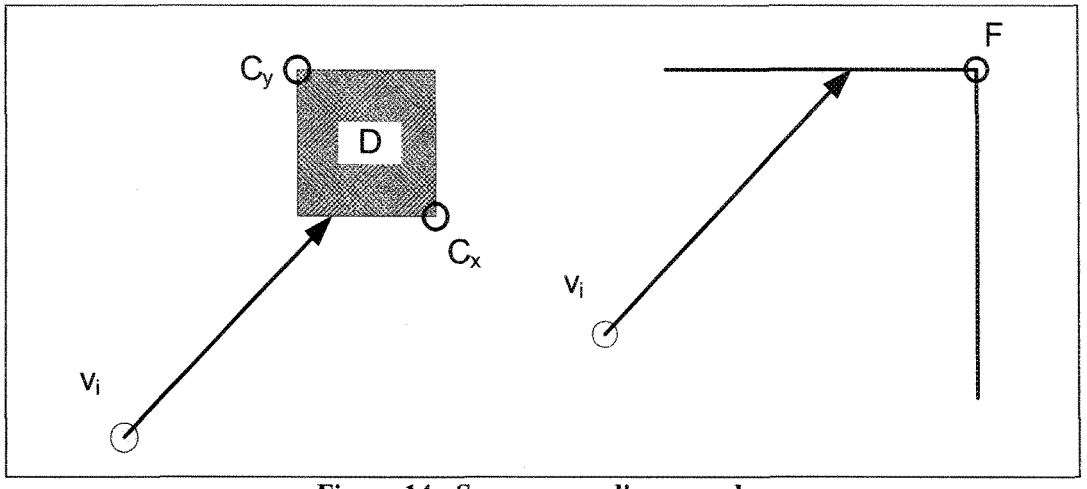

Figure 14 : Successeurs d'un noeud v_i

Brucker [12] a montré que le graphe N peut être construit en $O(r \log r)$ avec r le nombre d'obstacles se trouvant dans le graphe. De plus, le chemin le plus court dans le graphe acyclique est obtenu en $O(r)$. Le problème de job shop à deux tâches peut donc être résolu en $O(r \log r)$.

2.2.4.3 Recherche avec tabous

La recherche avec tabous est l'une des principales métaheuristiques appliquées à la résolution du problème de job shop [8]. Pour ce faire, le problème de job shop est modélisé, dans la majorité des cas, à l'aide du graphe disjonctif [69].

L'idée d'un algorithme basé sur la recherche avec tabous est comme suit. Initialement, l'algorithme se dote d'une solution (qui n'est pas obligatoirement réalisable). Cette solution initiale est généralement construite, à partir d'une heuristique d'insertion [63], d'une règle de priorité [20], ou d'un ordonnancement semi-actif [85] .

Une fois cette solution initiale construite, une structure de voisinage est alors créée. Cette structure consiste à déterminer la fonction par laquelle une nouvelle solution est créée à partir d'une solution existante. Cette transformation, d'une solution à une autre, se nomme mouvement. Dans le cadre d'une modélisation par graphe disjonctif, la structure de voisinage est créée par réarrangement du chemin critique. Un réarrangement consiste à déplacer les opérations du chemin critique. Par exemple, Dell'Amico *et al.* [20] construisent deux voisinages par permutation d'opérations du chemin critique. Si les solutions sont modélisées par des ordonnancements semi-actifs, comme c'est le cas de Widmer [85], un mouvement peut consister alors à modifier la position d'une opération.

La caractéristique fondamentale de la recherche avec tabous est l'utilisation d'une mémoire. Cette mémoire assure de ne pas rester bloqué(e) dans un minimum local. Pour éviter de revenir sur des solutions et guider vers des régions non explorées, des informations sur la solution courante sont stockées dans une liste nommée liste taboue. Généralement, les informations stockées sont l'opposé du ou des mouvements qui ont conduit à la solution. Par exemple, pour un ordonnancement semi-actif, le couple (opération déplacée, position avant déplacement) est mis dans la liste taboue [85]. Parfois, les éléments contenus dans cette liste n'ont pas la même structure [20]. De plus, la liste taboue est la plupart du temps gérée selon le mode FIFO (premier entré, premier sorti) où lorsqu'un nouvel élément tabou est inséré dans la liste, l'élément le plus ancien est retiré. En général, la taille de la liste taboue est fixe ; de taille 5 ou 7. Toutefois, il peut se trouver des applications où cette taille est modifiée au cours de la recherche suivant un ou des critères prédéterminés.

L'emploi de listes taboues peut être limitatif et restreindre ainsi l'espace d'exploration. Pour pallier cela, un critère d'aspiration y est introduit. Ce critère

permet de révoquer le statut tabou d'un mouvement pour permettre une recherche dans une région prometteuse. Un profit est alors évalué si le mouvement est choisi. Si ce profit est acceptable alors le mouvement est retiré de la liste taboue et la nouvelle solution est créée. Le critère d'aspiration le plus répandu est celui de révoquer le tabou d'un mouvement si celui-ci conduit à une solution qui a une valeur inférieure à celle de la meilleure solution trouvée avant l'itération courante.

Pour mettre en œuvre un algorithme de recherche avec tabous, un critère d'arrêt est déterminé, qui permet à l'algorithme de terminer. En général, ce critère est un mélange des conditions suivantes :

- dès que la recherche atteint une valeur du makespan prédéterminé (cela par exemple peut être la solution optimale quand elle est connue),
- un certain nombre d'itérations ou de temps de calcul,
- il n'y a plus d'amélioration de la solution depuis un certain nombre d'itération qui reste à déterminer.

Il a été rapporté dans [9] que l'implémentation de l'algorithme de recherche avec tabous la plus efficace pour le problème de job shop est celle de Nowicki et Smutnicki [63].

2.3 Utilisation de convoyeurs dans un job shop.

Les problèmes de job shop présentés jusqu'à maintenant considéraient le déplacement des tâches d'une machine à une autre comme instantané. Ces déplacements n'étaient donc pas pris en compte lors de la construction des ordonnancements. Or, cette hypothèse est souvent non justifiée en pratique [44]. En effet, le transport des tâches doit être intégré aux données du problème. Comme le dit Lee *et al.* [49], "La nouvelle tendance dans la théorie de l'ordonnancement est d'étendre les résultats des algorithmes classiques à des modèles plus proches des

problèmes réels. Même si beaucoup de résultats ne sont pas immédiatement applicables, ces nouveaux modèles sont au moins motivés par les problèmes industriels et ont un grand potentiel d'applications". Les chercheurs se sont donc intéressés à l'étude des ateliers prenant en compte le transport des tâches. L'industrie a, de plus, une forte demande pour l'obtention de résultats applicables, car comme Tomkins et White [82] l'ont montré, la manipulation matérielle peut prendre jusqu'à 80% du coût total de fabrication.

2.3.1 Définition d'une opération de transport et notation

L'ordonnancement de machines ayant des opérations de manipulation matérielle diffère de l'ordonnancement classique par le fait que deux types de ressources sont pris en compte : les machines et les transporteurs qui manipulent les matériels. Par la suite, le terme convoyeur sera utilisé pour désigner le moyen de transport des tâches. Les systèmes intégrant un dispositif de manipulation de matériels et de machine contrôlée de façon numérique, pouvant exécuter une grande variété de tâches, sont nommés systèmes flexibles de production (FMS pour Flexible Manufacturing Systems).

Dans de tels systèmes, le transport d'une tâche peut être décrit comme le chargement sur le matériel de transport, du transport entre deux machines et du déchargement de la tâche. Il est donc possible de, soit considérer l'opération de transport comme englobant toutes ses actions, soit de les considérer séparément, chacune devenant une opération distincte. De plus, les temps de transport associés à ces opérations peuvent être dépendants ou non des tâches qui sont transportées.

La prise en compte de la manipulation matérielle amène à étendre la notation de $\alpha|\beta|\gamma$ de Graham *et al.* [30] vu dans le Chapitre 1. La notation est étendue à $\alpha(K)|\beta|\gamma$ où K dénote le nombre de convoyeurs du système. Aux contraintes

« classiques » est ajoutée celle du nombre de types de tâches qui seront exécutées par l'atelier. Cette variété de tâches est notée $\beta = J$. Par exemple, $J2(1)|J>1|C_{max}$ dénote un job shop à deux machines et un convoyeur, devant effectuer au moins deux sortes de tâches et dont la fonction objectif est la minimisation du makespan.

Une autre extension de la notation est proposée par Lee et Chen [48]. Celle-ci ajoute un T à l'environnement machine. Et les contraintes $v = x$ et $c = y$ sont ajoutés à l'ensemble de contraintes. $V = x$ indique le nombre de convoyeurs et $c = y$ la capacité des convoyeurs. Par exemple, $TJ2|v = 2, c = 3|C_{max}$ représente un job shop à deux convoyeurs capables de transporter trois tâches à la fois et dont l'objectif est la minimisation du makespan.

2.3.2 Classification

Dans cette section, les problèmes de job shop sont placés dans la classification générale des problèmes d'ordonnancement.

Dans la plupart des systèmes manufacturiers, les tâches semi-finies (c'est-à-dire ayant déjà été exécutées au moins par une machine, mais dont toutes opérations ne sont pas terminées) sont transportées par des convoyeurs entre les machines. Ensuite, une fois finies, ces tâches sont délivrées aux clients par des moyens de transport (camions). Ce fonctionnement « classique » a amené Lee et Chen [48] à considérer deux types de transport selon l'état de finition des tâches. Le premier type correspond aux transports des tâches semi-finies à l'intérieur des ateliers et le second aux transports des tâches finies entre les ateliers et les clients. Dans le cadre de ce mémoire, seul le premier type sera considéré.

Dans le cas des transports des tâches semi-finies, il est encore possible de subdiviser les ateliers suivant la structure de leurs contraintes de transport. Lee *et al.* divisent les ateliers en trois catégories :

- les cellules robotiques,
- les ateliers possédant un ou plusieurs véhicules guidés et automatisés appelés AVG,
- les ateliers possédant une ou plusieurs grues cycliques sujets à des contraintes de temps.

La Figure 15 montre les différentes subdivisions.

Un atelier à cellules robotiques possède une ou plusieurs cellules constituées de machines. Chaque cellule est chargée d'exécuter une partie de la charge de tâche de l'atelier. Entre les cellules, un ou plusieurs convoyeurs sont chargés de transporter les tâches. Généralement les cellules sont des flowshop ou plus rarement un ensemble de machines parallèles. Typiquement, les cellules robotiques se trouvent dans les usines de semi-conducteurs ou de textiles. Pour une vue d'ensemble de ce type d'atelier, le lecteur peut se référer à l'article de Dawande *et al.* [18].

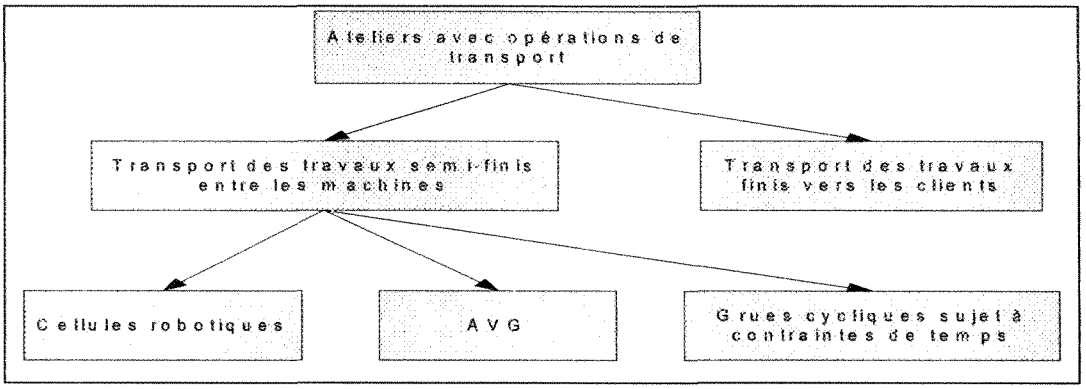

Figure 15 : Classification des problèmes de transport

Les véhicules guidés et automatisés sont des systèmes de transport de tâches sans conducteur. Ils sont apparus pour la première fois dans la littérature en 1983 [59]. Ces véhicules se trouvent souvent dans les flowshop et permettent à ceux-ci de réaliser une grande diversité de tâches souvent en petits lots. Les véhicules circulent sur un réseau guidé. Cela implique que la politique d'ordonnancement doit prendre en compte le trafic ou les collisions possibles. Par exemple, dans la grande distribution, ces véhicules transportent les différents produits à l'intérieur des entrepôts. Les articles de Vis [83] et Le-anh et Koster [47] fournissent une bonne vision de ce type d'ateliers.

La dernière catégorie est celle des ateliers possédant une ou plusieurs grues cycliques, sujet à des contraintes de temps peuvent être vues comme des cas particuliers *de* $J_m(K)|J>1|\pi_{\min}$, avec des contraintes de fenêtres de temps et où les tâches ne peuvent attendre entre les machines de peur d'être dégradées. Ces contraintes, bien que restrictives, décrivent des ateliers que l'on trouve le plus souvent dans les industries chimiques et de galvanoplasties. En effet, la galvanoplastie consiste à traiter chimiquement une pièce en l'immergeant dans une solution dans le but d'y déposer une couche de métal. Les pièces déplacées par les grues ne doivent donc pas rester ni trop peu, ni trop longtemps dans la solution.

Ces trois catégories sont, elles-mêmes, subdivisées en fonction de la taille des zones de stockage de l'atelier. Les zones de stockage sont les endroits où les tâches sont entreposées dans l'attente d'être exécutés par une machine ou d'être emmenés par un convoyeur. Ils peuvent être de deux types, soit d'entrée, soit de sortie. Les stocks d'entrées sont les endroits où les tâches sont placées dans l'attente de leurs exécutions sur une machine. Les stocks de sorties, eux, reçoivent les tâches venant d'être exécutées. Si celles-ci sont semi-finies, elles se trouvent dans une période d'attente d'un convoyeur pour continuer leurs chemins, ou elles sont terminées. Un

stock, d'entrée ou de sortie, peut être réservé à une machine particulière ou servir à plusieurs machines. Si de tels espaces n'existent pas ou sont de tailles limitées alors les machines peuvent avoir à attendre une tâche ou bien devoir garder une tâche terminée en l'absence de convoyeur. Dans ces conditions, ces machines sont considérées bloquées. De même, les convoyeurs peuvent se retrouver bloqués. Par exemple, les cas de blocage se produisent lorsque le convoyeur se trouve sur une machine qui n'a pas fini l'exécution de sa tâche. Ou si l'espace de stockage de la machine sur laquelle il arrive est plein, ce qui l'empêche alors de déposer la tâche qu'il transporte. La taille de l'entreposage peut aussi poser problème si l'atelier n'accepte pas les temps d'attente. Si le lecteur souhaite une vue d'ensemble des problèmes d'atelier avec blocages ou dont une des contraintes est la non-attente, il peut se référer à Hall et Sriskandarajah [32].

2.3.3 Problèmes de job shop avec contraintes de transport

L'introduction des contraintes de transport dans un job shop implique que les convoyeurs doivent être pris en compte dans l'ordonnancement de l'atelier. Deux choix sont possibles : soient d'une part les tâches sont ordonnancées puis les convoyeurs sont agencés en fonction de l'ordonnancement des tâches [33]. Un ordonnancement à deux niveaux de l'atelier est donc effectué. Soit, une approche globale est effectuée où les tâches et les convoyeurs sont ordonnancés en même temps. Du fait de la forte complexité des problèmes de job shop, il n'existe que peu de littérature dans laquelle le transport des tâches est pris en compte. Les problèmes d'ateliers à grues cycliques ne seront pas traités ici, car leurs contraintes sont trop restrictives. Dans ce mémoire, nous limiterons notre étude à des job shop ne possédant qu'un seul convoyeur.

Concernant le problème à un seul convoyeur, malgré une maigre littérature, il est à noter les articles récents de Brucker et Knust proposant des bornes inférieures [15], celui de Bécart *et al.* [5] résolvant une modélisation linéaire mixte et ceux de Hurink et Knust appliquant une recherche avec tabou à ce type d'atelier [34].

Hunrik et Knust [34] résolvent le problème de job shop à m machines possédant des espaces de stockage de taille illimitée et un seul convoyeur suivant deux approches. La première considère une approche globale des machines et du convoyeur. La seconde traite les machines et le convoyeur séparément. Le convoyeur étant considéré comme une machine à goulot d'étranglement, car devant exécuter plus de tâches que les autres machines. La présence de transport à vide pour le convoyeur est alors regardée comme des temps de mises en course qui dépendent de la séquence de tâches. Les deux approches permettent l'application d'une recherche avec tabou sur un graphe disjonctif modélisant les approches.

Les bornes inférieures proposées par Brucker et Knust [15] sont basées sur la seconde approche proposée par Hunrik et Knust [34]. L'article de Bécart *et al.* [5] modélise les ateliers à un seul convoyeur sous forme d'un programme linéaire mixte qui est résolu grâce à la méthode de séparation et d'évaluation progressive.

2.4 Objectifs de la recherche

On observe que le domaine de l'ordonnancement, particulièrement celui du job shop, présente un intérêt scientifique. De plus, l'ajout de convoyeur dans ce type d'atelier rend son étude plus réaliste d'un point de vue industriel. La limitation à deux machines nous a paru intéressante car elle est bien connue lorsqu'elle ne comprend pas de convoyeur. Comme il a été mentionné dans l'introduction, notre volonté était au départ de trouver des algorithmes efficaces. Or, comme nous le montrerons dans le chapitre suivant l'atelier étudié est NP-difficile. Pour ces raisons,

l'objectif de ce travail de recherche a donc été de trouver des méthodes de résolution approchées. Plus précisément, nos objectifs sont :

1. Proposer et utiliser des règles de priorité (dont la règle de Jackson) et évaluer les impacts de la qualité des solutions lorsqu'on ajoute un convoyeur dans la configuration d'un atelier à deux machines.

2. Appliquer une méthode de recherche avec tabous générique à notre atelier et évaluer les solutions engendrées.

Pour réaliser ces objectifs nous étudierons d'abord notre atelier d'une façon générale ce qui nous permettra de tirer des propriétés utiles par la suite. Nous établirons également une borne inférieure permettant de comparer les solutions. Puis grâce à différents articles, nous mettrons en place des règles de priorité et une méthode de recherche avec tabous. Des simulations nous permettrons d'évaluer les solutions engendrées.

2.5 Conclusion

Ce chapitre nous a permis de présenter les problèmes de job shop qui sont des ateliers constitués d'un ensemble fini de m machines différentes qui doivent exécuter un ensemble J, lui aussi fini, de n tâches. Dans ces ateliers, les machines sont indépendantes, disponibles et elles ne peuvent effectuer qu'une opération à la fois. De plus, la préemption n'est pas permise.

Les deux modélisations principales du job shop sont la modélisation linéaire mixte et le graphe disjonctif. La modélisation linéaire mixte décrit le problème sous forme d'équations traduisant la fonction objectif et les contraintes. Le graphe disjonctif est une modélisation où les nœuds représentent les opérations de toutes les tâches. Ces nœuds sont reliés par des arcs conjonctifs représentant le chemin des

tâches et des arcs disjonctifs à orienter représentant les contraintes de ressources. L'orientation des arcs disjonctifs doit produire un graphe acyclique. Si la fonction objectif est la minimisation du makespan alors ce temps sera donné par la recherche du chemin le plus long chemin entre le nœud initial et le nœud final dans le graphe acyclique.

Dans la plupart des cas, les problèmes de job shop sont NP-difficiles. Néanmoins, pour certains problèmes particuliers, dont la fonction objectif est la minimisation du makespan, des algorithmes efficaces existent. Parmi ceux-ci, nous avons présenté le problème à deux machines dont les tâches possèdent au plus deux opérations, résolu par l'algorithme de Jackson [36] qui est essentiellement basé sur l'algorithme de Johnson [39]. Le deuxième cas présenté est la résolution graphique de Akers [2], Brucker [12] qui résout le job shop à m machines ne devant exécuter qu'au plus deux tâches. Cette résolution crée un plan constitué par les opérations des deux tâches et dans lequel se trouvent des obstacles représentant les contraintes de ressources. Le temps d'accomplissement étant alors obtenu par la recherche du plus court chemin dans le graphe constitué de certains des coins des obstacles. Pour finir, nous avons présenté d'une manière générale la recherche avec tabous appliquée au job shop.

Dans la plupart de ces ateliers, le transport des tâches est considéré comme instantané et n'est alors pas pris en compte. Pour faciliter l'application des recherches à des ateliers réels, les opérations de transport des tâches ont donc été introduites. Il a fallu étendre la notation des problèmes telle que présenter dans le Chapitre 1, Section 1.2.4 . Deux extensions existent : l'une est proposée par Pinedo [66] et l'autre par Lee et Chen [48]. L'introduction des opérations de transport a conduit à classifier les différents ateliers suivant la finition des tâches, le type de convoyeur employé et finalement la taille des espaces de stockage des ateliers.

Concernant plus particulièrement les job shop, il n'existe que peu de littérature qui prend en compte les transports. Notons, tout de même, les articles de Brucker et Knust [15], Bécart *et al.* [5] ainsi que celui de Hunrik et Knust [34] qui apportent respectivement des bornes inférieures, une modélisation linéaire mixte et une recherche taboue aux problèmes de job shop avec un seul convoyeur.

Après cette vue d'ensemble du problème de job shop, restreignons notre étude au problème du job shop à deux machines et un seul convoyeur.

CHAPITRE 3

PRÉSENTATION DU JOB SHOP

À DEUX MACHINES ET UN CONVOYEUR

3.1 Introduction

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 [48] « 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 ses 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.* [42] explique cette réduction par le fait que les petits systèmes peuvent être construits et améliorés rapidement et cela, de façon économique. De plus, leurs maintenances et gestions se trouvent ainsi facilitées, tant au niveau matériel que logiciel. Notre étude portera donc sur un problème d'atelier à deux machines seulement, avec des espaces de stockage illimités et un seul convoyeur, dont l'objectif est de minimiser le makespan. Notons qu'il est facile de concevoir, à partir de cette situation d'autres modèles, différents du nôtre. Par exemple, avec des espaces de stockage de taille limitée. De plus, comme nous le montrons par la suite, il s'avère que même avec ce modèle restreint et simplifié, le problème est NP-difficile et qu'il n'a, à notre connaissance, pas été étudié dans la littérature. Tout cela a motivé notre étude.

