Le problème de job-shop avec transport : modélisation et optimisation

Mohand Larabi

N° d'Ordre : 2092
EDSPIC : 510

Université Blaise Pascal - Clermont Ferrand II

ECOLE DOCTORALE

SCIENCES POUR L'INGENIEUR DE CLERMONT FERRAND

THESE

Présentée par

Mohand LARABI

Diplômé d'Etudes Approfondies Linguistique, Logique et Informatique

Ingénieur en informatique industrielle

pour obtenir le grade de

Docteur d'Université

Spécialité : INFORMATIQUE

Le problème de job-shop avec transport : modélisation et optimisation

Soutenue publiquement le 15 décembre 2010 devant le jury :

Monsieur	Alain QUILLIOT	Président
Monsieur	Pierre CASTAGNA	Examinateur
Monsieur	Alexandre DOLGUI	Rapporteur
Monsieur	Aziz MOUKRIM	Rapporteur
Monsieur	Philippe LACOMME	Co-Directeur de thèse
Monsieur	Nikolay TCHERNEV	Co- Directeur de thèse

A ma femme

A toute ma famille, ma belle famille, mes amis et à tous ceux qui ont supporté mon absence durant toutes ces années et à tous ceux qui ont contribué de près ou de loin à l'aboutissement de cette thèse et à tous ceux qui trouveront du plaisir à la feuilleter un jour.

Remerciements

En premier lieu mes remerciements vont pour mes deux directeurs de thèse Lacomme Philippe, et Tchernev Nicolay, pour leurs conseils, leur disponibilité et leur accueil chaleureux au sein du laboratoire LIMOS. Ils sont pour moi comme une seconde famille, tous les moments passés avec eux ont été très enrichissants sur le plan professionnel et personnel.

En second lieu je remercie les rapporteurs de cette thèse : Aziz Moukrim Professeur d'Informatique à l'Université de Technologie de Compiègne (UTC) et Alexandre Dolgui Directeur du Laboratoire en Sciences et Technologies de l'Information (LSTI) et responsable du département Méthodes Scientifiques pour la Gestion Industrielle (MSGI) pour la rapidité avec laquelle ils ont lu mon manuscrit et l'intérêt qu'ils ont porté à mon travail.
Mes remerciements vont ensuite aux autres membres du jury qui ont accepté de consacrer une partie de leur temps pour l'évaluation de mon travail :

- A Pierre Castagna Professeur à l'Université de Nantes,
- A Alain Quilliot directeur du LIMOS : tout d'abord pour avoir accepté de présider mon jury de thèse, mais aussi pour de nombreuse discussions constructives et amicales que j'ai eu avec lui au sein de LIMOS.

Je tiens également à remercier Zhao, Alexandre, Olivier, Sylverin, Eilishe, Amine, Aziz, Nassima, Leila, Adelaïde, Nassim et mes collègues de L'IUP management de Clermont-Ferrand pour tous les bons moments que nous avons partagé ensemble.
Et mes remerciements vont aussi à celles et ceux, très nombreux, qui ont participé de prés ou de loin à mener à terme ce travail de recherche.

Mes derniers mots sont réservés à ma famille :

- A ma femme qui a tant sacrifier pour l'aboutissement de cette thèse, qui a été toujours présente à mes cotés malgré la distance : son soutien m'était d'un grand apport.
- A ma famille qui m'a soutenu et encourager sans relâche durant toutes ces années.
- A ma belle famille qui m'a épaulé durant les moments difficiles.

Sommaire

Introduction générale

La gestion des systèmes de production, de biens ou de services pose de très nombreux problèmes touchant la gestion de production, le marketing, la gestion des ressources humaines. La résolution de ces problèmes nécessite l'utilisation de techniques d'optimisation et/ou d'évaluation de performances issues de domaines très variés. Nous nous intéressons dans cette thèse, aux problèmes d'optimisation discrets, c'est-à-dire à la résolution de problèmes que l'on peut traiter par des techniques d'optimisation et qui sont généralement, ou le plus souvent liées à des problèmes de planification et/ou d'ordonnancement.

La difficulté de résolution de ces problèmes d'ordonnancement vient du grand nombre d'entités à gérer et de la nécessité de construire un planning comportant un grand nombre de taches et ceci avec de nombreuses contraintes comme par exemple des contraintes de précédence, de transport, de date, de délais et/ou de disponibilité de ressource.

La première étape dans la résolution de l'un de ces problèmes, consiste à définir et à construire un modèle du système qui est utilisé par la suite pour construire, rechercher et améliorer des solutions. La construction et l'élaboration d'un modèle constitue la première difficulté à résoudre. On peut remarquer que pour tous les problèmes « classiques » d'optimisation, des modèles efficaces sont connus et qu'ils reposent dans la majorité des cas, sur un usage intensif de la théorie des graphes.

La deuxième étape consiste à proposer et concevoir des méthodes efficaces tirant profit de la modélisation et permettant de définir des techniques d'amélioration itérative pour la recherche locale, des techniques de diversification pour l'exploration de l'espace et des techniques globales d'exploration de l'espace des solutions.

Dans la majorité des cas, on ne travaille pas directement sur le modèle à cause de la