Cette étude se fera en deux volets. Tout d'abord, nous décrivons précisément l'atelier à deux machines et un convoyeur : ses données, ses contraintes et son fonctionnement. Puis, nous présentons quelques propriétés remarquables de l'atelier. Ceci nous amènera à étudier spécifiquement le convoyeur et notamment sa position initiale, ses comportements possibles et leurs influences sur le makespan. Finalement, nous terminerons cette présentation par la proposition de bornes inférieures pour le problème à deux machines et un convoyeur.

3.2 Système flexible à deux machines et un convoyeur

Un système représente les organes de production et en particulier les systèmes flexibles qui ont la particularité d'être adaptables et réactifs. Ainsi, notre étude porte sur un système flexible de production particulier, les problèmes de job shop avec transport. Un système flexible est un vocable qui recouvre de nombreux systèmes, de la cellule manufacturière flexible (FMC) aux lignes de transfert flexible. Une cellule manufacturière flexible étant un ordinateur contrôlant un ensemble de machines et un système de manipulation automatisé. Quant à la ligne de transfert flexible, elle peut être vue comme un ensemble de machines reliées entre elles par un système de transport des tâches. Pour plus de précisions, nous invitons le lecteur à consulter les travaux de Gultekin *et al.* [31].

3.2.1 Présentation du problème

Le problème de job shop à deux machines et un convoyeur dont le but est de minimiser le makespan est, comme nous l'avons au cours du Chapitre 2, noté $J2(1)|J>1|C_{max}$, suivant la notation de Lee *et al.* [49]. Plus précisément, ce job shop est constitué de deux stations identiques nommées station A et station B respectivement.

Une station, comme le montre la Figure 16, est composée d'une machine et de deux stocks, nommés stock d'entrée et stock de sortie. La machine exécute en premier lieu, les tâches se trouvant sur son stock d'entrée. Une fois l'exécution de la tâche achevée, celle-ci est placée sur le stock de sortie. Comme il a déjà été mentionné, un stock est l'endroit où les tâches en attente sont laissées temporairement en dépôt. De plus, il est à noter les stocks fonctionnent comme des listes FIFO (premier arrivé, premier sorti).

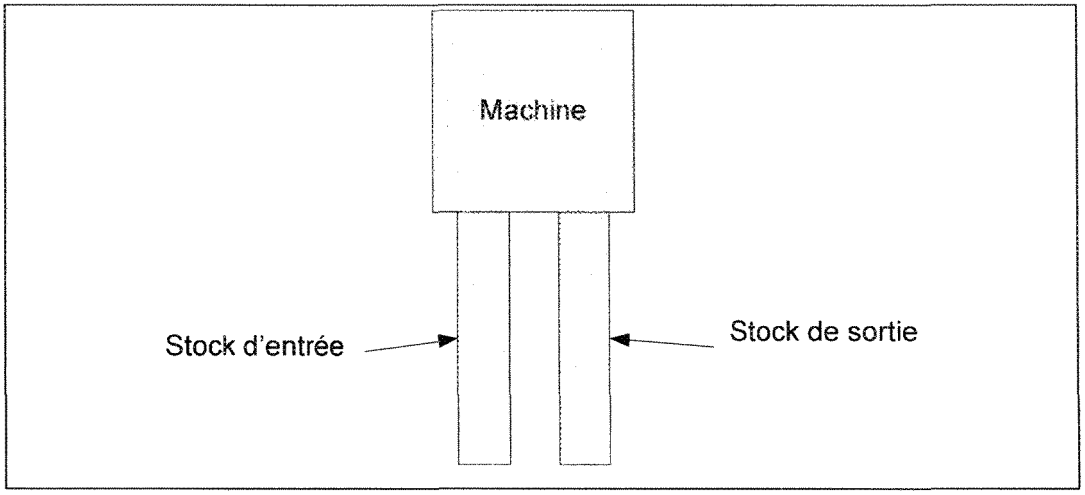

Figure 16 : Une station

L'atelier doit exécuter un ensemble $N = \{1, 2, ..., n\}$ de n tâches. Chaque tâche $j \in N$, consiste en deux opérations, notées de la façon suivante : O_{ij} tel que $\forall j \in N$ et $i \in \{A, B\}$. O_{Aj} et O_{Bj} devant être réalisées sur A et B respectivement, avec des temps d'exécution égaux à t_{Aj} et t_{Bj}. Un temps d'exécution t_{ij} est, alors, la somme de temps mis par une tâche pour passer du stock d'entrée à la machine, noté t_E, à son exécution sur la machine, noté t_M et enfin son placement sur le stock de sortie, noté t_S. Cela peut se résumer, par l'équation suivante :

$$\forall j \in N, \forall i \in \{A, B\}, t_{ij} = t_{Eij} + t_{Mij} + t_{Sij} \text{ tel que } \exists (t_{Eij}, t_{Mij}, t_{sij}, t_{ij}) \in \mathbb{N}.$$

Ceci implique qu'une tâche ayant un temps d'exécution nul sur l'une des deux machines, n'aura donc pas besoin d'être transporté d'une machine à l'autre, et sera considérée comme terminée à la fin de son exécution sur la machine où, son temps d'exécution est non nul. Il faut préciser que, dans le reste du présent mémoire, le temps d'exécution d'une tâche sera considéré comme une entité non séparable et dont la valeur sera entière. De plus, si une tâche j a deux temps d'exécution

identiques alors ce temps sera noté simplement t_j. Et t_i dénote un temps d'exécution identique sur une station i pour toutes les tâches.

Une tâche qui est à la fin de sa route d'exécution, et qui est passée sur au moins une machine et qui est terminée, elle sera alors qualifiée de « tâche réalisée ». Il est à noté qu'une tâche réalisée ne reste pas dans le stock de sortie, elle est considérée comme sortie du système, car terminée et cela n'ajoute donc, pas de temps supplémentaire à son temps d'accomplissement.

Pour rester dans un cadre applicable à un atelier réel, les contraintes usuelles à ce type de problème sont reprises. Une tâche est au plus exécutée sur une machine à un instant donné et une machine n'exécute pas plus d'une tâche à la fois. De plus, la préemption n'est pas permise. Quant aux temps de transport, ils surviennent lorsqu'une tâche, semi-finie, se trouve sur le stock de sortie de la station qui l'a exécuté, et que son itinéraire d'exécution indique, son passage sur l'autre machine. Un temps de transport correspond au temps que met le convoyeur, pour transporter une tâche j d'une station à l'autre. Plus précisément, ce temps de transport est la somme du temps de chargement de j sur le convoyeur, de son temps de transport et enfin du temps de déchargement de j sur le stock de la station d'arrivée. De façon similaire aux temps d'exécution, les temps de transport sont considérés globalement, ils ne seront donc pas séparés en trois temps différents. Un temps de transport est noté τ_{ODj} avec $\forall j \in N$, $\forall (O,D) \in \{A,B\}$ et $O \neq D$. j étant la tâche transportée et n'apparaissant que si les temps de transport sont spécifiques aux tâches sinon il est omis. O est la station de départ du convoyeur, D sa station d'arrivée, et O ne peut être la même station que D. Par exemple, τ_{ABj_3} correspond au temps que met la tâche j_3 pour être transportée de la station A vers la station B ; tandis que τ_{BA} indique que tous les temps de transport pour aller de B vers A sont identiques. Enfin, il est à

noter que le convoyeur ne peut transporter qu'une tâche à la fois. Initialement, il est placé à la sortie d'un des stocks de sorties de l'une des stations.

3.2.2 Fonctionnement de l'atelier

Au départ, les deux stations sont considérées disponibles et prêtes à fonctionner. Les tâches doivent, tout d'abord, être placées sur les stocks d'entrées des deux stations. La façon dont les tâches sont placées dans les stocks d'entrées est appelée « politique de placement ». De même, pour le convoyeur, il faut dès le départ décider de son comportement. La description de ce comportement est appelée la « politique du convoyeur ». Ces deux politiques seront discutées un peu plus tard. Notons que le temps de placement des tâches qui vient d'être mentionné, n'est pas pris en compte dans le makespan ; en effet, il est considéré comme instantané.

Par exemple, pour un ensemble $N = \{j_1, j_2, j_3, j_4, j_5\}$ de cinq tâches, leur placement est décrit par la Figure 17.

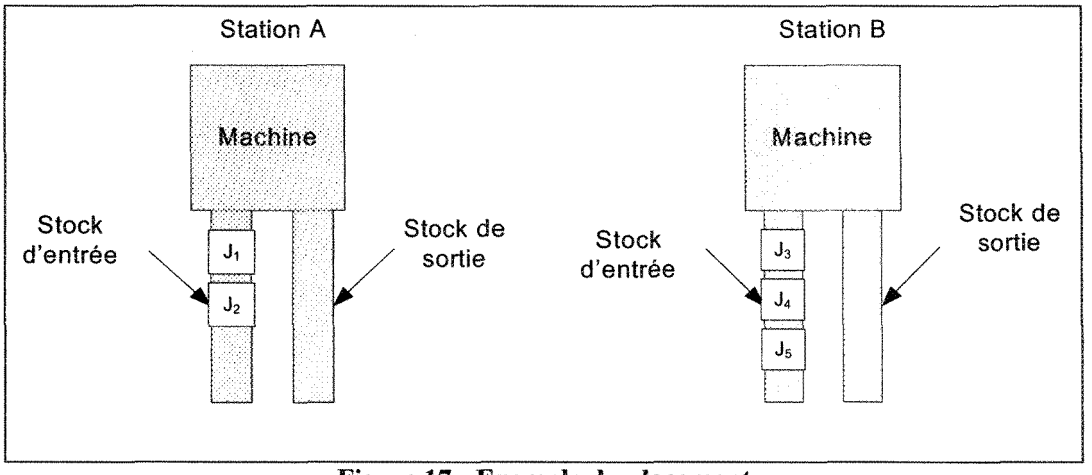

Figure 17 : Exemple de placement

Notons $N_{AB} = \{j_1, j_2\}$ et $N_{BA}=\{j_3, j_4, j_5\}$. En effet, toutes les tâches de N_{AB} ont une route d'exécution partant de la station A et finissant sur la station B. De même, pour les tâches de N_{BA} qui vont de la station B à la station A.

Une fois le placement des tâches opéré sur les stations, les premières tâches se trouvant sur chaque stock d'entrée, sur l'exemple j_1 et j_3, sont placées sur les machines et exécutés. A partir de cet instant, les machines exécutent immédiatement toutes les tâches dès que celles-ci arrivent dans leurs stocks d'entrée. Suivant sa politique de fonctionnement, le convoyeur quant à lui, transporte les tâches semi-finies d'une machine vers l'autre. La Figure 18, ci-dessous, schématise ce fonctionnement.

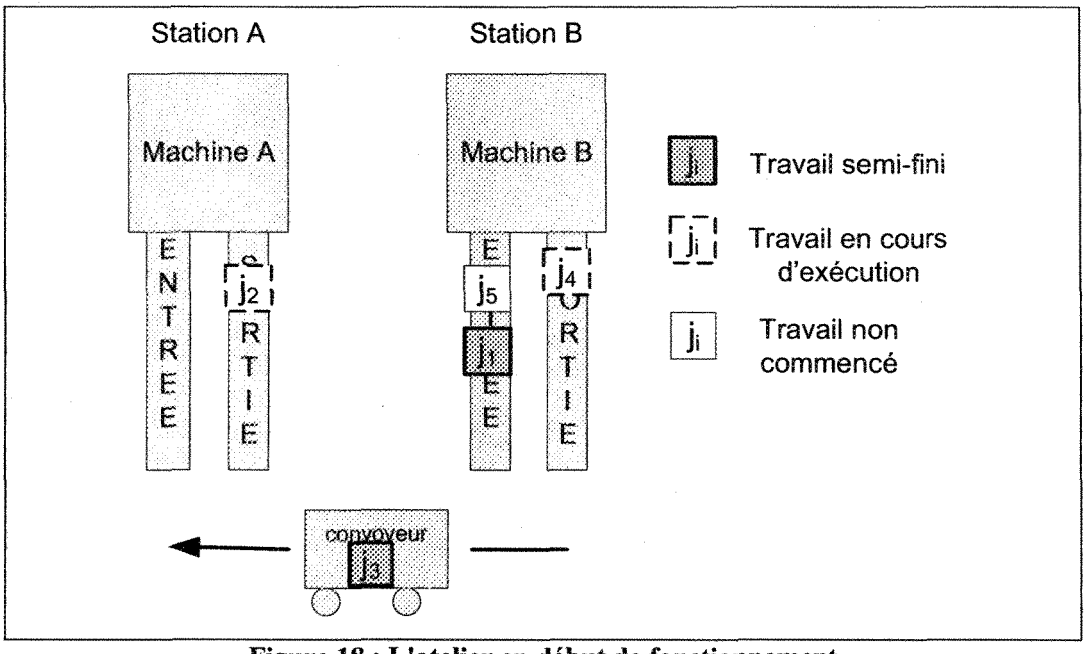

Figure 18 : L'atelier en début de fonctionnement

A cet instant, dans l'atelier, j_1 est semi-finie. A la suite de son transport, elle se trouve en attente dans le stock d'entrée de la station B. De même, j_3 est la

première tâche a avoir été exécutée sur la station B. Elle est en train d'être véhiculée vers le stock d'entrée de A. Les tâches j_2 et j_4 sont en cours d'exécution sur leurs machines respectives et, finalement j_5 n'est pas encore commencée et est en attente.

L'atelier s'arrête quand toutes les tâches sont réalisées, c'est-à-dire quand elles ont toutes suivi leurs routes d'exécutions prévues. Le makespan correspondant alors au temps écoulé depuis le démarrage de l'atelier. Il est à noter que le convoyeur possède également un temps d'accomplissement correspondant au temps passé entre le démarrage de l'atelier et la fin du déchargement de la dernière tâche transporter.

3.2.3 Complexité de l'atelier

Il est facile de voir que notre problème est NP-difficile au sens fort. En effet, il est montré par Hunrik et Knust [35] que le problème de flowshop à deux machines, un convoyeur et des espaces de stock illimités est NP-difficile au sens fort même si les temps de transport se résument à t_1 et que le temps de retour du convoyeur t_2 est nul. Or ce problème est un cas particulier de notre problème, pour cela il suffit de prendre $N_{BA} = 0$.

3.2.4 Quelques propriétés de l'atelier

Avant de faire une étude plus approfondie de l'atelier lorsqu'il doit effectuer un ensemble de n tâches quelconques, voici quelques propriétés particulières que notre travail nous a amené à mettre en exergue.

Propriété 1 : Le makespan du convoyeur est toujours inférieur à celui de l'atelier.

Démonstration : Comme décrit précédemment, le convoyeur est chargé de transporter les tâches semi-finies d'une station à l'autre et le makespan correspond au temps de la dernière tâche transportée. De plus, cette dernière tâche doit, après

son transport, être exécutée sur une machine avant de sortir de l'atelier. Puisque seules les tâches qui ont un temps d'exécution non nul sur la machine de destination sont transportées alors le makespan du convoyeur est donc toujours strictement inférieur au makespan du système. □

Propriété 2 : Le makespan de l'atelier correspond au maximum des temps d'accomplissement totaux des deux stations.

Démonstration : La dernière tâche, pour sortir de l'atelier, doit avoir suivi sa route d'exécution, c'est-à-dire être exécuté sur au moins une des deux machines. Elle est considérée réalisé lorsqu'elle sort de la dernière station appartenant à sa route d'exécution, posons que cela soit A. A sa sortie de A, deux cas sont envisageables, soit B a déjà exécuté toutes ses tâches et donc c'est la station A qui a le temps d'accomplissement le plus important correspondant ainsi au makespan du système. Soit la station B termine sa dernière tâche au même instant ou plus tard que la station A et la propriété est toujours vérifiée.□

Propriété 3 : Il n'y a pas de blocages possibles des machines ou du convoyeur.

Démonstration : Une situation de blocage survient lorsqu'au moins une machine dans un atelier ayant terminé l'exécution d'une tâche ne peut « relâcher » cette tâche pour en commencer une autre. Cette situation entraînant le blocage de l'atelier. Typiquement cela survient lorsqu'une station ne possédant pas de stock de sortie, sa machine termine une tâche, mais ne peut continuer, car le matériel devant prendre en charge la tâche terminée n'est pas disponible. Or dans notre atelier, chaque station possède deux zones de stockage de taille illimitée. Si le convoyeur n'est pas là au moment où la machine finit l'exécution d'une tâche, cela n'a pas d'incidence sur son fonctionnement, car elle la dépose sur son stock de sortie et peut donc continuer

normalement d'exécuter les tâches. De même, le convoyeur peut toujours déposer une tâche, il ne peut donc se retrouver bloqué. □

Propriété 4 : Les premières opérations des tâches sont ordonnancées de manière continuelle.

Démonstration : Soit une solution qui ne satisfait pas cette propriété. Il existe alors un temps t où la machine A (respt. B) est inactive. Si toutes les tâches exécutées après le temps t sont déplacées au temps t, le makespan n'augmentera pas et la solution obtenue est encore valide. □

Cette propriété permet de montrer que les premières opérations des machines définissent une permutation. Par la suite, il suffit de décrire les ordonnancements uniquement à l'aide des permutations des premières opérations.

Propriété 5 : Si les temps d'exécutions des tâches sont tous égaux et que les temps de transport sont quelconques alors le rangement en ordre décroissant ou croissant des tâches sur les deux machines suivant les temps de transport ne sont pas nécessairement optimaux.

Démonstration : Cette propriété est démontrée grâce au contre-exemple suivant. Six tâches ayant comme un temps d'exécution de 4 unités de temps sur les deux stations et dont les temps de transport sont donnés dans le Tableau 8 suivant, doivent être exécutés par l'atelier.

Tâches devant commencer sur la station A	Temps de transport	Tâches devant commencer sur la station B	Temps de transport
j_1	5	j_4	2
j_2	7	j_5	10
j_3	1	j_6	8

Tableau 8 : Données du contre-exemple

La Figure 19 montre les trois ordonnancements résultants du placement des tâches, tout d'abord, par (a) ordre croissant des temps de transport, puis par (b) ordre décroissant et enfin par (c) ordre alphanumérique des noms des tâches.

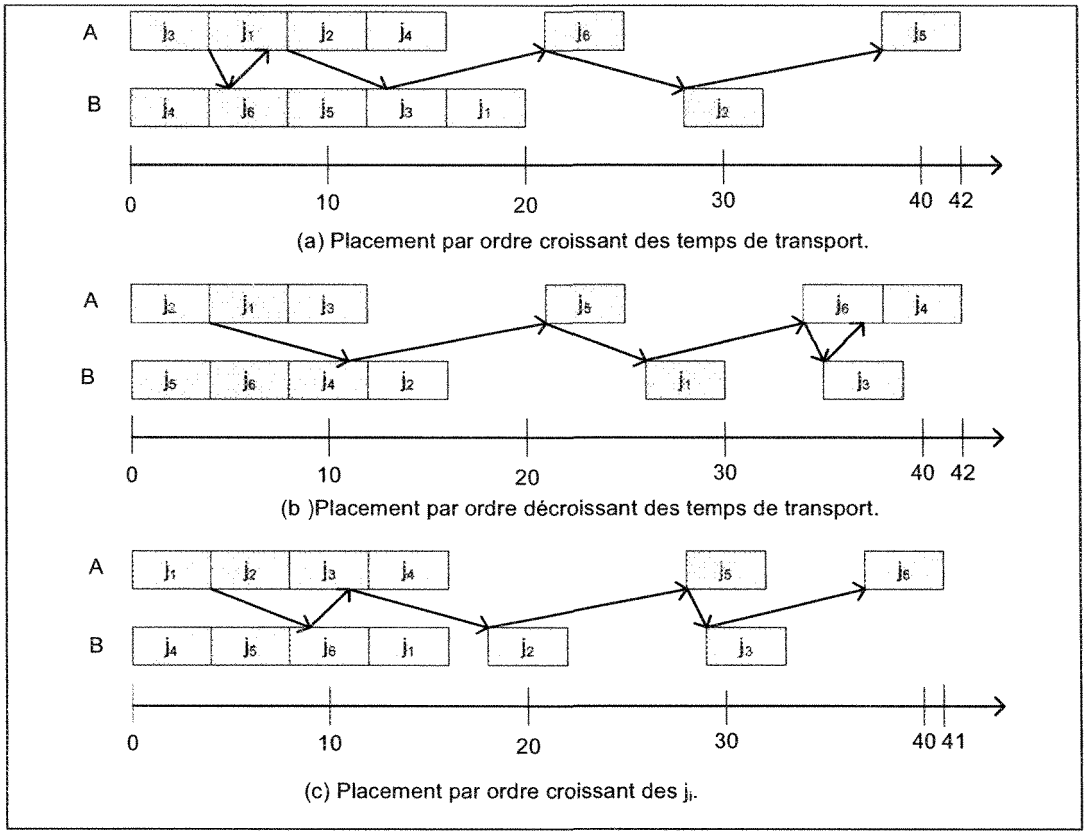

Figure 19 : Ordonnancements du contre-exemple

Le meilleur ordonnancement est donc donné par le placement des tâches par ordre alphanumérique des j_i, ce qui montre que le placement par ordre croissant ou décroissant des temps de transport dans le cas de tâches ayant tous le même temps d'exécution et des temps de transport quelconques n'est pas optimal. □

Propriété 6 : Si pour toutes les tâches, tous les temps de transport (y compris les temps de transport à vide éventuels) sont strictement inférieurs aux temps

d'exécution alors, n'importe quelle permutation est optimale, et le makespan a pour valeur $C_{\max} = \max\left\{\sum_{i=1}^{n} t_{Ai}, \sum_{i=1}^{n} t_{Bi}\right\}$.

Démonstration : Puisque tous les temps de transport sont strictement inférieurs au temps d'exécution alors la somme des temps de transport est strictement inférieure à la somme des temps d'exécution des tâches sur chaque station. Quelle que soit la politique du convoyeur, les tâches se retrouveront toujours à passer sans interruption sur les machines. Celles-ci n'auront pas de temps mort correspondant à l'attente d'une tâche, en effet le convoyeur aura toujours le temps nécessaire pour transporter les tâches. Le makespan sera alors le maximum des sommes des temps d'exécution des deux stations. □

Soit la stratégie S1 suivante : lorsque les temps de transport sont strictement supérieurs aux temps d'exécution, l'ordonnancement suivant, placé sur le stock de départ de la machine d'où part initialement le convoyeur est créé. En première position, la tâche qui a le temps d'exécution minimale sur cette machine puis en dernière position, la tâche qui a le temps d'exécution sur l'autre machine le plus faible . L'ordonnancement entre ses deux tâches est fait dans un ordre quelconque, de même que sur la station qui a le moins de tâches.

Propriété 7 : La stratégie S1 n'est pas optimale.

Démonstration : Ceci est démontré grâce au contre-exemple suivant. Notre atelier doit effectuer les sept tâches décrites dans le Tableau 9 suivant.

Nom de tâches	Temps d'exécution sur la station A	Temps d'exécution sur la station B	Temps de transport
j_1	2	1	9
j_2	1	2	6
j_3	2	3	8
j_4	1	2	7
j_5	3	1	7
j_6	1	3	9
j_7	2	4	6

Tableau 9 : Liste des tâches du contre-exemple

Les tâches j_1, j_2, j_3, j_4 commencent sur la station A et j_5, j_6 et j_7 sur la station B. La station possédant le plus de tâches sur son stock d'entrée est donc la station A. L'ordonnancement sur la station A, est j_2, j_4, j_3, j_1 car la tâche j_2 a le temps d'exécution sur A le plus faible et j_1 a le temps d'exécution le plus faible sur B. Sur B, les travaux sont laissés en ordre alphanumérique, j_5, j_6, j_7 . L'atelier avec ce placement initial a un temps d'accomplissement total de 55 unités de temps. Or pour les ordonnancements j_2, j_4, j_3, j_1 sur la station A et sur B j_5, j_7, j_6 donne un temps d'accomplissement de 54 unités de temps. La construction de l'ordonnancement proposé n'est donc pas optimale. □

Le makespan est dépendant du comportement du convoyeur et de l'ordonnancement des tâches. Dès lors, pour optimiser notre job shop, il faut donc les considérer ensemble. Malgré cela, il est intéressant d'étudier à part le convoyeur pour tenter de tirer des propriétés pouvant nous servir par la suite.

3.2.5 Étude du convoyeur

Le convoyeur est chargé de transporter les tâches semi-finies entre les deux stations. Lorsqu'il arrive, chargé, à une station et dépose une tâche sur le stock d'entrée, plusieurs cas de figure peuvent être envisagés. Ces cas dépendent de la

présence ou non de tâches en cours d'exécution sur la machine de la station ou présents dans son stock de sortie. Soit une, ou plusieurs, tâches sont en attentes dans le stock de sortie, alors l'une de celles-ci est chargée sur le convoyeur qui repart immédiatement, soit il n'y a pas de tâche en attente dans la zone de stockage et une décision sur le comportement du convoyeur doit être prise. Le choix du comportement va dépendre de la présence ou non d'une tâche en cours d'exécution sur la machine de la station. En effet, si une telle tâche n'existe pas, le convoyeur n'a d'autre choix que de repartir à vide. Dans le cas contraire, le convoyeur peut attendre la fin de l'exécution de la tâche ou repartir à vide. Ce choix est effectué en fonction de la station où ne se trouve pas le convoyeur, mais le critère permettant de choisir entre l'attente ou le retour à vide sera discuté plus tard.

Une autre décision à prendre concernant le convoyeur est le choix de son placement au départ de l'atelier. Une fois les tâches devant être exécutées par l'atelier, placées sur les stocks d'entrée des machines, le convoyeur doit être placé à la sortie d'un des stocks de sortie d'une des deux stations.

Pour essayer de répondre à ces questions sur le convoyeur, le comportement de l'atelier a été simulé par un programme Java. Ce programme prend comme données d'entrée un fichier contenant les tâches devant être exécutées par l'atelier. Ainsi, il a été créé, de façon aléatoire à partir d'une distribution uniforme, 4×100 instances de problèmes-tests de 10, 20, 50 et 200 tâches. Les temps d'exécution et de transport des tâches sont répartis d'une manière uniforme dans l'intervalle 0 et 50 unités de temps. Le programme exécute six fois chaque instance correspondant à la simulation de six passages dans l'atelier pour chaque ensemble de tâches : une fois pour chaque comportement et placement initial du convoyeur. Ces trois comportements sont ceux présentés ci-dessus et plus précisément : l'attente de la tâche en cours d'exécution, le retour à vide et celui où la station opposée (celle sur

laquelle le convoyeur ne se trouve pas) est prise en compte pour savoir s'il est plus intéressant d'attendre la tâche ou de repartir à vide. Plus précisément, s'il y a une tâche en attente dans le stock de la machine opposée et si le temps pour effectuer le transport à vide est inférieur au temps d'attente de la tâche en cours d'exécution alors le convoyeur repartira à vide. S'il n'y a pas de tâche en attente dans le stock de la station opposée, mais si la machine de cette station est en cours d'exécution d'une tâche et si la somme composée du temps de retour à vide $t_{retourVide}$, du temps d'exécution restant de la tâche sur la station opposée $t_{execRestantStationOpposée}$ et du temps de transport $t_{transport}$ de cette tâche est inférieur au temps d'attente $t_{attente}$ alors le convoyeur repartira à vide.

Le Tableau 10, ci-dessous, résume les différents comportements du convoyeur en fonction des situations possibles.

Comportement du convoyeur	Situations
Chargement de la tâche et transport vers la station opposée.	Une tâche est en attente dans le stock de sortie de la machine.
Attente.	Tâche en attente dans le stock de sortie ou tâche en cours d'exécution sur la station opposée et : $T_{retourVide} + t_{execRestantStationOpposée} + t_{transport} \geq t_{attente}$
Retour à vide.	Tâche en attente dans le stock de sortie ou tâche en cours d'exécution sur la station opposée et : $T_{retourVide} + t_{execRestantStationOpposée} + t_{transport} < t_{attente}$

Tableau 10: Comportements du convoyeur en fonction des différentes situations

Par la suite, « attente » notera le comportement du convoyeur quand celui-ci attend systématiquement la tâche en cours d'exécution, « retour à vide » quand il repartira systématiquement à vide et « dépend » des tâches correspondra soit à

l'attente ou au retour à vide dépendamment si l'inéquation $t_{retourVide} + t_{execRestantStationOpposée} + t_{transport} < t_{attente}$ est vrai ou fausse.

Ces algorithmes sont codés et exécutés sur un pentium III 996Mhz avec 384Mo de RAM. Une étude statistique des résultats générés est effectuée. Tout d'abord, cette étude porte sur la dépendance entre la position initiale du convoyeur et le nombre initial de tâches sur les stocks d'entrée des stations. En effet, intuitivement, la position initiale du convoyeur serait préférablement choisie sur la station qui a le plus de tâches dans son stock d'entrée. Ainsi, les retours à vide inutiles seraient évités.

Les deux histogrammes suivants, Figure 20 et Figure 21 montrent le nombre de fois (exprimé sous forme de pourcentage) où le placement initial du convoyeur permet d'atteindre le makespan le plus faible. La légende égalité indique que quelle que soit la position initiale du convoyeur, le makespan le plus faible a été atteint.

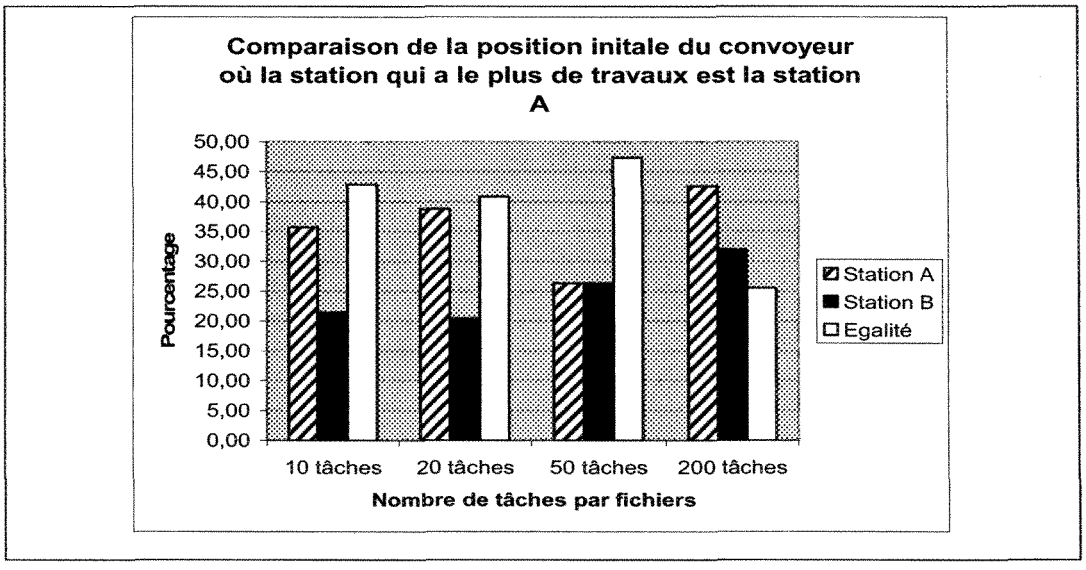

Figure 20 : Comparaison de la position initiale du convoyeur (station A)

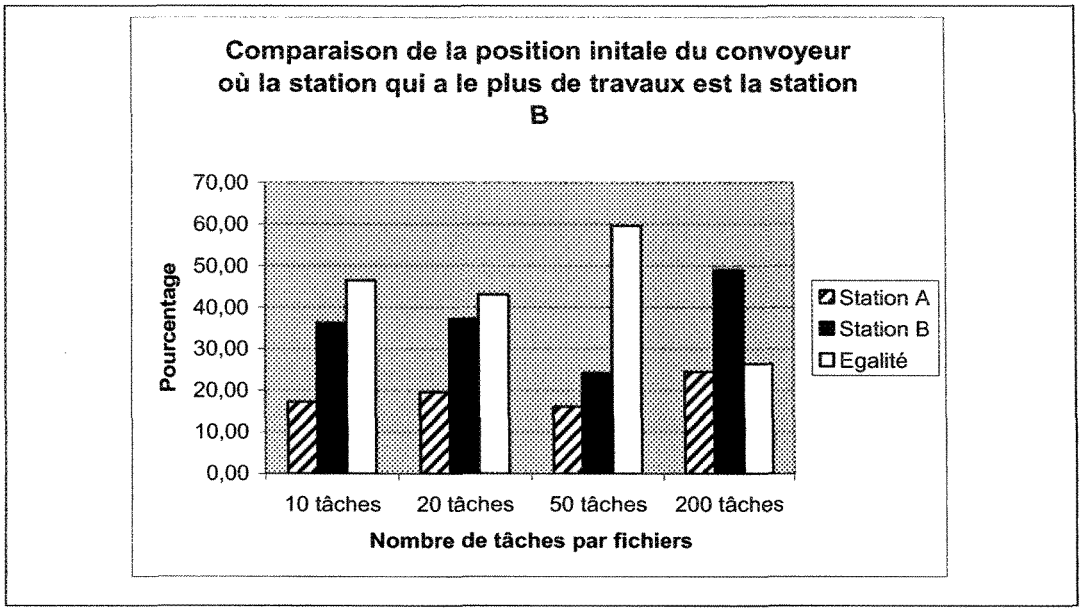

Figure 21 : Comparaison de la position initiale du convoyeur (station B)

Comme le montrent les figures, l'étude des résultats ne permet pas de déterminer une dominance. En effet, une dominance est établie lorsque, pour toutes les instances de problèmes-tests, la solution donnée pour une position du convoyeur est toujours égale ou meilleure que l'autre position. Or cela n'est pas le cas. Il est seulement possible de dire que positionner le convoyeur sur la station qui a le plus de tâches sur son stock d'entrée semble donner généralement de meilleurs résultats.

De la même façon, comme le montre la Figure 22, les comportements du convoyeur ont été comparés en calculant le nombre de fois où le temps le plus faible est atteint en fonction de la station de départ du convoyeur. Ainsi, seul le comportement du convoyeur influe sur le résultat.

Comme précédemment, il n'est pas possible de tirer de ces résultats une dominance. Mais pour les instances dont le nombre de tâches est peu élevé (10 tâches seulement) le comportement d' « attente » donne de meilleurs résultats que le

comportement de « retour à vide ». Toutefois, cette tendance semble s'inverser pour les instances ayant de plus grands nombres de tâches. Cela peut s'expliquer par le fait, que pour un nombre de tâches importants, la probabilité que le retour à vide permette de transporter une tâche pendant l'exécution d'une autre sur la station opposée est plus forte. De plus, la valeur du temps de retour à vide relativement faible (10 unités de temps) pour des fichiers de tâches dont les temps d'exécution et de transport sont longs est un avantage sur l'attente des tâches. Concernant le comportement « dépend », son pourcentage est comparable à celui de l' « attente » car en moyenne dans 80.5 % des instances le comportement « dépend » est le comportement de l' « attente ».

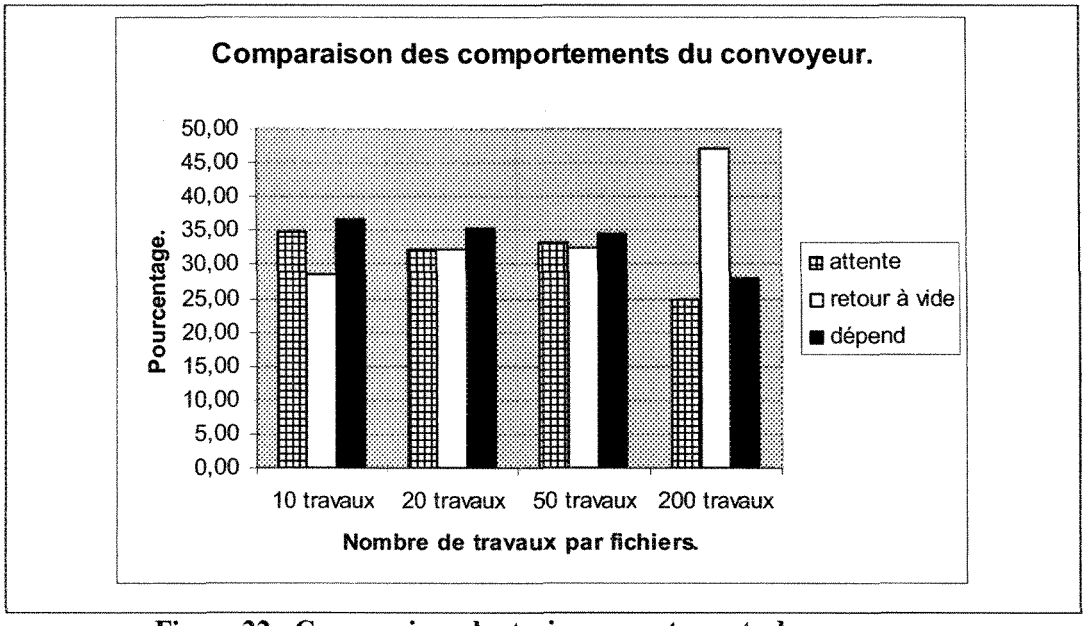

Figure 22 : Comparaison des trois comportements du convoyeur

Ne prendre en compte que le makespan pour le convoyeur ne permet pas d'appréhender toutes les spécificités de celui-ci. En effet, en ne considérant que le makespan, les temps d'attente ne sont pas distingués des temps de transport à vide.

Or, pour le convoyeur, les temps de transport à vide, contrairement aux temps d'attente, entraînent une usure et une dépense d'énergie supplémentaire pouvant induire des coûts financiers et de maintenance. Un temps supplémentaire a donc été introduit, le temps d'utilisation du convoyeur qui correspond à la somme des temps de transport des tâches, et des temps de transport à vide. Si le convoyeur est en attente sur une station alors il est considéré non utilisé et il n'y a pas d'ajout de temps supplémentaire à son temps d'utilisation.

Propriété 8 : Pour minimiser le temps d'utilisation du convoyeur alors celui-ci doit être positionné sur la station possédant le plus de tâches.

Démonstration : Si la station *A* possède plus de tâches sur son stock d'entrée que la station *B*, posons qu'il y a *n* tâches en attente sur le stock de la station *A* et *n-1* sur celui de la station *B*. Il est à noter que cette démonstration reste vraie si le nombre de tâches sur chaque station est inversé. Pour minimiser le temps d'utilisation du convoyeur, dans le cas où il n'y a pas de tâche dans le stock de sortie de sa station d'arrivée et qu'une tâche est en cours d'exécution sur la machine alors le convoyeur va attendre cette tâche. S'il n'y a pas de tâche en cours d'exécution alors le convoyeur sera forcé d'effectuer un retour à vide.

Si le convoyeur est initialement placé sur la station qui a le plus de tâches, dans ce cas *A*, il a au maximum $n + (n-1)$ tâches à transporter (certaines tâches peuvent n'avoir besoin que d'être exécuté sur une seule machine). Si toutes les tâches ont besoin d'être transportées alors le convoyeur va faire $n+(n-1)$ transports et ne faire aucun transport à vide. En effet, le convoyeur va tout d'abord faire les *n-1* aller et retour sans transport à vide, car le convoyeur va attendre au besoin les tâches. A la fin de ces *n-1* aller-retour, le convoyeur est revenu à la station *A*, où il

attend la $n^{ème}$ et dernière tâche, puis il la transporte sur la station *B*. Le convoyeur a donc terminé, et cela, sans faire aucun retour à vide, son temps d'utilisation est, dans ce cas, minimal.

En revanche, si le convoyeur est initialement placé sur la station *B*, il va comme précédemment faire *n-1* aller et retour et transporter les *2*(n-1)* tâches. Une fois cela accompli, il est de retour sur sa station de départ, c'est-à-dire *B*, or il reste une tâche sur la station *A* qui a besoin d'être transporté. Le convoyeur doit alors effectuer un retour à vide pour finalement transporter la dernière tâche. Dans ce cas, son temps d'utilisation n'est pas minimal, car il comporte un transport à vide. Il faut alors le placer sur la station qui a initialement le plus de tâches.

Notons que si les deux stations ont le même nombre de tâches attendant d'être exécutées sur leur stock d'entrée, posons *n* tâches chacun, alors le convoyeur peut être placée sur l'une ou l'autre des stations, car celui-ci fera toujours *2n* transports et cela sans faire de transport à vide. □

L'étude des cas simples de l'atelier et du convoyeur a permis une meilleure compréhension de l'atelier ; pour terminer sa description, il reste à présenter une borne inférieure pour le makespan.

3.2.6 Calcul d'une borne inférieure

Lors de l'approche heuristique du problème, celle-ci, comme nous l'avons déjà mentionné, ne fournit que des solutions approchées. Pour avoir connaissance de la qualité des solutions obtenues, il faut un élément de comparaison, le mieux étant bien sûr de prendre comme élément comparateur la solution optimale. Or, dans le cas présent, à notre connaissance, aucune solution optimale n'existe à l'heure actuelle, il faut donc trouver une autre valeur. Comme l'objectif est une

minimisation, cette valeur ne devra jamais être supérieure à la valeur d'une solution approchée. Cette valeur est appelée « borne inférieure » car elle borne les valeurs des solutions qui ne peuvent lui être inférieures.

Pour borner inférieurement notre problème, nous nous sommes inspirés du travail de V. A. Strusevitch [78]. Tout d'abord, il faut définir ce qu'est la charge de travail d'une station. La charge de travail d'une station *i* correspond à la somme des temps d'exécution des tâches que la station doit exécuter, et elle est notée $w_i(Q)$ avec Q un sous ensemble de N. Ceci peut se résumer pour nos deux stations A et B par :

$$w_A(Q) = \sum_{j \in Q} t_{Aj} \quad \text{et} \quad w_B(Q) = \sum_{j \in Q} t_{Bj}$$

De plus, $w_A(\varnothing) = w_B(\varnothing) = 0$, c'est-à-dire que si une station n'a pas de tâche à exécuter alors sa charge de travail est considérée nulle. Si par exemple $Q = N$, alors $w_A(N)$ représente la charge totale de travail de la station A.

La borne inférieure, notée LB, est donc, définie comme suit :

$$LB = \max\left\{\sum_j \tau_j, w_A(N), w_B(N), \max\{t_{Aj} + \tau_j + t_{Bj}\} + \min\{\tau_k\} \middle| j,k \in N\right\}$$

Il est clair que la relation $LB \leq C_{\max}$ est toujours vérifée. En effet, la charge totale des machines ne prenant en compte que les temps d'exécution des tâches, le makespan de l'atelier ne peut que leur être supérieur ou égal. Le cas d'égalité se produisant dans les cas où les machines n'ont aucun temps mort durant la totalité de fonctionnement de l'atelier.

En ce qui concerne la somme des temps de transport effectués, elle est inférieure au temps d'accomplissement du convoyeur, car les temps d'attente ou de retour à vide ne sont pas considérés. Or, précédemment il a été prouvé que le temps d'accomplissement du convoyeur était inférieur au makespan. Par conséquent, nous avons $\sum_j \tau_j < C_{\max}$.

Il reste à prouver que le makespan de l'atelier sera toujours supérieur à la somme constituée par la tâche la plus longue et le temps de transport le plus court. La tâche la plus longue est la tâche dont le temps constitué par ses temps d'exécution sur les deux stations et son temps de transport est le plus long.

Propriété 9 : $\max\{t_{Aj} + \tau_j + t_{Bj}\} + \min\{\tau_k\} \le C_{\max}$

Démonstration : Posons que j_1 soit la tâche possédant le temps d'accomplissement noté T le plus long et qu'elle se trouve initialement dans le stock d'entrée de la station A. A la fin de son exécution sur A, le temps d'accomplissement de A sera alors : $C_A = C_A^{'} + t_{A1}$ avec $C_A^{'}$ le temps d'accomplissement de la station A avant l'exécution de j_{A1}. Puis j_{A1} est transporté sur la station B par le convoyeur, de même que sur la station A, à la fin de son exécution, le temps d'accomplissement sur B est égal à : $C_B = C_B^{'} + t_{B1}$ avec $C_B^{'}$ le temps d'accomplissement de B avant l'exécution de j_{A1}.

Comme, le montre la Figure 23, lors de l'arrêt de l'atelier, le temps d'accomplissement de la station A et de la station B sont les suivants :

$$\begin{cases} C_A = C_A^{'} + t_{A1} + C_A^{''} \\ C_B = C_B^{'} + t_{B1} + C_B^{''} \end{cases}$$

$C_A^{"}$ et $C_B^{"}$ étant les temps d'accomplissement des tâches exécutées sur A et B après j_1.

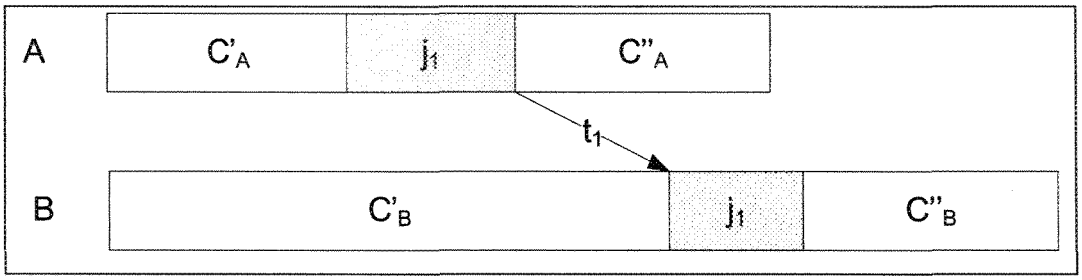

Figure 23 : Place de la tâche la plus longue

Comme il a été vu précédemment, le makespan correspond au maximum des temps d'accomplissement des stations, c'est-à-dire $C_{\max} = \max\{C_A, C_B\}$. Au temps de la tâche la plus longue, il faut ajouter le temps de transport le plus court de toutes les tâches. Ce temps de transport est considéré comme ne pouvant être celui de la tâche la plus longue et l'atelier ne peut au minimum n'exécuter que deux tâches : la tâche la plus longue et celle possédant le temps de transport le plus court.

Il existe donc toujours une tâche j appartenant soit à $C_A^{'}$, à $C_B^{'}$, à $C_A^{"}$ ou à $C_B^{"}$ qui a le temps de transport minimum. Puisque cette tâche est transportée, elle a deux temps d'exécution non nuls sur chacune des machines de chaque station, donc : $\forall C_A^{'}, C_B^{'}, C_A^{"}, C_B^{"} \in \mathbb{N} \quad T + \min\{\tau_i\} < C_{\max}$.□

Cette borne inférieure LB est donc le comparateur entre les heuristiques que nous allons proposer.

3.3 Conclusion

Le problème de job shop, présenté dans ce mémoire, possède deux stations et un convoyeur. Il a pour fonction objectif la minimisation du makespan. Chaque

station est elle-même composée d'une machine et deux espaces de stockages : un stock d'entrée où sont placées les tâches en attente d'exécution et un stock de sortie ou les tâches semi-finies ou terminées sont placées. Un convoyeur est chargé de transporter les tâches semi-finies d'une station à l'autre. Il ne peut transporter qu'une seule tâche à la fois et est initialement placé sur le stock d'entrée d'une des deux stations.

Nous avons montré, que le temps d'accomplissement du convoyeur est toujours inférieur au makespan, et que ce dernier temps correspond au maximum des temps d'accomplissement totaux des deux stations. De plus, grâce aux stocks illimités, ni les machines ni le convoyeur ne peuvent se bloquer. Trois autres propriétés sur des tâches ayant des caractéristiques ont été démontrées. En effet, si les temps d'exécutions des tâches sont tous égaux, mais que les temps de transport sont tous quelconques alors les politiques de placement par rangement en ordre croissant ou décroissant suivant les temps de transport ne sont pas optimaux. De même, si pour des tâches ayant des temps d'exécution tous inférieurs aux temps de transport alors la politique de placement consistant à placer en première position sur la station qui a le plus de tâches, la tâche qui a le temps d'exécution le plus court et à la fin de la séquence de tâches la tâche qui a le temps d'exécution le plus faible sur la station opposée n'est pas optimale. En revanche, si tous les temps de transport sont strictement inférieurs aux temps d'exécution alors n'importe quelle permutation de tâches est optimale et le makespan est égal au maximum des sommes des temps d'exécution des tâches que les deux stations doivent exécutées, aussi appelées charges des stations.

Par la suite, le convoyeur a été étudié en commençant par sa position initiale. Il est apparu que bien que les résultats de la simulation effectuée ne soient pas catégoriques, il semblerait que la meilleure position soit celle du stock de sortie de la

station possédant initialement le plus de tâches. Une fois l'atelier lancé, le convoyeur a trois comportements possibles, soit il attend les tâches en cours d'exécution, soit il repart systématiquement à vide et enfin suivant si l'inéquation $t_{voyageVide} + t_{execRestantStationOpposée} + t_{transport} < t_{attente}$ est vrai ou fausse, le convoyeur attendra ou repartira à vide. Au vu des résultats de la simulation, pour des instances de problèmes comportant peu de tâches, le comportement d' « attente » semble le plus intéressant. Toutefois, pour des instances ayant un nombre élevé de tâches, le comportement « à vide » semble meilleur. Ce résultat doit être nuancé, car le temps de retour à vide était, dans ce cas, de valeur relativement faible. Concernant le comportement « dépend », dans la plupart des cas que nous avons testés, celui-ci correspond au comportement d' « attente ». Il est à noter que pour minimiser le temps d'utilisation du convoyeur où seul le temps de mouvement du convoyeur est pris en compte alors c'est le comportement d' « attente » qui doit être privilégié.

A notre connaissance, il n'a pas été trouvé de solution optimale pour ce problème. Pour terminer cette description de l'atelier, une borne inférieure a donc été proposé, qui va permettre, par la suite, la comparaison d'heuristiques. Il a été montré que la borne correspond au maximum des charges des stations, de la somme des temps de transport et de la somme composée par le temps de la tâche la plus longue et du temps de transport le plus court.

Le problème d'atelier à deux stations et un convoyeur étant maintenant bien défini, il faut alors le résoudre. Pour cela, nous avons commencé par proposer des règles de priorité pour le placement des premières opérations des tâches.

CHAPITRE 4

RÉSOLUTION PAR RÈGLES DE PRIORITÉ

4.1 Introduction

Lors de la résolution d'un problème d'ordonnancement, si celui-ci est montré NP-difficile, alors deux familles de méthodes, les méthodes exactes et les méthodes approchées peuvent être utilisées. Comme le nombre de solutions possibles ne permet pas leur énumération en un temps raisonnable, nous allons utiliser, pour résoudre notre problème, les méthodes approchées et plus précisément les heuristiques. Celles-ci ont comme particularité la rapidité dans la génération des solutions [71].

Pour résoudre le problème de job shop à deux stations et un convoyeur, nous avons tout d'abord testé plusieurs règles de priorité, basées sur la notion de priorité entre les activités à ordonnancer. Leurs principaux avantages sont, en général, leur simplicité et surtout leur rapidité. De plus, ce type d'heuristiques est fréquemment appliqué au problème de job shop [8].

Trois règles ont donc été appliquées pour l'ordonnancement des tâches sur les deux stations. Les deux premières règles sont basées sur le couplage des tâches. La troisième règle de placement, quant à elle, est basée sur l'algorithme de Jackson. L'application de cette dernière règle permet de déterminer si, malgré la présence des temps de transport, cet algorithme génère des solutions acceptables. Finalement, dans le but d'améliorer nos résultats, nous avons modifié les règles en nous inspirant des articles de Mitten [58] et Panwalkar [64].

4.2 Descriptions des règles de priorité

Les deux premières règles de priorité proposées sont basées sur l'appariement des tâches des deux ensembles N_{AB} et N_{BA}. On rappelle que l'ensemble N_{LJ} contient les tâches ayant une route d'exécution de la station L vers la station J.

Ces deux règles fabriquent le maximum de couple de tâches possible entre les deux ensembles N_{AB} et N_{BA}. La première règle nommée règle R1, que nous avons conçue, forme des couples de tâches (k, j) qui sont le résultat de l'appariement des tâches ayant leurs temps d'exécution les plus courts donc tels que $k \in \{min\ t_{Ak}\}$ et $j \in \{min\ t_{Bj}\}$.

La seconde règle conçue, nommée R2, quant à elle, forme les couples (k, j) dont les tâches k ont un temps d'exécution les plus courts avec les tâches j qui ont les temps d'exécution les plus importants, c'est-à-dire (k, j) tel que $k \in \{min\ t_{Ak}\}$ et $j \in \{max\ t_{Bj}\}$. Il est également possible de choisir $k \in \{min\ t_{Bk}\}$ et $j \in \{max\ t_{Aj}\}$.

Pour chaque règle, une fois les couples créés, l'algorithme de Johnson est appliqué sur la station qui a le moins de tâches. Puis sur la station où l'algorithme de Johnson n'a pas été appliqué, les tâches sont rangées de façon à retrouver le couplage créé précédemment.

Le choix de la station ayant le moins de tâches dans son stock d'entrée pour l'application de l'algorithme de Johnson s'explique par le fait que l'on désire garder le couplage. En effet, sur la station ayant le plus de tâches, le couplage implique que certaines tâches seront « célibataires ». Or, si l'algorithme de Johnson est appliqué sur cette station, un mélange de tâches couplées et non couplées sera créé. Le rangement, pour rétablir le couplage entre les tâches sur la station ayant le moins de tâches, impliquera alors la création de « vide » pour garder le couplage ; ce qui n'est pas souhaitable.

Par exemple, si $N_{AB} = \{k_1, k_2, k_3, k_4\}$ et $N_{BA} = \{j_1, j_2\}$ et que le couplage « petit-petit », suivant R1, ait généré les couples (k_1, j_2) et (k_4, j_1). Deux tâches sont alors « célibataires » k_2 et k_3. L'application de l'algorithme de Johnson sur la station qui a le plus de tâches, c'est-à-dire A, génère la séquence k_3, k_1, k_2, k_4. Comme le

montre la Figure 24, le rangement sur la station B, permettant de retrouver le couplage, entraîne la création de deux tâches vides v tels que v, j_2, v, j_1.

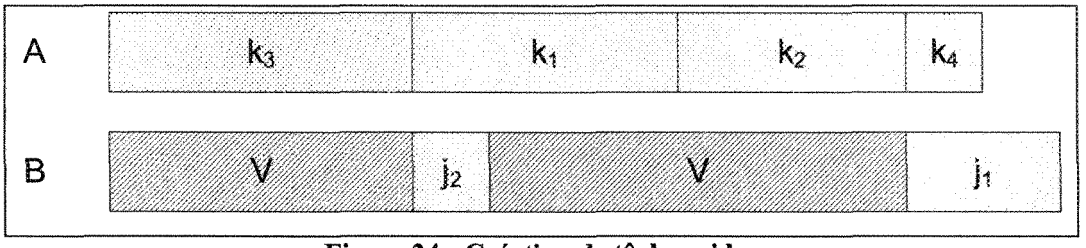

Figure 24 : Création de tâches vides

Il est clair que cet ajout de tâches vides va nuire à la qualité des solutions. Les algorithmes des deux premières règles de priorité sont comme suit.

Algorithme petit-petit

1. Sélectionner parmi $\{N_{AB}, N_{BA}\}$ l'ensemble qui contient le moins de tâches. Supposons N_{AB} cet ensemble
2. Tant qu'il y a des tâches dans N_{AB} faire
   1.4. Créer un couple $(k_{N_{AB}}, j_{N_{BA}})$ tel que :
      1.4.1. $k_{N_{AB}}$ <- tâche qui a le temps d'exécution t_{Ak} minimum de N_{AB}
      1.4.2. $j_{N_{BA}}$ <- tâche qui a le temps d'exécution t_{Bj} minimum de N_{BA}
   1.5. Garder en mémoire ce couple
   1.6. Placer les tâches $k_{N_{AB}}$ et $j_{N_{BA}}$ dans les stocks d'entrées de leurs stations respectives
3. Fin TantQue
4. Appliquer l'algorithme de Johnson sur le stock d'entrée de A
5. Réordonner les taches contenues dans le stock de B de façon à ce que les tâches se trouvent à la même position dans le stock que celle de la tâche avec laquelle elles ont été précédemment couplées

Algorithme petit-grand

1. Sélectionner parmi $\{N_{AB}, N_{BA}\}$ l'ensemble qui contient le moins de tâches. Supposons N_{AB} cet ensemble
2. Tant qu'il y a des tâches dans N_{AB} faire
   1.7. Créer un couple $(k_{N_{AB}}, j_{N_{BA}})$ tel que :

1.7.1. Si A est la station où l'on souhaite sélectionner les tâches qui ont les temps d'exécution les plus long alors
1.7.1.1. $k_{N_{AB}}$ <- tâche qui a le temps d'exécution t_{Ak} maximum de N_{AB}
1.7.1.2. $j_{N_{BA}}$ <- tâche qui a le temps d'exécution t_{Bj} minimum de N_{BA}
1.7.2. Sinon
1.7.2.1. $k_{N_{AB}}$ <- tâche qui a le temps d'exécution maximum de N_{AB}
1.7.2.2. $j_{N_{BA}}$ <- tâche qui a le temps d'exécution minimum de N_{BA}
1.7.3. fin Si
1.8. Garder en mémoire ce couple
1.9. Placer les tâches $k_{N_{AB}}$ et $j_{N_{BA}}$ dans les stocks d'entrées de leurs stations respectives
3. Fin TantQue
4. Appliquer l'algorithme de Johnson sur le stock d'entrée de A
5. Réordonner les taches contenues dans le stock de B de façon à ce que les tâches se trouvent à la même position dans le stock que celle de la tâche avec laquelle elles ont été précédemment couplées

NB : Dans ces deux algorithmes, c'est l'ensemble N_{AB} qui est supposé contenir le moins de tâches. Or, si ce n'est pas le cas, alors il suffit de permuter les ensembles et les stations dans les deux algorithmes.

La troisième règle, que nous présentons, consiste à appliquer l'algorithme de Jackson. Ce choix de cette règle est motivé par le fait que cet algorithme est performant lorsque les opérations de transport sont considéré instantanée, il est donc intéressant de l'étudier quand cela n'est plus le cas.

Notons que ces trois règles de priorité ont une complexité temporelle en $O(n \log n)$.

4.2.1 Application de ces règles sur un exemple

Illustrons maintenant par un exemple ces trois règles. Par la suite, seuls les ordonnancements sur les deux stations au départ de l'atelier seront présentés. Soient les tâches décrites par le Tableau 11 suivant.

Identification de la tâche	Temps d'exécution sur A	Temps d'exécution sur B	Station de départ
k_1	4	5	A
k_2	3	4	A
k_3	9	2	A
k_4	7	4	A
k_5	5	3	A
j_1	1	8	B
j_2	7	6	B
j_3	4	7	B
j_4	3	2	B

Tableau 11 : Données de l'exemple

Les ensembles N_{AB} et N_{BA} sont les suivants : $N_{AB} = \{k_1, k_2, k_3, k_4, k_5\}$ et $N_{BA} = \{j_1, j_2, j_3, j_4\}$. La station qui a le moins de tâches est la station B, c'est donc sur celle-ci que l'algorithme de Johnson sera appliqué pour les deux premières règles. Cet algorithme retourne la séquence suivante : j_1, j_3, j_2, j_4.

Commençons par la règle R1. La tâche de temps minimum sur A est k_2 et celle sur B est j_4, ce qui donne le couple (k_2, j_4). De la même façon, les couples (k_1, j_2), (k_5, j_3) et (k_4, j_1) sont obtenus. La tâche k_3 est célibataire. Suite à l'algorithme de Johnson, les tâches de la station A sont rangées pour retrouver les couples, ce qui donne la séquence : i_4, i_5, i_1, i_2, i_3. Finalement, la règle R1 construit l'ordonnancement décrit par la Figure 25 suivante.

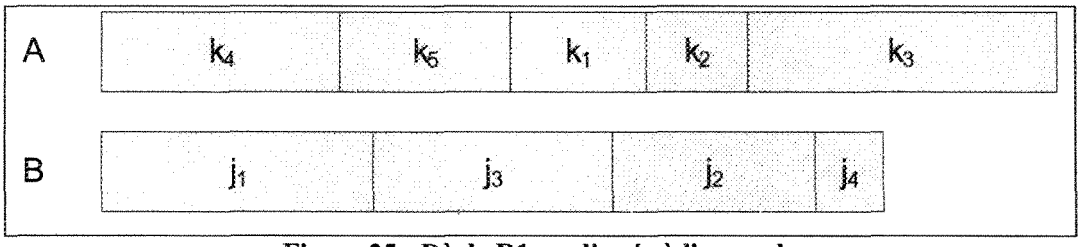

Figure 25 : Règle R1 appliquée à l'exemple

Pour la règle R2, commençons par choisir la station *A* pour sélectionner les tâches de temps d'exécution les plus courtes et la station *B* pour le choix des tâches les plus longues. La tâche la plus courte sur *A* est k_2. Sur *B*, la tâche de temps d'exécution la plus longue est j_1, ceci crée le couple (k_2, j_1). De la même façon, sont créés les couples (k_1, j_3), (k_5, j_2) et (k_4, j_4). C'est encore k_3 qui est célibataire. Le rangement des tâches sur *A*, après l'application de Johnson sur la station *B*, donne : k_2, k_1, k_5, k_4, k_3. La Figure 26 montre l'ordonnancement résultant.

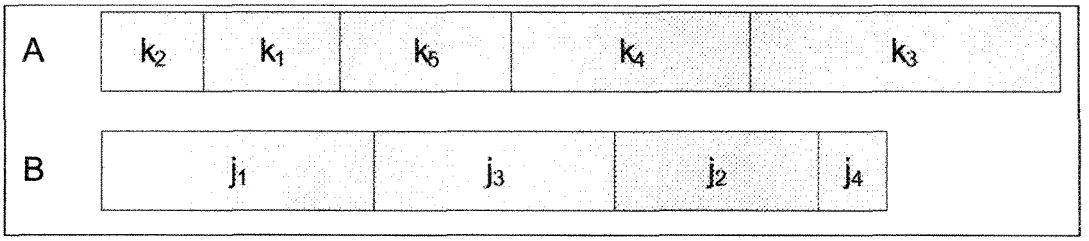

Figure 26 : Règle R2 avec les temps d'exécution les plus courts sur la station *A*

Appliquons une nouvelle fois, la règle R2, mais cette fois-ci avec *B* la station sur laquelle les tâches ayant le temps d'exécution les plus courts sont sélectionnées, et *A* les temps d'exécution les plus longs. Les couples suivants (j_4, k_3), (j_2, k_4), (j_3, k_5), (j_1, k_1) sont créés. La tâche k_2 est la tâche célibataire. Le rangement sur *A* après l'application de l'algorithme de Johnson donnera alors : k_1, k_5, k_4, k_3, k_2. La Figure 27 montre l'ordonnancement sur les deux stations.

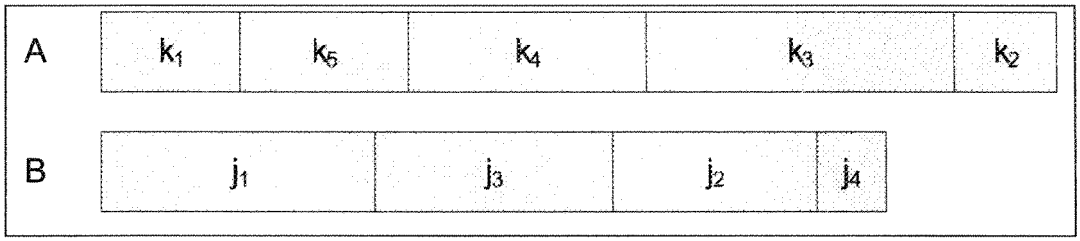

Figure 27 : Règle R2 avec les temps d'exécution les plus courts sur la station *B*

La dernière règle est celle de Jackson. L'application de cette règle, comme le montre la Figure 28, donne l'ordonnancement k_2, k_1, k_4, k_5, k_3 sur la station A et sur B les tâches sont ordonnancés de façon quelconque, par exemple alphanumérique, j_1, j_2, j_3, j_4.

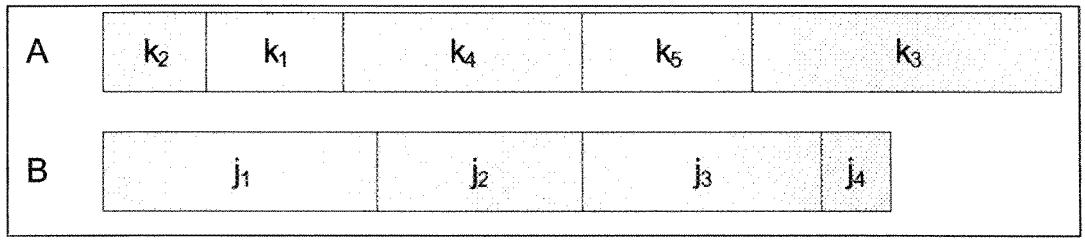

Figure 28 : Règle de Jackson appliquée à l'exemple

4.3 Résultats

Les règles de priorité ont été simulées grâce au programme précédemment employé pour l'étude du convoyeur. Par soucis de comparaison, un ordonnancement aléatoire des stocks a aussi été mis en place. Il consiste à remplir les stocks d'entrée dans l'ordre de lecture du fichier de données. Les instances de problèmes-tests conçues lors de l'étude du convoyeur ont été reprises et pour chaque règle, les différents comportements du convoyeur ont été appliqués. Rappelons que 4×100 instances de problèmes-tests de 10, 20, 50 et 200 tâches ont été générées de manière aléatoire à partir d'une distribution uniforme. Les temps d'exécution et de transport

des tâches sont répartis uniformément dans [0, 50]. Concernant la position initiale du convoyeur, il a été choisi de le placer sur la station possédant le plus de tâches dans son stock d'entrée. Le temps de retour à vide du convoyeur a été fixé à 10 unités de temps. Par soucis de clarté, la règle de Jackson est notée RJ.

Un résumé des résultats, selon la taille des instances, est donné par les tableaux suivants (Tableaux 12, 13, 14 et 15). Dans ces tableaux, et par la suite, le comportement du convoyeur est spécifié : attente, vide (pour retour à vide) et dépend (pour le comportement où le convoyeur attend ou effectue un retour à vide dépendamment si l'inéquation $t_{voyageVide} + t_{execRestantStationOpposée} + t_{transport} < t_{attente}$ est vrai ou fausse). Concernant la règle R2, il est aussi spécifié, si les tâches, ayant les temps d'exécution les plus longs ou les plus courts, lors de la construction de la règle ont été sélectionnées sur la station *A*. Par exemple, « R2 Vide A Gd » marque l'application de la règle R2, le comportement du convoyeur est le retour à vide et lors de la construction de la règle, ce sont les tâches ayant les temps d'exécution les plus longs qui ont été sélectionnées sur la station *A*.

Les tableaux suivants expriment la déviation moyenne qui est le pourcentage moyen de la différence entre le temps d'accomplissement donné par la règle et la borne inférieure. L'écart type rend compte de la dispersion des résultats. Les tableaux donnent également la déviation maximale et minimale. Enfin, on présente le nombre de fois où la borne est atteinte, où la règle obtient la meilleure puis la pire solution par rapport aux résultats des autres règles.

Notons que les résultats concernant la déviation minimale sont explicables par le fait qu'un certain nombre des instances de problèmes-tests donne systématiquement la borne inférieure quelque soit l'ordonnancement sur les stations effectué. Les temps moyens de calcul n'ont pas été intégrés à ces résumés de

résultats, car ils seront discutés plus tard. Finalement, on remarque que les valeurs des lignes du nombre de fois que la règle donne la meilleure, la pire solution et la borne donnent une somme plus grande que le nombre d'instances expérimentées. Cela s'explique, par le fait que dans un certain nombre de cas les règles donnent le même makespan.

Au vu de ces résultats, il semble difficile de discriminer les deux premières règles R1 et R2. En effet, les valeurs concernant la déviation moyenne par rapport à la borne sont similaires. L'explication de ces résultats très similaires peut être la construction des règles. Comme il a été vu dans l'exemple précèdent, cette construction donne une bonne diversité pour la station ayant le plus de tâches. Mais concernant la station ayant le moins de tâches, son ordonnancement est toujours identique pour les deux premières règles.

Règle de priorité	R1 attend	R1 vide	R1 dépend	R2 attend A Pt	R2 vide A Pt	R2 dépend A Pt	R2 attend A Gd	R2 vide A Gd	R2 dépend A Gd	RJ attend	RJ vide	RJ dépend	Aléatoire
Déviation moyenne par rapport à la borne inférieure (en %)	19,99	21,77	19,74	22,25	23,50	22,12	22,41	23,92	22,32	10,95	13,44	10,78	21,82
Ecart type de la déviation	13,52	14,21	13,80	15,59	16,21	15,91	15,02	15,21	15,08	10,39	11,55	10,34	14,31
Déviation maximale par rapport à la borne inférieure	58,63	54,26	58,63	63,67	70,31	63,67	66,22	61,75	66,22	39,79	37,79	39,79	56,67
Déviation minimale par rapport à la borne inférieure	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Nombre de fois que la borne est atteinte	12	12	15	13	13	15	11	9	11	26	23	28	10
Nombre de fois que la règle obtient la meilleure solution	16	12	19	16	13	17	13	9	13	81	33	83	12
Nombre de fois que la règle obtient la pire solution	9	23	9	18	31	17	17	29	15	4	4	4	25

Tableau 12 : Résumé des résultats pour les instances de 10 tâches

Règle de priorité	R1 attend	R1 vide	R1 dépend	R2 attend A Pt	R2 vide A Pt	R2 dépend A Pt	R2 attend A Gd	R2 vide A Gd	R2 dépend A Gd	RJ attend	RJ vide	RJ dépend	Aléatoire
Déviation moyenne par rapport à la borne inférieure (en %)	16,54	17,60	16,46	18,43	18,55	17,97	17,65	18,82	17,75	9,98	11,84	9,81	17,86
Ecart type de la déviation	12,45	12,67	12,44	13,38	13,51	13,20	12,37	12,71	12,41	10,23	10,94	10,16	12,24
Déviation maximale par rapport à la borne inférieure	50,00	53,20	50,00	53,65	57,76	53,65	50,46	48,86	50,46	34,02	38,36	34,02	56,62
Déviation minimale par rapport à la borne inférieure	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Nombre de fois que la borne est atteinte	16	15	16	10	12	11	15	14	15	33	25	33	10
Nombre de fois que la règle obtient la meilleure solution	17	16	17	11	12	12	15	15	15	90	31	93	10
Nombre de fois que la règle obtient la pire solution	16	23	15	28	26	17	14	28	16	8	8	8	23

Tableau 13 : Résumé des résultats pour les instances de 20 tâches

Règle de priorité	R1 attend	R1 vide	R1 dépend	R2 attend A Pt	R2 vide A Pt	R2 dépend A Pt	R2 attend A Gd	R2 vide A Gd	R2 dépend A Gd	RJ attend	RJ vide	RJ dépend	Aléatoire
Déviation moyenne par rapport à la borne inférieure (en %)	14,94	15,42	14,91	16,50	16,38	16,29	15,80	15,98	15,77	11,94	12,96	11,94	16,21
Ecart type de la déviation	10,74	10,89	10,73	11,33	11,47	11,25	11,12	11,19	11,14	10,34	10,57	10,34	11,33
Déviation maximale par rapport à la borne inférieure	39,26	40,41	39,26	41,28	41,43	40,25	42,85	41,83	42,85	36,98	38,47	36,98	41,83
Déviation minimale par rapport à la borne inférieure	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Nombre de fois que la borne est atteinte	16	13	16	13	13	14	11	11	11	20	18	20	12
Nombre de fois que la règle obtient la meilleure solution	16	13	16	13	13	14	11	11	11	97	21	97	12
Nombre de fois que la règle obtient la pire solution	14	20	14	31	28	21	28	25	26	11	11	11	28

Tableau 14 : Résumé des résultats pour les instances de 50 tâches

Règle de priorité	R1 attend	R1 vide	R1 dépend	R2 attend A Pt	R2 vide A Pt	R2 dépend A Pt	R2 attend A Gd	R2 vide A Gd	R2 dépend A Gd	RJ attend	RJ vide	RJ dépend	Aléatoire
Déviation moyenne par rapport à la borne inférieure (en %)	13,63	13,74	13,62	14,01	13,98	13,96	13,91	13,92	13,90	12,67	13,01	12,67	14,41
Ecart type de la déviation	9,59	9,57	9,58	9,64	9,66	9,62	9,71	9,71	9,71	9,42	9,47	9,42	9,45
Déviation maximale par rapport à la borne inférieure	34,93	35,30	34,93	36,26	36,22	36,10	35,13	35,30	35,13	34,30	34,67	34,30	35,32
Déviation minimale par rapport à la borne inférieure	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Nombre de fois que la borne est atteinte	8	8	8	8	8	8	8	8	8	10	10	10	7
Nombre de fois que la règle obtient la meilleure solution	8	8	8	8	8	8	8	8	8	100	10	100	7
Nombre de fois que la règle obtient la pire solution	9	11	8	22	20	14	18	18	17	7	7	7	51

Tableau 15 : Résumé des résultats pour les instances de 200 tâches

La règle de Jackson semble être la plus profitable, car elle atteint le plus de fois la meilleure solution et la borne inférieure. De plus, sa déviation moyenne, son écart-type et sa déviation maximale sont inférieurs à ceux des deux premières règles, et cela, quel que soit le comportement du convoyeur. A l'opposé, la règle R2 avec semble la moins intéressante, car elle génère le plus souvent la plus mauvaise solution. Cette tendance se vérifie quand un classement des règles est tenté. Les quatre graphiques suivants (Figures 29, 30, 31 et 32) présentent la dominance stochastique entre les règles. La dominance stochastique est définie de la façon suivante. Soient R_A et R_B deux règles et x_A et x_B le nombre de solutions données respectivement par ces règles dont la valeur est inférieure à la déviation. Alors, la règle R_A domine stochastiquement R_B si pour toutes les valeurs de la déviation de la borne, x_A est supérieure à x_B. Ces graphiques montrent que la règle de Jackson domine stochastiquement les deux autres règles.

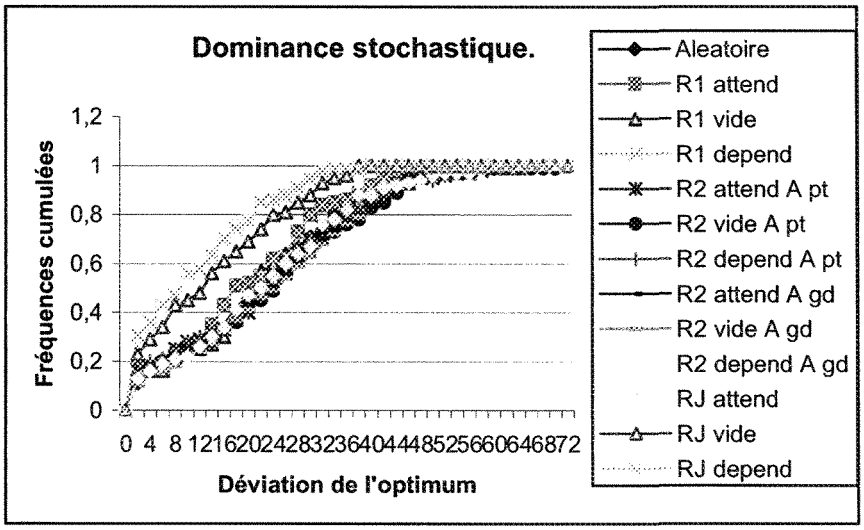

Figure 29 : Dominance stochastique pour instances de 10 tâches

La dominance stochastique de la règle de Jackson, par rapport aux règles R1 et R2, s'estompe lorsque le nombre de tâches à effectuer devient plus important. Suite à ces résultats, une recherche d'amélioration de ces règles a été tentée.

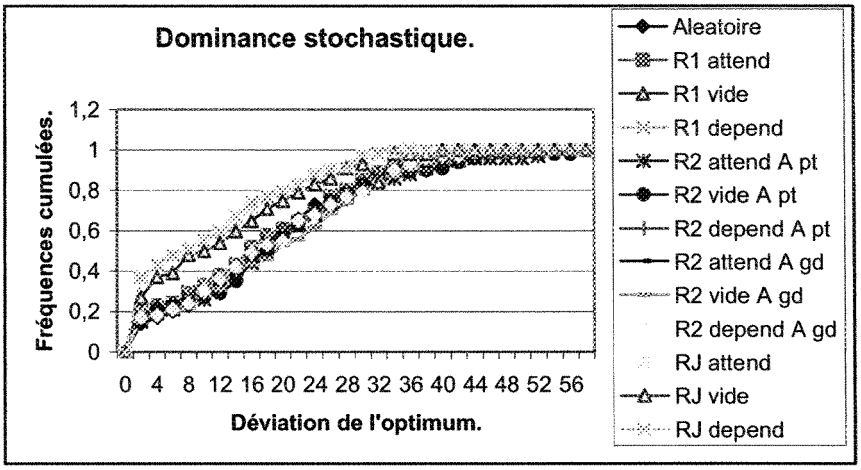

Figure 30 : Dominance stochastique pour les instances de 20 tâches

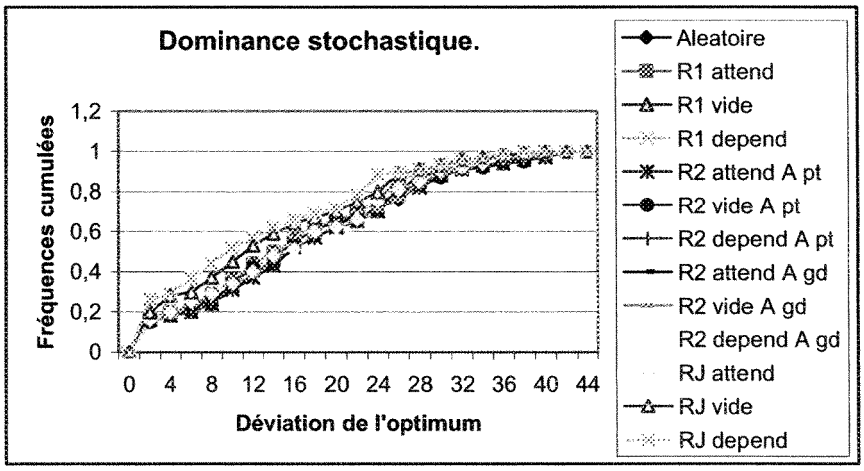

Figure 31 : Dominance stochastique pour les instances de 50 tâches

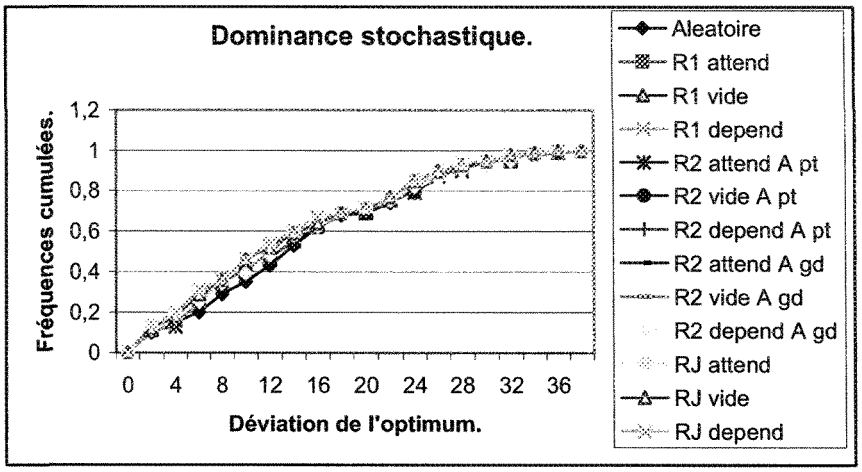

Figure 32 : Dominance stochastique pour les instances de 200 tâches

4.4 Modification de deux règles de priorité

Les deux premières règles R1 et R2, basées sur l'appariement des tâches, ont été développées pour prendre en compte les temps de transport des tâches. Dans ce dessein, nous avons adapté l'algorithme basé sur celui de Johnson, proposé par S. S. Panwalker [64]. Cet article traite d'un flowshop à deux machines et un convoyeur dont l'une des machines n'a pas d'espace de stockage. De plus, les temps de transport sont indépendants des tâches transportées. Panwalkar considère deux temps de transport constants T et T' dépendants de la direction du convoyeur. Pour l'adaptation de l'algorithme, une des stations de l'atelier a été considérée comme la station de départ d'un flowshop. En ce qui concerne les deux temps de transport constants, l'adaptation a consisté à former pour chaque couple de tâches, créé grâce à l'appariement, un couple de temps transport (T, T') correspondant aux temps de transport de chacune des tâches constituant le couple. Si par exemple, le couple (j_1, j_5) a été créé alors le couple de temps de transport (T, T') sera égal à (τ_{AB1}, τ_{BA5}) si $j_1 \in N_{AB}$ et $j_5 \in N_{BA}$. Dans cet article, le fait que la machine de départ ne possède pas

de zone de stockage implique que l'algorithme proposé minimise les temps de blocage de la dite machine, c'est-à-dire que le temps d'attente de la machine pour le convoyeur est limité. Or même si notre atelier n'a pas cette contrainte, l'application de l'algorithme à notre problème reste valable.

4.4.1 Extension de l'algorithme

L'extension fonctionne de la façon suivante : la station qui a le moins de tâches sur son stock d'entrée est celle qui est prise comme station de départ du flowshop. Posons, par exemple que B soit cette station. Le choix de la station ayant le moins de tâches a pour but de garder valide la remarque faite plus haut lors de l'appariement des tâches et la création d'espace vide. Comme précédemment les couples de tâches sont créés les « petits » avec les « petits » pour la règle R1 et « petit » avec « grand » pour R2. Ensuite pour chaque tâche j, la valeur $t'_{Bj} = \max\left\{t_{Bj}, \tau_{BAj} + \tau_{ABj'}\right\}$ (avec j' la tâche couplé avec j) est calculée. Les tâches sont alors rangées dans deux ensembles U et V tel que $U = \left\{j \middle| t'_{Bj} < t_{Aj}\right\}$ et $V = \left\{j \middle| t_{Aj} \leq t'_{Bj}\right\}$. La première séquence S_1 est construite en rangeant tout d'abord les tâches de l'ensemble U en ordre croissant selon t'_{Bj} puis en joutant les tâches de V rangés par ordre décroissant toujours selon t_{Aj}, le temps d'accomplissement M_1 est ensuite calculé. Finalement, tant que toutes les tâches n'ont pas été déplacées, des séquences de tâches résultant de permutation sont créées et l'ordonnancement-résultat est celui qui a le makespan le plus petit.

Voici en résumé cet algorithme :

Algorithme d'extension des règles R1 et R2

1. Sélectionner la station qui a le moins de tâches. Posons que B soit cette station
2. Créer les couples de tâches suivant la règle R1 ou R2

3. Pour chaque tâche j de N_{BA}, calculer $t'_{Bj} = \max\left\{t_{Bj}, \tau_{BAj} + \tau_{ABj'}\right\}$ avec $j' \in N_{AB}$ associé à j
4. Construire les ensembles U et V tel que $U = \left\{j \middle| t'_{Bj} < t_{Aj}\right\}$ et $V = \left\{j \middle| t_{Aj} \leq t'_{Bj}\right\}$
5. Construire la séquence S_1 en commençant par les tâches de U rangés de façon croissante selon t'_{Bj} puis toutes les tâches de V en ordre décroissant toujours selon t'_{Aj}
6. Ranger les tâches de la station A de façon à retrouver le couplage des tâches de A et B
7. Calculer M_1 le temps d'accomplissement total
8. Posons $cpt = 2$
9. Tant que $cpt \leq N_{BA}$ faire
   - 9.1. Soit r la première tâche de S_1
   - 9.2. Si $\tau_{BAr} + \tau'_{ABr'} \leq t_{Br}$ alors
     - 9.2.1. Créer la séquence S_{cpt} en déplaçant la tâche r à la $(cpt-1)^{ème}$ position
     - 9.2.2. Ranger les tâches de la station A de façon à retrouver le couplage des tâches de A et B
     - 9.2.3. Calculer M_{cpt}
   - 9.3. Sinon
     - 9.3.1. $M_{cpt} = \infty$
     - 9.3.2. $cpt = cpt + 1$
10. fin tant que
11. Soit S_k la séquence résultat telle que $M_k = \min\left(M_1, M_2, .., M_{N_{AB}}\right)$

Notons que cet algorithme a une complexité en $O(n^2)$.

4.4.2 Application sur un exemple

Cet algorithme a été appliqué sur l'exemple dont la description des tâches est donnée par le Tableau 16 suivant.

Tâches	j_1	j_2	j_3	j_4	j_5	j_6	j_7	j_8	j_9
Temps d'exécution t_{Aj}	3	5	1	6	7	2	-	3	7
Temps d'exécution t_{Bj}	6	2	2	6	5	4	5	4	2
Temps de transport τ_j	2	8	3	6	8	1	2	5	6

Tableau 16 : Constitution des tâches

Les tâches commençant sur la station A sont $N_{AB} = \{j_1, j_2, j_3, j_4, j_5\}$ et ceux commençant sur B, $N_{BA} = \{j_6, j_7, j_8, j_9\}$. Pour la règle R1, les couples suivants sont alors construits : (j_3, j_9), (j_1, j_6), (j_2, j_8), (j_4, j_7). La tâche j_5 est célibataire. La station, qui a le moins de tâches sur son stock d'entrée, est la station B. C'est donc cette station qui sera considérée comme celle de départ d'un flowshop.

Tout d'abord, les valeurs t'_{Bj} sont calculées pour les tâches j commençant sur la station B, $j \in N_{BA}$. Le Tableau 17 montre ces valeurs.

Tâches j	j_6	j_7	j_8	j_9
$\tau_{Bj} + \tau'_{Bj'}$	3	8	13	9
t'_{Bj}	4	8	13	9

Tableau 17 : Calcul de t'_{Bj}.

Il faut maintenant construire les ensembles U et V. Par exemple, pour la tâche j_6, comme $t_{A6} < t'_{B6}$, alors $j_6 \in V$. De la même façon, $U = \{\varnothing\}$ et $V = \{j_6, j_7, j_8, j_9\}$. Comme l'ensemble U est vide, pour construire la séquence S_1, il suffit de trier les

tâches de V en ordre de t_{Aj}, ce qui donne $S_1 = \{j_9, j_8, j_6, j_7\}$. Sur la station B, les tâches sont ordonnées suivant le couplage précédemment, soit : $\{j_3, j_2, j_1, j_4, j_5\}$. Le temps d'accomplissement M_1 de S_1 est 55. Maintenant, les séquences découlant de S_1 sont construites. Un compteur est alors initialisé à deux.

La tâche, à la position 2, est j_8. Or $t_{A8} < \tau_{BA8} + \tau'_{AB2}$, la séquence S_2 est alors la suivante : $S_2 = \{j_8, j_9, j_6, j_7\}$. Les tâches sur B sont ordonnées ainsi : $\{j_2, j_3, j_1, j_4, j_5\}$ et le temps d'accomplissement de cet ordonnancement est $M_2 = 59$.

Le compteur est incrémenté et est maintenant égal à trois. La tâche r est maintenant j_6. Comme $\tau_{BA6} + \tau'_{AB1} < t_{B6}$ alors $M_3 = \infty$.

Le compteur vaut 4 et la tâche r est j_7. Comme $t_{B7} < \tau_{BA7} + \tau'_{AB4}$ alors $S_4 = \{j_7, j_9, j_8, j_6\}$. L'ordonnancement des tâches de B donne $\{j_4, j_3, j_2, j_1\}$. Le temps d'accomplissement est $M_4 = 60$.

L'algorithme se termine alors car le compteur a la même valeur que le nombre de tâche dans N_{BA}.

Le Tableau 18 suivant récapitule les valeurs trouvées.

Numéro de séquence	Ordre des tâches sur les deux stations	Temps d'accomplissement
S_1	A : j_9, j_8, j_6, j_7 B : j_3, j_2, j_1, j_4, j_5	55
S_2	A : j_8, j_9, j_6, j_7 B : j_2, j_3, j_1, j_4, j_5	59
S_4	A : j_7, j_9, j_8, j_6 B : j_4, j_3, j_2, j_1	60

Tableau 18 : Récapitulatif des valeurs de l'exemple

L'extension de la règle R1 donne alors pour cet exemple les meilleurs séquences : pour la station A $\{j_9, j_8, j_6, j_7\}$ et pour B $\{j_3, j_2, j_1, j_4, j_5\}$ avec un temps d'accomplissement minimal M_1 égal à 55.

La règle étendue de R2 n'a pas été appliquée à cet exemple, car elle est similaire, nous laissons au lecteur, s'il le souhaite, d'effectuer cette application.

4.4.3 Résultats

Les règles R1 et R2 ont été modifiées, comme présentés précédemment, et testées sur les instances précédentes. Pour savoir si l'extension des règles améliore les performances de l'atelier, les résultats obtenus sont comparés avec ceux des règles non étendues. Les tableaux suivants (Tableau 19, Tableau 20 et Tableau 21) donnent le pourcentage des instances où le résultat de l'extension est strictement meilleur que la règle. Les résultats obtenus ont été séparés pour une meilleure visibilité suivant le comportement du convoyeur.

Le Tableau 19 montre les résultats de l'amélioration de la règle R1.

Nombre de tâches	R1 étendue attend	R1 étendue vide	R1 étendue dépend
10 tâches	44	24	41
20 tâches	55	27	55
50 tâches	68	15	68
200 tâches	84	20	84

Tableau 19 : Pourcentage où R1 étendue est strictement meilleure que R1

De même, le Tableau 20 montre l'extension de la règle R2.

Nombre de tâches	R2 A Pt étendue attend	R2 A Pt étendue vide	R1 A Pt étendue dépend	R2 A Gd étendue attend	R2 A Gd étendue vide	R1 A Gd étendue dépend
10 tâches	43	24	41	44	29	42
20 tâches	68	31	64	59	34	59
50 tâches	63	19	59	62	25	62
200 tâches	77	35	76	80	34	81

Tableau 20 : Pourcentage où R2 étendue est strictement meilleure que R2

Ces résultats montrent une réelle amélioration des règles de priorité présentées. Le pourcentage un peu plus élevé pour le comportement d'attente de convoyeur est tout à fait explicable par la configuration de l'algorithme repris de Panwalkar qui minimise les blocages de la station de départ, or l'attente du convoyeur minimise ceux–ci.

Comme le montre le Tableau 21 suivant, l'amélioration des règles de priorité est réelle, mais a pour conséquence une augmentation du temps d'exécution. Plus le nombre de tâches a traité est important, plus la différence de temps d'exécution est prédominante. Bien que ces temps soient faibles (millisecondes), c'est un critère à prendre en compte de lors de l'implantation de ces améliorations.

Nombre de tâches	Règles R1 et R2	Règles R1 et R2 améliorées	Augmentation (en %)
10 tâches.	6,75	7.59	12.35
20 tâches.	8.05	17.92	122.73
50 tâches.	12.99	64.79	398.75
200 tâches.	36.63	648.92	1671.56

Tableau 21 : Temps moyen d'exécution (millisecondes) des règles de départ et augmentation induite

Suite à la modification des règles R1 et R2, tentons de faire de même avec la règle de Jackson.

4.5 Modification de la règle de Jackson

De même que précédemment, la règle de Jackson a été enrichie afin de prendre en compte les temps de transport en nous basant sur l'article de Panwalkar [64]. Au départ, une adaptation de l'article de Mitten [58] a été effectuée, mais celle-ci n'a pas donné de résultats satisfaisants. Elle n'a donc pas été insérée ici.

La modification de l'algorithme de Jackson a simplement été effectuée en ajoutant le temps de transport, pour chaque tâche, au temps d'exécution de la station de départ. Puis l'algorithme de Jackson est appliqué normalement. Ainsi, le temps d'exécution de chaque tâche de la station de départ a été pondéré par son temps de transport.

Algorithme d'extension de la règle de Jackson

1. Pour toutes les tâches, si la tâche j a deux temps d'exécution non nuls alors
   1.10. Si la station de départ est la station A alors $t'_{Aj} = t_{Aj} + \tau_j$
   1.11. Sinon $t'_{Bj} = t_{Bj} + \tau_j$
2. Sinon la tâche ne doit être exécutée que sur une seule station et n'a donc pas de temps de transport, elle n'est alors pas modifiée

3. Appliquer l'algorithme de Jackson avec les temps d'exécution modifiés

Comme il est facile de le voir, cet algorithme est en $O(n \log n)$.

4.5.1 Illustration sur l'exemple précédent

Tout d'abord rappelons que les tâches commençants sur A sont $N_{AB} = \{j_1, j_2, j_3, j_4, j_5\}$ et ceux commençant sur B, $N_{BA} = \{j_6, j_7, j_8, j_9\}$. Le Tableau 22 récapitule les données de l'exemple et donne les valeurs modifiées des temps d'exécution.

Tâches j	j_1	j_2	j_3	j_4	j_5	j_6	j_7	j_8	j_9
Temps d'exécution t_{Aj}	3	5	1	6	7	2	-	3	7
Temps d'exécution t_{Bj}	6	2	2	6	5	4	5	4	2
Temps de transport τ_j	2	8	3	6	8	1	2	5	6
Temps d'exécution t'_{Aj}	5	13	4	12	15	-	-	-	-
Temps d'exécution t'_{Bj}	-	-	-	-	-	5	7	9	8

Tableau 22 : Constitution des tâches avec leurs temps d'exécution modifiés

Il reste maintenant à appliquer l'algorithme de Jackson sur ces tâches en prenant comme temps d'exécution t'_{Aj} et t_{Bj} pour les tâches $j \in N_{AB}$ et t_{Aj} et t'_{Bj} pour les tâches $j \in N_{BA}$. Cette application donne les séquences de tâches suivantes. j_1, j_5, j_4, j_3, j_2 pour la station A et toujours pour un ordonnancement alphanumérique j_6, j_7, j_8 et j_9 pour la station B.

4.5.2 Résultats

Pour savoir si la modification sur l'algorithme de Jackson est une réelle amélioration comme précédemment, les résultats obtenus sont comparés à ceux de la règle de Jackson. Le Tableau 23 donne le pourcentage des instances où le résultat de l'extension est strictement meilleur que celle de Jackson.

Nombre de tâches	RJ étendue attend	RJ étendue vide	RJ étendue dépend
10 tâches	11	14	10
20 tâches	5	10	4
50 tâches	0	3	2
200 tâches	0	0	0

Tableau 23 : Pourcentage où RJ étendue est strictement meilleure que RJ

Ces résultats tendent à montrer que la modification de la règle de Jackson n'est réellement une amélioration que dans le cas d'un atelier ayant peu de tâches à effectuer. Globalement, cette méthode d'ordonnancement ne peut donc remplacer avantageusement celle de Jackson.

Pour tenter de comprendre pourquoi notre intuition sur la prise en compte des temps de transport n'a pas été vérifiée, cette modification a été testée sur des instances particulières. Ces instances sont de trois types :

- Les temps de transport sont strictement inférieurs aux temps d'exécution, $\forall j, \tau_j < t_j$,
- Les temps d'exécution sont strictement inférieurs aux temps de transport, $\forall j, \tau_j > t_j$,
- Les temps de transport sont égaux aux temps d'exécution, $\forall j, \tau_j = t_j$.

Comme précédemment, les résultats trouvés sont comparés aux résultats de l'algorithme de Jackson sur les mêmes instances. Le Tableau 24 résume les résultats trouvés.

Les valeurs nulles de la première ligne du tableau s'expliquent par le fait que, comme nous l'avons démontré dans le chapitre précédent, lorsque pour toutes les tâches le temps de transport est strictement inférieur aux temps d'exécution, n'importe quelle permutation est optimale.

	RJ étendue attend	RJ étendue vide	RJ étendue dépend
$\forall j, \tau_j < t_j$	0	0	0
$\forall j, \tau_j > t_j$	6	14	6
$\forall j, \tau_j = t_j$	66	74	77

Tableau 24 : Comparaison pour des instances particulières

Lorsque les temps transport sont strictement supérieurs aux temps d'exécution alors la règle de Jackson étendue revient à ordonnancer les tâches dans le sens décroissant des temps d'exécution de la station qui n'est pas celle de départ de la tâche, car $\forall j, t_{Depj} + \tau_j > t_{Finj}$ où t_{Depj} marque le temps d'exécution de la station de départ et t_{Finj} celui de la station d'arrivée. Ce qui en regard des résultats, n'est pas meilleur que la règle de Jackson.

Concernant la dernière série d'instances, en revanche, l'algorithme étendu semble être meilleur que l'algorithme de Jackson. En effet, puisque les temps d'exécution sont égaux au temps de transport, alors l'algorithme étendu permet de discriminer les tâches suivant leurs temps d'exécution alors que, pour l'algorithme de Jackson, cela n'est pas possible. En effet, l'algorithme de Jackson sélectionne la tâche qui possède le temps d'exécution le plus petit. Puis selon la station auquel

appartient ce temps, la tâche est placée au début ou à la fin de la séquence de tâches. Or, comme les temps sont égaux, par défaut la tâche sera mise en fin d'ordonnancement.

Pour résumer, l'algorithme étendu est meilleur que l'algorithme de Jackson uniquement dans le cas d'atelier où les tâches ont des temps d'exécution et de transport égaux.

4.6 Conclusion

Pour résoudre le job shop à deux machines et un convoyeur, trois règles d'ordonnancement ont été proposées. Les deux premières, nommées R1 et R2, sont basées sur l'appariement des tâches pour former des couples dont chaque membre débute sa réalisation sur des stations différentes. La règle R1 apparie les tâches qui ont des temps d'exécution sur leur station de départ les plus courts. Et la règle R2 crée des couples de tâches dont un des membres a un temps d'exécution le plus court et l'autre le plus long. Une fois les couples créés pour les deux règles, l'algorithme de Johnson est appliqué sur la séquence qui a le moins de tâches. La troisième règle est l'application de l'algorithme de Jackson.

Les trois règles d'ordonnancement ont été testées sur les instances de problèmes-tests créées lors de l'étude du convoyeur. Les résultats obtenus ont montré que les règles R1 et R2 sont similaires et sont dominées de façon stochastique par la règle de Jackson. Puisque les règles proposées ne prenaient pas en compte les temps de transport, celles-ci ont été étendues en se basant sur l'article de Panwalkar [64]. Pour les règles R1 et R2, les couples de tâches sont créés de la même façon. Puis pour chaque tâche j, la valeur $t'_{Aj} = \max\left\{t_{Aj}, \tau_{ABj} + \tau'_{BAj'}\right\}$ (avec j' la tâche couplée avec j) est calculée. Les tâches sont alors rangées dans deux

ensembles U et V tel que $U = \left\{ j \middle| t'_{Aj} < t_{Bj} \right\}$ et $V = \left\{ j \middle| t_{Bj} \leq t'_{Aj} \right\}$. Une première séquence de tâches est créée puis tant que toutes les tâches n'ont pas été déplacées, des séquences de tâches résultant de la permutation de tâches sont créées et l'ordonnancement-résultat est celui qui a le temps d'accomplissement minimum. Les règles R1 et R2 modifiées pour prendre en compte les temps de transport ont montré une réelle amélioration des résultats, mais au prix d'une perte de rapidité d'exécution des règles.

La règle de Jackson a été modifiée en ajoutant au temps d'exécution de chaque tâche de sa station de départ, le temps de transport. Cet ajout n'est effectué que si la tâche possède deux temps d'exécution non nuls. Puis la règle de Jackson est appliquée normalement. Cette modification n'a été profitable que pour des problèmes avec peu de tâches. Pour étudier plus finement l'influence de la prise en compte des temps de transport dans la règle de Jackson, celle-ci a été testée sur des instances dont les tâches ont des caractéristiques particulières. Ces tâches possédaient soit des temps d'exécution strictement inférieurs ou, supérieurs aux temps de transport ou, enfin des temps identiques. Comme il a été montré précédemment, n'importe quelle permutation est optimale dans le cas de tâches ayant des temps d'exécution strictement supérieurs au temps de transport. Dans ce cas, la modification n'a pas d'intérêt. Concernant les tâches ayant des temps d'exécution strictement inférieurs aux temps de transport, la modification n'a pas montré une réelle amélioration, car elle revient à ordonnancer les tâches dans le sens décroissant des temps d'exécution de la station qui n'est pas celle de départ de la tâche. Toutefois, la modification est une réelle amélioration lorsque les temps d'exécution et de transport sont identiques, car elle permet contrairement à l'algorithme de Jackson de discriminer les tâches.

CHAPITRE 5

RÉSOLUTION PAR UNE MÉTAHEURISTIQUE :

LA RECHERCHE AVEC TABOUS

5.1 Introduction

Une métaheuristique, la recherche avec tabous, a été appliquée au problème de job shop à deux machines et un convoyeur. Comme nous l'avons présentée au Chapitre 2, cette méthode est une extension de la méthode de recherche locale, crée par Glover [27]. Elle est basée sur une analogie de la mémoire humaine.

Nous avions également mentionné que selon [9], la recherche avec tabous la plus efficace actuellement est celle de Nowicki et Smutnicki [63]. Or, concernant l'application à notre problème, nous n'avons pas employé cette recherche, car elle ne prend pas en compte les transports des tâches. Nous avons choisi d'appliquer la recherche avec tabous présentée dans [55]. Dans cet article, Mati *et al.* proposent une recherche avec tabous applicable aux problèmes de job shop multi-ressource et acceptant les blocages. Ce cadre très général est donc tout à fait applicable à notre problème de job shop à deux machines et un convoyeur. En effet, les spécificités de notre problème que sont les espaces de stockages et du convoyeur peuvent y être prises en compte.

Ce chapitre commence par la présentation de la notation des tâches dans le cadre de la métaheuristique, puis la recherche avec tabous est décrite précisément en débutant par la fonction objectif, différente de la minimisation du makespan, et par la construction grâce à une heuristique simple de la solution initiale. Finalement, la liste taboue et son comportement sont décrits et l'algorithme de recherche est proposé. Cet algorithme se basant essentiellement sur la méthode de résolution graphique étendue par Mati *et al.*. Cette méthode simplifiée pour l'atelier considéré, dans ce mémoire, est également présentée ainsi que sa complexité. Les résultats obtenus par la métaheuristique sont présentés et comparés avec la meilleure des règles de priorité proposée dans le chapitre précèdent.

5.2 Description d'une tâche dans le cadre de la recherche avec tabous

Cette méthode considère une opération comme une description détaillée des différentes phases d'une tâche. Ce qui permet d'inclure dans les opérations, les opérations d'attente correspondant aux tâches se trouvant dans les espaces de stockages et les opérations de transport. Cette description de tâches amène à redéfinir la notation d'une tâche dans le cadre de cette méthode. La tâche j_i est maintenant décrite de la façon suivante :

$$j_i = \left\{\left(S_A^{in},0\right),\left(A,p_{Ai}\right),\left(S_A^{out},0\right),\left(C,\Delta\right),\left(S_B^{in},0\right),\left(B,p_{Bi}\right),\left(S_B^{out},0\right)\right\}$$

Comme on peut l'observer, cette notation ajoute les opérations d'attente de la tâche lorsqu'elle se trouve dans les stocks des stations. Puisque l'atelier est constitué de deux stations ayant chacune deux espaces de stockage, une tâche a alors quatre opérations d'attente. Ces opérations sont considérées comme étant de temps d'exécution nuls. De plus, il est spécifié si l'espace de stockage est celui d'entrée ou de sortie de la station, en notant S_l^{in} l'espace d'entrée et S_l^{out} celui de sortie de la station l. L'opération de transport est, quant à elle, notée par la ressource C pour convoyeur et a un temps d'exécution de Δ unité de temps. L'ensemble de ressources est donc : $R = \{A, B, C, S_A^{in}, S_A^{out}, S_B^{in}, S_B^{out}\}$ où chaque ressource n'est présente qu'une seule fois, c'est-à-dire $\forall r \in R, Q_r = 1$, si Q_r désigne la quantité de ressource que possède l'atelier. Pour celui-ci, une tâche décrite sous cette forme aura toujours la propriété suivante : Une tâche constituée de n opérations aura k opérations de temps d'exécution non nuls et $(k+1)$ opérations d'attentes. De plus, une opération de temps d'exécution non nul sera toujours encadrée par deux opérations d'attentes.

Il est à noter que, puisque nous sommes dans un environnement multi-ressources, une opération O_{ij} dénote la $i^{ème}$ opération de la tâche j.

Une fois les tâches notées de cette manière, intéressons-nous à la fonction objectif de la recherche avec tabous qui permet de quantifier la qualité d'une solution.

5.3 Fonction objectif de la recherche avec tabous

Dans le cadre de cette recherche avec tabous, la fonction objectif est une maximisation de gains. Plus précisément, c'est la somme maximale des gains possible pour chaque séquence de tâches $\prod = \{\pi(1), \pi(2), .., \pi(N)\}$ où $\pi(i)$ corresponds à la tâche qui est en position i dans l'ordonnancement. Ce « gain » est vu comme l'avantage d'ordonnancer une tâche j_i avant une tâche j_j. Pour déterminer ce gain, un problème d'optimisation fictif est utilisé. En effet, le gain devient un temps de mise en course entre deux tâches j_i et j_j, définit comme suit : $S_{ij} = \sum_{k=1}^{n_i} p_{ki} + \sum_{k=1}^{n_j} p_{kj} - C_{\max ij}$.

$\sum_{k=1}^{n_i} p_{ki}$ est la somme des temps d'exécution des n_i opérations composant la tâche j_i (respectivement $\sum_{k=1}^{n_j} p_{kj}$ pour la tâche j_j). C_{maxij} désigne le temps d'accomplissement des tâches i et j, obtenu grâce à la méthode de résolution graphique d'un problème de job shop à deux tâches, présentée au Chapitre 2.

Ce calcul de gain peut être vu comme un problème a une machine unique dont les tâches ont des temps de mises en course dépendants de la séquence d'ordonnancement. La matrice S des gains S_{ij} est alors construite pour l'ensemble des tâches, à partir de cet instant le calcul de la valeur de la fonction objectif pour

une séquence $\prod$ est alors fort simple, car elle consiste à sommer les valeurs d'une ligne (ou d'une colonne, car la matrice est symétrique) de S : $Gain = \sum_{i=1}^{N-1} S_{\pi(i)\pi(i+1)}$.

Exemple de calcul de la fonction objectif Gain

Si par exemple, l'atelier doit exécuter 3 tâches telles que :

$$j_1 = \left\{\left(S_A^{in},0\right),(A,2),\left(S_A^{out},0\right),(C,4),\left(S_B^{in},0\right),(B,5),\left(S_B^{out},0\right)\right\}$$

$$j_2 = \left\{\left(S_B^{in},0\right),(B,5),\left(S_B^{out},0\right),(C,1),\left(S_A^{in},0\right),(A,2),\left(S_A^{out},0\right)\right\}$$

$$j_3 = \left\{(S_A^{in},0),(A,5),(S_A^{out},0),(C,10),(S_B^{in},0),(B,8),(S_B^{out},0)\right\}$$

Les tâches j_1 et j_3 doivent donc tout d'abord être exécutés sur la station A puis sur la station B et j_2, lui, doit être exécuté d'abord sur la station B puis sur la station A. Les sommes des temps d'exécution pour les tâches sont les suivantes :

- pour j_1 : $\sum_{k=1}^{7} p_{k1} = 0+2+0+4+0+5+0 = 11$,
- pour j_2 : $\sum_{k=1}^{7} p_{k2} = 0+5+0+1+0+2+0 = 8$,
- pour j_3 : $\sum_{k=1}^{7} p_{k3} = 0+5+0+10+0+8+0 = 23$.

De plus, le temps d'accomplissement de deux tâches, calculé grâce à la résolution de graphique sont donné par le Tableau 25 :

C_{maxij}	j_1	j_2	j_3
j_1	X	11	25
j_2	11	X	22
j_3	25	22	X

Tableau 25 : Temps d'accomplissement des tâches prises deux à deux

La matrice S construite est alors donnée par le Tableau 26.

S	j_1	j_2	j_3
j_1	X	8	9
j_2	8	X	9
j_3	9	9	X

Tableau 26 : Matrice S

Il est à noter que dans la matrice, le gain de tâche avec lui-même est considéré nul et est noté par un X. Pour connaître, par exemple, la valeur de la fonction objectif pour la séquence de tâches $\Pi = \{j_1, j_2, j_3\}$, il suffit de calculer la somme 8 + 9 = 17.

5.4 Création de la solution initiale

La solution initiale est construite grâce à une simple heuristique d'insertion et non pas choisie arbitrairement. Cette heuristique fonctionne de la façon suivante.

À partir de l'ensemble de toutes les tâches devant être exécutées par l'atelier, deux sous ensembles E_1 et E_2 sont créés. E_1 contient les tâches déjà examinées et E_2 ceux qui ne le sont pas encore. De plus, E_1 contient initialement les deux tâches donnants, grâce à la résolution graphique précédente, une tâche appelée combinée, de temps d'accomplissement minimum par rapport à toutes les autres paires de tâches.

Cette tâche combinée, notée j_{com}, est construite par découpage du chemin le plus court trouvé dans le plan formé par les deux tâches dont le temps d'accomplissement minimum est recherché. La construction de la tâche combinée sera expliquée par la suite plus en détail. Puis à chaque pas, une tâche de E_2 donnant le temps d'accomplissement minimum avec j_{com} est retirée de E_2 et mise dans E_1. La tâche j_{com} est alors mise à jour. L'heuristique d'insertion se termine lorsque E_2 est vide. L'ordre dans lequel les tâches ont été insérées dans l'ensemble E_1 constitue la solution initiale.

Si l'exemple précédent est repris, suivant le Tableau 25, les tâches permettant de construire une tâche combinée de temps d'accomplissement minimum sont j_1 et j_2, les deux ensembles E_1 et E_2 sont donc initialement remplis ainsi : $E_1 = \{j_1, j_2\}$ et $E_2 = \{j_3\}$.

La résolution graphique de j_1 et j_2 donne la tâche combinée j_{com} suivante :

$$j_{com} = \begin{Bmatrix} (S_A^{in} S_B^{in}, 0), (AB, 2), (S_A^{out}, 0), (CB, 3), (S_B^{out}, 0), \\ (C, 1), (S_B^{in}, 0), (BC, 1), (S_A^{in}, 0), (BA, 2), (S_A^{out}, 0), (B, 2), (S_B^{out}, 0) \end{Bmatrix}$$

Les tâches non examinées contenues dans E_2 sont alors combinées avec j_{com} et celle donnant le temps d'accomplissement minimum et transférée dans l'ensemble E_1 et j_{com} est mise à jour. Dans l'exemple, puisque E_2 ne contient qu'une seule tâche c'est alors j_3 qui sera transférée. La solution initiale construite par l'heuristique est l'ordre d'entrer dans l'ensemble E_1, c'est-à-dire j_1, j_2, j_3.

5.5 Caractérisation de la liste taboue et algorithme de recherche

Pour finir de spécifier la recherche avec tabous, le mouvement d'une solution à une autre doit être décrit ainsi que le fonctionnement de la liste taboue.

Un mouvement est défini comme l'échange d'une tâche j_i à la position $\pi(i)$ et d'une tâche j_j à la position $\pi(j)$ dans l'ordonnancement $\prod$. Par exemple, si $\prod = \{j_1, j_2, j_3, j_4\}$ un mouvement peut être l'échange de j_2 par j_4 ce qui donne : $\prod = \{j_1, j_4, j_3, j_2\}$.

La liste taboue contient les couples composés des positions des deux tâches qui ont été déplacées. L'entrée d'un mouvement dans la liste taboue va empêcher son réemploi pendant un certain nombre d'itération au cours de l'algorithme. Ce nombre d'itérations a été choisi de taille variable. Cette taille est sélectionnée aléatoirement dans l'intervalle $\left[\frac{n}{2}, n\right]$ où n est le nombre de tâches de l'atelier. De plus, cette modification de la taille de la liste survient toutes les $2n$ itérations de l'algorithme. Il est à noter que cette liste est circulaire c'est-à-dire que l'insertion d'un nouvel élément tabou dans la liste pleine implique que le plus ancien élément tabou soit retiré. Un mouvement se trouvant dans la liste tabou peut s'avérer être le meilleur mouvement à faire à un instant donné. Le critère d'aspiration, permettant de révoquer un élément tabou, retire l'élément de la liste lorsque le mouvement donne la meilleure solution.

Pour finir la description de l'algorithme, celui-ci se termine après 100 itérations. L'algorithme de recherche avec tabous que nous avons utilisé est comme suit:

Algorithme de recherche avec tabous

1. Création de la solution initiale grâce à l'heuristique d'insertion. Soit $\prod = \{j_{\pi(1)}, j_{\pi(2)}, \ldots, j_{\pi(n)}\}$ cette solution. Posons $C^*_{\max}$ le meilleur makespan et initialisons-le à $C^*_{\max} = \infty$
2. Tant que le nombre d'itérations n'est pas atteint, faire
   2.1. Construire la tâche combinée j_{com} à partir de la séquence $\prod$ grâce à l'algorithme vorace. Soit C_{max} son makespan

2.2. Si $C_{max} < C^*_{max}$ alors
  2.2.1. $C^*_{max} \leftarrow C_{max}$
  2.2.2. Garder en mémoire la tâche combinée j_{com}
2.3. Trouver le meilleur mouvement grâce à la fonction objectif et au critère d'aspiration
2.4. $\prod$ devient alors la séquence modifiée par l'application du mouvement
2.5. Ajouter le mouvement effectué à la liste taboue
2.6. Ajuster la taille de liste taboue si le nombre d'itérations est égal à $2nk$ (n nombre de tâches et k entier)

La solution obtenue correspond à l'ordonnancement de la tâche combinée j_{com} et son temps d'accomplissement est alors C^*_{max}. La construction de cette tâche combinée est réalisée grâce à un algorithme vorace. Il fonctionne de la façon suivante. Initialement, la tâche combinée j_{com} est constituée de la première tâche de la séquence d'entrée. Puis pour toutes les tâches de l'ensemble $\prod$, une tâche j' combinée est construite, qui est le résultat de la résolution graphique étendue de la tâche combinée j_{com} avec une tâche de $\prod$. Cette résolution sera vue plus précisément dans le paragraphe suivant. La sélection de la tâche se faisant suivant l'ordre de la séquence. Finalement j_{com} est mise à jour avec la tâche j'.

5.6 Méthode de résolution graphique

L'algorithme vorace est basé sur l'approche géométrique de Mati *et al.* [55]. Cette approche étend la méthode de résolution graphique d'un job shop à deux tâches (Akers [2], Brucker [12]), à la résolution d'un job shop à n tâches. Elle n'est qu'une version simplifiée de celle de Mati *et al.* qui, à cause de leur prise en compte des blocages de l'atelier, a mis en place une recherche des situations de blocages et une construction différente du graphe. Pour pouvoir appliquer la méthode étendue à

notre job shop, celui-ci doit suivre la contrainte suivante. Cette contrainte est comme suit :

Apres l'exécution de l'opération O_{ki} de la tâche J_i, toutes les ressources de R_{ik} (ressource pour l'opération O_{ki}) sont bloquées par la tâche J_i jusqu'à ce que les ressources dont on a besoin pour l'opération O_{k+1i} suivante deviennent disponibles. A cet instant, $\max\left\{0, B_{ki}^r - B_{k+1,i}^r\right\}$ *unités de chaque ressource r qui ne sont nécessaire pour l'opération O_{k+1i} deviennent disponibles et les autres ressources restent bloqués pour l'opération O_{k+1i}.*

Au premier abord, on peut se questionner sur la validité de cette contrainte pour les espaces de stockage de taille illimitée de l'atelier. Or, celle-ci est parfaitement applicable, car un espace de taille illimité peut être considéré comme un espace composé de multiple sous espace ne pouvant contenir qu'une seule tâche.

Détaillons maintenant le fonctionnement de la méthode de résolution graphique étendue. Similairement à la méthode non étendue, elle prend deux tâches et construit un plan formé par leurs opérations. Mais, comme l'a montré Mati *et al.*, l'introduction de la contrainte, présentée ci-dessus, entraîne qu'un nœud peut avoir plus de deux successeurs. En effet, tous les nœuds Nord-Ouest et Sud-Est des obstacles pouvant être atteints depuis le nœud v_i en partant diagonalement puis en allant verticalement, ou horizontalement, sont alors les successeurs de v_i. Plus précisément comme le montre la Figure 33 suivante, si l'arc entre v_i et l'obstacle est strictement diagonal alors les nœuds Nord-Ouest et Sud-Est font partis des successeurs de v_i. En revanche, si l'arc est au départ diagonal puis horizontal alors seul le nœud Sud-Est de l'obstacle touché fait parti des successeurs. De même, si l'arc est diagonal puis vertical alors seul le nœud Nord-Ouest est successeur.

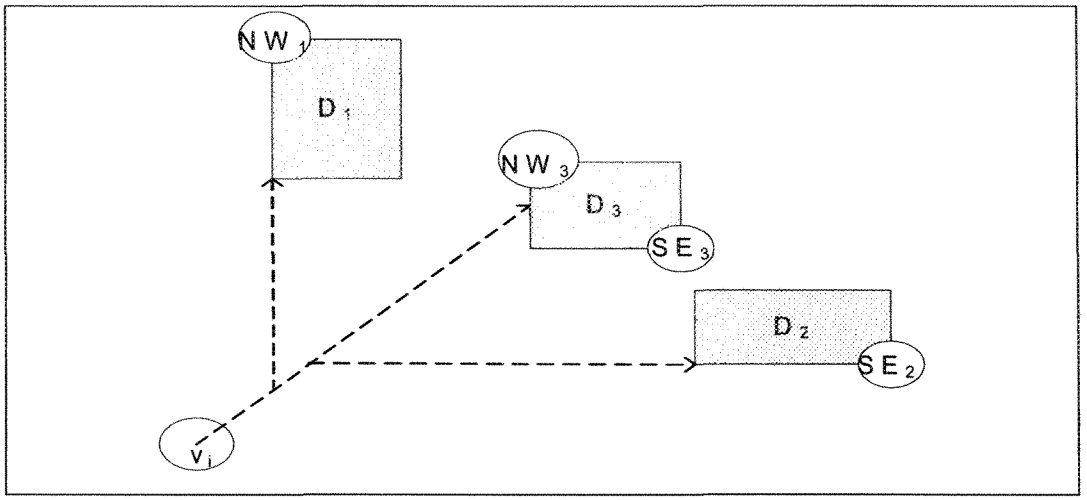

Figure 33 : Successeurs du noeud v_i

Comme dans la méthode non étendue, si lors de la recherche des successeurs, la frontière Nord ou Est du plan est touché, alors F fait parti des successeurs du nœud considéré. De plus, la longueur des arcs est calculée de la même manière que dans la résolution non étendue. Comme on le voit sur la Figure 34, la longueur entre un nœud $v_i = (x_i, y_i)$ et un coin Nord-Ouest $v_{jNW} = (x_{jNW}, y_{jNW})$ est égale à $d(v_i, vj_{NW}) = y_{jNW} - y_i$ et pour un coin Sud-Est $v_{jSE} = (x_{jSE}, y_{jSE})$, $d(v_i, v_{jSE}) = x_{jSE} - x_i$.

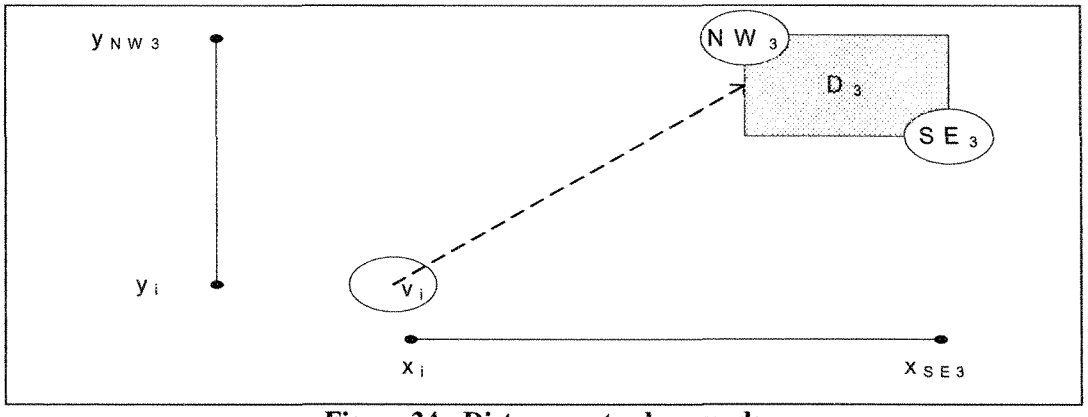

Figure 34 : Distance entre les nœuds

Une fois le graphe constitué de tous les nœuds, trouvés dans le plan créé, la recherche du plus court chemin dans ce graphe est effectuée. Il reste alors à construire une tâche combinée. Pour cela, l'intervalle de temps $[0, C_{\max}]$, où C_{max} le makespan correspond à la valeur du chemin le plus court, est décomposé en sous intervalles $[t_0, t_1],...,[t_{u-1}, t_u]$ telle qu'aucune opération ne commence ou ne se termine dans cet intervalle. Autrement dit, toutes les dates de début ou de fin des opérations doivent appartenir à l'ensemble $\{t_0, t_1, .., t_{u-1}, t_u\}$. La tâche combinée sera donc constituée de u opérations. Chaque opération O_{kj} avec $1 \leq k \leq u$ de la tâche combinée requière alors $t_k - t_{k-1}$ unités de temps, et a pour ressource, toutes les ressources des opérations des deux opérations qui sont exécutées dans l'intervalle $[t_{k-1}, t_k]$.

Pour revenir à l'exemple, la Figure 35 présente le plan formé dans le cadre de la résolution graphique étendue par les tâches j_1 et j_2.

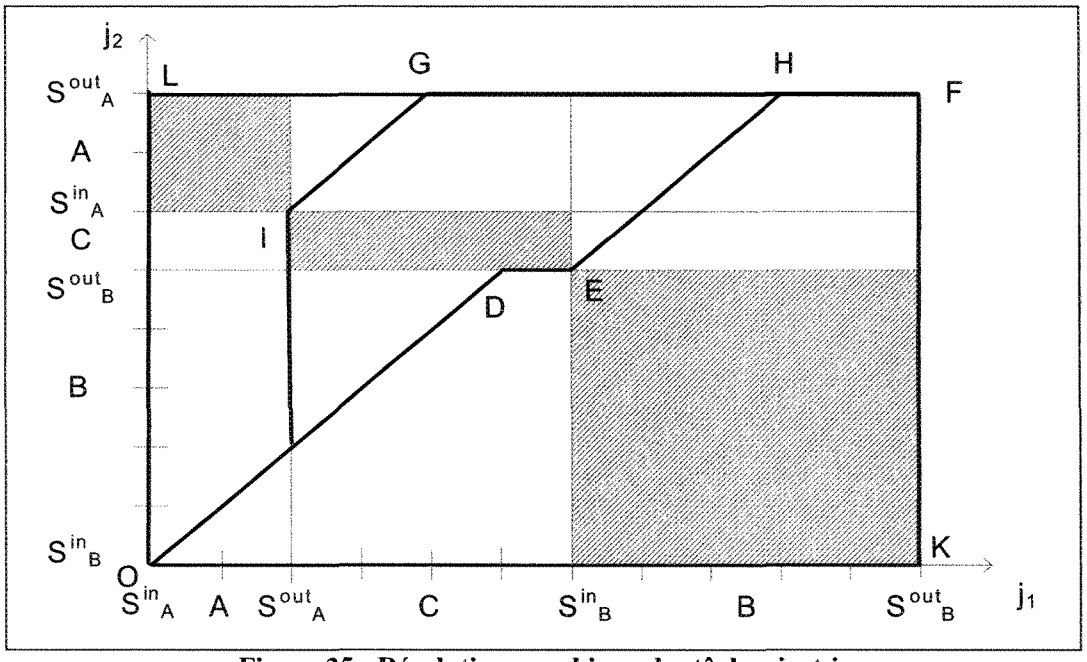

Figure 35 : Résolution graphique des tâches j_1 et j_2

Le chemin le plus court est ODEHF avec un temps d'accomplissement de 11 unités de temps. L'intervalle [0, 11] est donc décomposé en 11 sous intervalles. Ces sous intervalles sont les suivants : $[0,0]$, $[0,2]$, $[2,2]$, $[2,5]$, $[5,5]$, $[5,6]$, $[6,6]$, $[6,9]$, $[9,9]$, $[9,11]$, $[11,11]$. Ceci permet de construire la tâche combinée suivante :

$$j_{com} = \left\{ \begin{array}{l} (S_A^{in} S_B^{in}, 0), (AB, 2), (S_A^{out}, 0), (CB, 3), (S_B^{out}, 0), \\ (C, 1), (S_B^{in}, 0), (BC, 1), (S_A^{in}, 0), (BA, 2), (S_A^{out}, 0), (B, 2), (S_B^{out}, 0) \end{array} \right\}$$

Notons une particularité lors de la construction d'une tâche combinée. Les opérations d'attente ne se retrouvent jamais dans une opération combinée avec d'autres ressources, car leur temps d'exécution est nul. De plus, la propriété des tâches est toujours valide pour une tâche combinée, c'est-à-dire que chaque opération de temps d'exécution non nul sera toujours encadrée par deux opérations de temps d'exécution nul, soit deux opérations d'attente. Ceci garantit le non-blocage de l'atelier.

Pour résumer, l'algorithme vorace dans son ensemble pour une séquence Π donnée, est comme suit.

Algorithme vorace

2. Posons $j_{com} \leftarrow j_{\pi(1)}$.
3. Pour $i = 2$ à n faire (avec n le nombre de tâches de Π)
   - 3.1. Résoudre le problème à deux tâches j_{com} et $j_{\pi(i)}$ en utilisant la résolution géométrique étendue
   - 3.2. Construire la tâche j' combinant j_{com} et $j_{\pi(i)}$ suivant l'ordonnancement optimum
   - 3.3. Posons $j_{com} \leftarrow j'$
4. Extraire la séquence σ_r d'opérations pour chaque ressource r de la tâche combinée final j_{com}

La solution trouvée par la recherche taboue est donnée par la séquence des opérations σ_r.

5.7 Complexité de la méthode géométrique étendue

Comme il a été vu dans le Chapitre 2, Brucker [12] a montré que le graphe issu de la résolution graphique peut être construit en *O(r log r)* avec *r* le nombre d'obstacles se trouvant dans le graphe. De plus, le chemin le plus court dans le graphe acyclique est obtenu en *O(r)*. Le problème d'atelier à deux tâches peut donc être résolu en *O(r log r)*.

Lors de la construction du graphe G = (V, E) de deux tâches j_1 et j_2 dans le cadre de la méthode étendue et de l'atelier proposé, il y a au plus $k_1 \times k_2$ obstacles créés, avec $n_1 = k_1 + k_1 + 1$, $n_2 = k_2 + k_2 + 1$ où k_i est le nombre d'opérations de la tâche j_i. Il est à noter que ces opérations ne sont pas des opérations d'attentes. En effet, les opérations d'attentes, du fait de la taille illimitée des espaces de stockage, ne peuvent être bloquées. Un nœud a donc au plus $k_1 + k_2$ successeurs. La construction du graphe prendra alors au maximum $O\left(k_1 \times k_2 \times (k_1 + k_2)\right)$. La recherche du plus court chemin prendra un temps $O = \left(|E| + |V|\right)$ où $|V| = k_1 \times k_2$ et $|E| = |V| \times (k_1 + k_2)$ dans le pire cas. La méthode étendue pour deux tâches a donc une complexité de $O\left(k_1 \times k_2 \times (k_1 + k_2)\right)$.

5.8 Résultats

La recherche avec tabous proposée a été testée sur les mêmes instances de problèmes tests que celles utilisées pour les règles de priorité. Ces instances de 10, 20, 50 et 200 tâches ont été générés d'une manière aléatoire à partir d'une distribution uniforme. Les temps d'exécution et de transport des tâches sont répartis

uniformément dans [0, 50]. Le Tableau 27 montre ces résultats. Chaque instance a été résolue 10 fois par la recherche avec tabous. La meilleure solution, pour chaque instance, a été comparée avec la solution de la meilleure règle de priorité, c'est-à-dire avec la règle de Jackson, RJ. La notation des temps est la suivante, *min : secondes : millisecondes*.

La déviation moyenne de la recherche avec tabous et son écart-type sont plus faibles que ceux de la règle de priorité. La métaheuristique est donc en moyenne meilleure que la règle de Jackson. Mais elle atteint moins souvent la borne inférieure et sa déviation minimale est plus importante. Toujours dans la même optique, elle obtient plus souvent la meilleure solution, mais aussi la pire. De même, le temps moyen de calcul de la métaheuristique, contrairement à celui de la règle de priorité qui est négligeable, est important.

Notons que les instances de 200 tâches ne sont pas présentes dans le tableau de résultats. En effet, bien que les métaheuristiques soient faites pour traiter les gros problèmes, dans notre cas, cela n'a pas été possible. Cela n'est pas dû à l'algorithme lui-même, mais à la résolution graphique. En effet, Mati *et al.* [55] traite d'ateliers avec blocage ce qui leur permet grâce à l'introduction de trois matrices P_x, P_y et P_{xy} de limiter le nombre de nœuds du graphe. Or notre atelier, n'ayant pas de blocage, tous les coins Nord-Ouest et Sud-Est des obstacles du plan sont pris en compte. De plus, les auteurs dans leur article ont testé leur méthode uniquement sur des problèmes ayant au maximum 10 tâches à exécuter, ce qui ne permet pas de savoir si celle-ci est applicable sur des ateliers possédant un grand nombre de tâches.

	Problémes tests à 10 travaux.				Problémes tests à 20 travaux.				Problémes tests à 50 travaux.			
Heuristique	**Tabou**	**RJ attend**	**RJ vide**	**RJ depend**	**Tabou**	**RJ attend**	**RJ vide**	**RJ depend**	**Tabou**	**RJ attend**	**RJ vide**	**RJ depend**
Déviation moyenne par rapport à la borne inférieure (en %)	4,42	10,95	13,44	10,78	5,27	9,98	11,84	9,81	5,82	11,94	12,96	11,94
Ecart type de la déviation	4,11	10,39	11,55	10,34	3,56	10,23	10,94	10,16	2,85	10,34	10,57	10,34
Déviation maximale par rapport à la borne inferieure	16,61	39,79	37,79	39,79	13,41	34,02	38,36	34,02	15,67	36,98	38,47	36,98
Déviation minimale par rapport à la borne inferieure	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,15	0,00	0,00	0,00
Nombre de fois que la borne est atteinte	22	26	23	28	6	33	25	33	0	20	18	20
Nombre de fois que l'heuristique obtient la meilleure solution	78	34	25	36	59	46	25	46	68	33	18	33
Nombre de fois que l'heuristique obtient la pire solution	23	24	75	20	36	15	61	11	30	6	64	4
Temps moyen de calcul	0:18:36	00:00:06	00:00:10	00:00:04	1:26:38	00:00:07	00:00:06	00:00:10	18:16:13	00:00:11	00:00:11	00:00:11

Tableau 27 : Résultats de la recherche avec tabous

5.9 Conclusion

La recherche avec tabous présentée pour résoudre le problème de job shop à deux stations et un convoyeur est basée sur l'article de Mati *et al.* [55]. Dans ce cadre, la description d'une tâche a été modifiée pour prendre en compte les opérations d'attentes survenant lorsqu'une tâche se trouve dans l'un des espaces de stockage de l'atelier. Puisque l'atelier considéré ne peut se bloquer, les tâches auront comme caractéristique que toutes les opérations d'exécution sur une station, ou de transport sont toujours placées entre deux opérations d'attente.

La fonction objectif n'est pas celle de minimisation du problème, mais de maximisation du gain d'une séquence de tâches. En effet, une solution est une séquence de tâches. Ce gain est vu comme l'avantage de placer dans la séquence une tâche avant un autre. La solution initiale est créée grâce à une heuristique d'insertion. Lors de la recherche, un mouvement est la permutation de deux tâches dans une solution. Une fois le mouvement effectué, la position des deux tâches déplacées est placée dans la liste taboue. Cette liste est circulaire et sa taille varie entre $\left[\frac{n}{2}, n\right]$ toutes les $2n$ itérations de la recherche (n est le nombre de tâches que doit exécuter l'atelier). Enfin, le critère d'aspiration est la révocation d'un mouvement tabou lorsque celui-ci donne la meilleure solution.

La particularité de cette recherche est qu'elle est basée sur la méthode de résolution graphique proposée par Mati *et al.*. Cette méthode généralise à n tâches la résolution graphique de deux taches présentée par Akers [2] et Brucker [12]. Cette méthode a une complexité de $O\left(k_1 \times k_2 \times \left(k_1 + k_2\right)\right)$ ou k_i est le nombre d'opérations de la tâche i.

La recherche avec tabous ainsi créée a été testée sur les mêmes instances de problèmes que celles générées pour les règles de priorité. Une fois les résultats obtenus, ils ont été comparés à ceux de la meilleure règle de priorité c'est-à-dire la règle de Jackson. La recherche avec tabous a fourni des résultats attendus de la part d'une métaheuristique, mais n'a pu traiter les instances de 200 tâches. Ceci n'est pas le fait de la métaheuristique qui est faite pour résoudre les « gros » problèmes, mais de la méthode de résolution graphique étendue employée. En effet, puisque notre atelier ne peut bloquer, il a été possible de simplifier la méthode de Mati *et al.* [55] mais la contrepartie a été de ne pouvoir diminuer le nombre de nœuds des graphes créés.

CONCLUSION

Dans ce mémoire, nous avons étudié le problème de job shop à deux machines, muni d'un convoyeur. Le critère d'évaluation choisi est celui du temps d'accomplissement total, communément appelé makespan. L'importance de cette étude n'est pas à démontrer, car la réalité industrielle ne permet pas de négliger les opérations de transport. L'introduction de convoyeurs au problème de job shop amène, de facto, d'autres modèles qui sont liés à la présence et à la caractéristique de lieux de stockage. Nous avons considéré dans notre étude, le modèle où il n'y a pas de limite de stockage. Cela peut sembler un peu simpliste, mais il est de bon augure de commencer de nouvelles études par des cas plus simples afin d'être mieux armés pour des cas plus complexes. De plus, même avec ces implications, il s'est avéré que notre problème est étonnamment NP-difficile. Ceci nous a amenés à nous tourner vers l'approche heuristique pour résoudre ce problème.

La méthodologie que nous avons suivie pour aborder notre problématique est la suivante. Dans un premier temps, nous avons présenté, au Chapitre 1, la théorie de l'ordonnancement. Ce chapitre expose la modélisation des problèmes d'ordonnancement et les différents critères d'optimisation. Ensuite, nous nous sommes intéressés à la complexité des problèmes, en l'occurrence la classe P (contenant les problèmes dits « faciles ») et la classe NPC (contenant les problèmes dits « difficiles »). La présentation de la complexité des problèmes d'ordonnancement nous a amenés à présenter leurs différentes méthodes de résolution. En effet, en exhibant une solution polynomiale, le problème est classé comme étant facile. Dans ce cas, le défi est de trouver de nouvelles solutions de meilleure complexité algorithmique. En revanche, si le problème est difficile, deux approches sont utilisées : les méthodes exactes et les méthodes heuristiques. La première approche génère une solution optimale, mais avec des temps de calcul généralement élevés. Les méthodes heuristiques génèrent, quant à elles, des

solutions en des temps raisonnables, mais ces solutions ne sont qu'approchées. Le défi dans ce cas est de concevoir des algorithmes améliorant la qualité de la solution obtenue. Notons qu'on distingue deux types d'approche heuristique : l'une est constructive et l'autre est itérative. L'approche constructive génère, à l'aide de simples règles et de manière progressive, une seule solution. En revanche, l'approche itérative (dont font partie les métaheuristiques) génère autant de solutions que l'on désire. Même si la performance de ces méthodes se fait généralement d'une manière expérimentale, elles offrent en revanche une capacité d'adaptation permettant la formulation de contraintes diverses afin de mieux représenter une large gamme de situations.

Suite à cela, dans le Chapitre 2, nous avons décrit le problème de job shop dans un cadre général. Le job shop est modélisé généralement de deux façons : la modélisation linéaire mixte et le graphe disjonctif. Ensuite, nous avons résumé les principales complexités de ce type de problème et décris les méthodes de résolution usuelles que nous avons appliquées, par la suite, à notre problème : l'algorithme de Jackson, la résolution graphique à deux travaux et la recherche avec tabous. Nous avons également introduit les opérations de transport et fait un état de l'art considérant les opérations de transport dans les problèmes de job shop. Pour terminer, nous avons exposé nos objectifs de recherche.

Le job shop, que nous avons étudié, est constitué de deux stations, elles-mêmes composées d'une machine et de deux stocks : un d'entrée et un de sortie, de taille illimitée. Un convoyeur est chargé de transporter les tâches semi-finies d'une station à l'autre. Dans le Chapitre 3, une fois la description du fonctionnement de ce job shop effectué et sa complexité donnée, nous avons exhibé ces propriétés. Et nous nous sommes intéressés plus particulièrement au convoyeur et à sa politique de (dé)placement. Une simulation a été effectuée par un programme Java sur des

instances de tailles n = 10, 20, 50 et 200 générées de manière aléatoire sur une distribution uniforme. Les temps d'exécution et de transport de ces instances sont compris entre 0 et 50 unités de temps. Grâce à cette simulation, il semble que la meilleure position initiale du convoyeur est le stock de sortie de la station possédant le plus de tâches. De plus, le comportement le plus intéressant du convoyeur est celui de l'attente systématique des taches lorsque celles-ci sont en cours d'exécution sur une machine, et que son stock de sortie est vide. Nous avons également proposé une borne inférieure ce qui nous a permis, par la suite, de comparer nos résultats.

Etant donné que notre problème est NP-difficile, nous avons, pour sa résolution, au Chapitre 4, conçu trois heuristiques basées sur des règles de priorité. Les deux premières règles construisent des couples de tâches suivant leurs temps d'exécution sur leurs machines de départ. Puis elles appliquent l'algorithme de Johnson sur la station qui a le moins de travaux. Et finalement, réordonne les tâches placées sur le stock d'entrée de l'autre station, de façon à retrouver l'appariement créé. La première règle crée des couples dont les temps d'exécution sont minimums alors que la seconde apparie une tâche qui a un temps minimum avec une tâche qui a un temps maximum. La troisième règle applique l'algorithme de Jackson. Toujours grâce à une simulation sur les instances de problèmes-tests déjà utilisées pour l'étude du convoyeur, les résultats ont montré que même lorsque les règles dont modifiées pour prendre en compte les temps de transport ; c'est l'algorithme de Jackson qui donne les meilleurs résultats.

Pour terminer notre étude, au Chapitre 5, nous avons utilisé une recherche avec tabous pour résoudre notre problème de job shop. Cette méthode est une application de la recherche avec tabous de Mati *et al.* [55], elle-même basée sur une généralisation à n tâches de la résolution graphique du job shop à deux tâches de Akers [2] et Brucker [12]. Nous avons choisi cette approche, car elle prend en

compte les opérations de transport. L'application à notre problème à deux stations et un convoyeur a permis une simplification de l'algorithme. Mais cela a eu pour effet de ne pouvoir traiter les instances de 200 tâches. Cette limitation n'est pas due à la métaheuristique, faite pour traiter les « gros » problèmes, mais à la résolution graphique utilisée qui ne permet pas de diminuer le nombre de nœuds des graphes créés.

L'objectif général de ce mémoire, qui était de proposer des méthodes de résolution approchées, a été atteint à l'aide d'objectifs spécifiques. Le premier de ces objectifs était de proposées des règles de priorité, et plus particulièrement de s'intéresser dans leurs construction à l'influence des temps de transport. Cet objectif a été atteint, la règle de Jackson a été montrée la plus performante bien qu'elle ne prenne pas en compte les temps de transport. Le second de ces objectifs a été de modifier une méthode de recherche avec tabous générique. Cette application n'a pas été très performante car elle n'a pas permis de traiter un grand nombre de tâches. Une suite à cette recherche serait de reprendre les travaux de Hunrik et Kunst [34]. En effet, ces auteurs ont proposé également une recherche avec tabous prenant en compte les temps de transport, mais qui passe par la modélisation du job shop sous forme d'un graphe disjonctif. L'application de cette recherche avec tabous au problème à deux stations et un convoyeur permettrait de comparer les résultats et donc les deux méthodes.

D'un point de vue personnel, ce travail m'a permis de découvrir le travail de recherche et la discipline de l'ordonnancement. J'ai pu acquérir des méthodes de travail et des connaissances, qui me seront certainement utiles par la suite. Cette expérience enrichissante m'a donné envie de poursuivre dans cette voie.

Bibliographie

1. Adams J., Balas E., and Zawack D., (1988): The Shifting Bottleneck Procedure for Job Shop Scheduling, Management Science, 34(3): pp.391.
2. Akers S.B., (1956): A graphical approach to production scheduling problems, Operations Research, 4: pp.244-245.
3. Baker K.R., (1974): *Introduction to sequencing and scheduling.* New York: Wiley, 305 p.
4. Balas E. and Vazacopoulos A., (1998): Guided Local Search with Shifting Bottleneck for Job Shop Scheduling, Management Science, 44(2): pp.262-275.
5. Bécart M., Lacomme P., Moukrim A., and Tchernev N. (2004), Proposition de résolution exacte d'un MILP pour les systémes flexibles de production du type job-shop avec transport. in *5e conference francophone de Modelisation et simulation MOSIM*, Nantes.
6. Bellman R., (1957): *Dynamic programming.* Princeton, New Jersey: Princeton university press .
7. Bellman R. and Dreyfus S.E., (1962): *Applied dynamic programmming.* Princeton, New Jersey: Princeton university press .
8. Blazewicz J., Domschke W., and Pesch E., (1996): The job shop scheduling problem: Conventional and new solution techniques, European Journal of Operational Research, 93(1): pp.1-33.
9. Blazewicz J., Ecker K.H., Pesch E., Schmidt G., and Weglarz J., (1996): *Scheduling computer and manufacturing processes,* 2nd edition ed. Springer-Verlag, 485 p.
10. Blazewicz J., Ecker K.H., Schmidt G., and Weglarz J., (1996): *Scheduling in computer and manufacturing systems.* Springer-Verlag .
11. Bovet D. and Crescenzi P., (1994): *Introduction to the Theory of Complexity.* New York: PrenticeHall .
12. Brucker P., (1988): An efficient algorithm for the job-shop problem with two jobs, Computing, 40(4): pp.353-359.
13. Brucker P., (1994): A polynomial algorithm for the two machine job-shop scheduling problem with a fixed number of jobs, OR Spectrum, 16(1): pp.5-7.
14. Brucker P., Jurisch B., and Sievers B., (1994): A branch and bound algorithm for the job-shop scheduling problem, Discrete Applied Mathematics, 49(1-3): pp.107-127.
15. Brucker P. and Knust S., (2002): Lower bounds for scheduling a single robot in a job-shop environment, Annals of Operations Research, 115(1): pp.147.
16. Brucker P., Kravchenko S.A., and Sotskov Y.N., (1996): *Preemptive jobshop scheduling problems with a fixed number of jobs*, Osnabrücker schriften zur mathematik, Reihe P, Heft 186.

17. Cook S. (1971), The complexity of theorem-proving procedures. in *3rd annual symposium on theory of computing*.
18. Dawande M., Geismar H.N., Sethi S.P., and Sriskandarajah C., (2005): Sequencing and Scheduling in Robotic Cells: Recent Developments, Journal of Scheduling, 8(5): pp.387-426.
19. De rumeur J., (1994), Masson: Réseaux de préprocesseurs.
20. Dell'amico M. and Trubian M., (1993): Applying tabu search to the job-shop scheduling problem, Annals of Operations Research, 41(3): pp.231.
21. Dorigo M., Maniezzo V., and Colorni A., (1996): The ant system: Optimization by a colony of cooperating agents, IEEE Transactions On Systems, Man, And Cybernetics. Part B, 26(1): pp.29-41.
22. Eglese R.W., (1990): Simulated annealing : A tool for operational research, European Journal of Operational Research, 46(3): pp.271-281.
23. Esquirol P. and Lopez P., (1999): *L'ordonnancement.* .
24. French S., (1982): *Sequencing and scheduling : an introduction to the mathematics of the jobshop,* John wiley and sons ed. Ellis horwood limited, 243 p.
25. Garey M.R. and Johnson D.S., (1979): *Computers and intractibility, a guide to the theory of NP-completeness*. New Jersey: Murray hill.
26. Garey M.R., Johnson D.S., and sethi R., (1976): The complexity of flowshop and job shop scheduling, Mathematics of operationsl research, 1(2): pp.117-129.
27. Glover F., Taillard E., and Taillard E., (1993): A user's guide to tabu search, Annals of Operations Research, 41(1): pp.1-28.
28. Gonzalez K., Feofilo|Sahni,K,Sartaj, (1978): Flowshop and Jobshop Schedules: Complexity and Approximation., Operations research, 26(1): pp.36.
29. GOTHA, (1993): Les problémes d'ordonnancement, RAIRO-Recherche Operationnelle: pp.77-150.
30. Graham R., Lawler E., Lenstra J.K., and Rinnooy Kan A., (1979): Optimization and approximation in deterministic sequenecing and scheduling theory : a survey, Annals of discrete mathematics, 5: pp.287-326.
31. Gultekin H., Akturk M.S., and Karasan O.E., (2006): Cyclic scheduling of a 2-machine robotic cell with tooling constraints, European Journal of Operational Research, 174(2): pp.777-796.
32. Hall N.G. and Sriskandarajah C., (1996): A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process, Operations Research, 44(3): pp.510-525.

33. Hurink J. and Knust S., (2002): A tabu search algorithm for scheduling a single robot in a job-shop environment, Discrete Applied Mathematics, 119(1-2): pp.181-203.
34. Hurink J. and Knust S., (2005): Tabu search algorithms for job-shop problems with a single transport robot, European Journal of Operational Research, 162(1): pp.99-111.
35. Hurink J. and Kunst S., (1998): *Flow-shop problems with transportation times and a single robot*, Universität Osnabruck.
36. Jackson J.R., (1956): An extension of Johnson's results on job lot scheduling, Naval Research Logistics Quaterly, 3: pp.201-203.
37. Jain A.S. and Meeran S., (1999): Deterministic job-shop scheduling: Past, present and future, European Journal of Operational Research, 113(2): pp.390-434.
38. Johnson S.M., (1954): Optimal two- and three-stage production schedules with setup times included, Naval Research Logistics Quaterly, 1: pp.61-68.
39. Johnson S.M., (1959): Discussion: Sequencing n Jobs on Two Machines with Arbitrary Time Lags, Management Science, 5(3): pp.299-303.
40. Jones A. and Rabelo L.C., (1998): *Survey of Job shop scheduling techniques*.
41. Karp R.M., *Reducibility among combinatorial problems*, in *Complexity of Computer Computations*, M.e. Thatcher, Editor. 1972, Plenum Press. p.85-103.
42. Kise H., Shioyama T., and Ibaraki T., (1991): Automated Two-Machine Flowshop Scheduling: A Solvable Case, IIE Transactions, 23(1): pp.10.
43. Kubiak W., Sethi S., and Sriskandarajah C., (1995): An efficient algorithm for a job shop problem, Annals of Operations Research, 57(1): pp.203-216.
44. Kunst S., (1999): *Shop-scheduling problems with transportation*, Osnabruck: Osnabruck. p.137.
45. Lageweg B.J., Lenstra J.K., and Rinnooy Kan A., (1977): Jobshop scheduling by implicit enumeration, Management Science (pre-1986), 24(4): pp.441.
46. Land A.H. and Doig A.G., (1960): *The traveling Salesman Problem : Solution by a method of rnaking assigments.*, C.i. technology, Editor. Non publié.
47. Le-Anh T. and De Koster M.B.M., (2006): A review of design and control of automated guided vehicle systems, European Journal of Operational Research, 171(1): pp.1-23.
48. Lee C.-Y. and Chen Z.-L., (2001): Machine scheduling with transportation considerations, Journal of Scheduling, 4(1): pp.3-24.
49. Lee C.-Y., Lei L., and Pinedo M., (1997): Current trends in deterministic scheduling, Annals of Operations Research, 70(0): pp.1-41.

50. Lenstra J.K., AHG R.K., and Brucker P., (1977): Complexity of machine scheduling problems, Annals of discrete mathematics, 4: pp.121-140.
51. Lenstra J.K. and RHG R.K., (1979): Computational complexity of discrete optimization problems, Annals of discrete mathematics, 4: pp.121-140.
52. Leung J.Y.-T., (2004): *Handbook of scheduling : algorithms. models and performance analysis*. Boca Raton : Chapman & Hall
53. Lutz J.H., *The quantitative structure of exponential time*, in *Complexity theory retrospective*. 1997, Springer-Verlag. p.225-260.
54. Manne A.S., (1960): On the Job-Shop Scheduling Problem, Operations Research, 8(2): pp.219-223.
55. Mati Y., Rezg N., and Xie X., (2001): Geometric approach and taboo search for scheduling flexible manufacturing systems, Robotics and Automation, IEEE Transactions on, 17(6): pp.805-818.
56. Mellor P., (1966): A Review of Job Shop Scheduling, Operational Research Quaterly, 17(2): pp.161-171.
57. Metropolis M., Rosenbluth A., Rosenbluth M., Teller A., and Teller E., (1953): Equation of state calculations by fast computing machines, Journal of chemical physics, 21: pp.1087-1092.
58. Mitten L.G., (1959): Sequencing n Jobs on Two Machines with Arbitrary Time Lags, Management Science, 5(3): pp.293-298.
59. Müller T., (1983): *Automated guided vehicules*. IFS Ltd Springer-verlag .
60. Murty K.G., Karel C., and Little J.D.C., (1962): An automatic method for solving discrete programming problems, Econometrica, 28: pp.497-520.
61. Muth J.F. and L. T.G., (1963): *Industrial scheduling*. Emglewood Cliffs: Prentice-hall.
62. Nicholson T., *Optimization in industry*, in *Optimization Techniques*. 1971, Longmann Press: London, Chapitre 10.
63. Nowicki E. and Smutnicki C., (1996): A fast taboo search algorithm for the job shop problem, Management Science, 42(6): pp.797-804.
64. Panwalkar S.S., (1991): Scheduling of a Two-Machine Flowshop with Travel Time between Machines, Journal of the Operational Research Society, 42(7): pp.609-613.
65. Penz B., (1994): *Constructions agrégatives d'ordonnancements pour des job-shops statiques, dynamiques et réactifs*: Grenoble.
66. Pinedo M., (2002): *Scheduling : theory, algorithms and systems,* 2nd ed. New York: Prentice Hall, 586 p.
67. Pirlot M., (1996): General local search methods, European Journal of Operational Research, 92(3): pp.493-511.
68. Reeves C.R., (1993): *Modern heuristic techniques for combinatorial problems*. Halsted Press, 320 p.

69. Roy B. and Sussmann B., (1964): *Les problémes d'ordonnancement avec contraintes disjonctives*. .
70. Ruiz R., Serifoglu F.S., and Urlings T., (2006): Modeling realistic hybrid flexible flowshop scheduling problems, Computers & Operations Research, In Press, Corrected Proof.
71. S. H. Zanakis, J. R. Evans, and Vazacopoulos A.A., (1989): Heuristic methods and applications : a categorized survey, European Journal of Operational Research, 43(2): pp.88-110.
72. Sakarovitch M., *De l'efficacité des algorithmes à la complexité des problèmes*, in *Analyse combinatoire*, Dunod, 1986, Chapitre 2.
73. Schaerf A., (1999): A Survey of Automated Timetabling, Artificial Intelligence Review, 13(2): pp.87-127.
74. Sethi S.P., Sriskandarajah C., Sorger G., Blazewicz J., and Kubiak W., (1992): Sequencing of parts and robot moves in a robotic cell, International Journal of Flexible Manufacturing Systems, 4(3): pp.331.
75. Sipser M., (1997): *Introduction to the Theory of Computation,* 1ere ed. ed. Course Technology, 416 p.
76. Sotskov Y.N. and Shakhlevich N.V., (1995): NP-hardness of shop-scheduling problems with three jobs, Discrete Applied Mathematics, 59(3): pp.237-266.
77. Srinivas M. and Patnaik L.M., (1994): Genetic algorithms: a survey, Computer, 27(6): pp.17-26.
78. Strusevich V.A., (1999): A heuristic for the two-machine open-shop scheduling problem with transportation times, Discrete Applied Mathematics, 93(2-3): pp.287-304.
79. Timkovsky V., (1985): On the complexity of scheduling an arbitrary system, Soviet journal of computer and systems sciences, 5: pp.46-52.
80. Timkovsky V.G.: *Reducibility among scheduling classes*, Star data systems Inc.: Toronto.
81. Timkovsky V.G., (1997): A polynomial-time algorithm for the two-machine unit-time release-date job-shop schedule-length problem, Discrete Applied Mathematics, 77(2): pp.185-200.
82. Tomkins J.A. and White J.A., (1984): *Facilities planning*. New York: Wisley
83. Vis I.F.A., (2006): Survey of research in the design and control of automated guided vehicle systems, European Journal of Operational Research, 170(3): pp.677-709.
84. Walton M., (1988): *The deming management method.* Perigee trade, 288 p.
85. Widmer M., (1991): Job Shop Scheduling with Tooling Constraints: A Tabu Search Approach, Journal of the Operational Research Society, 42(1): pp.75-82.

86. Zacharia P.T. and Aspragathos N.A., (2005): Optimal robot task scheduling based on genetic algorithms, Robotics and Computer-Integrated Manufacturing, 21(1): pp.67-79.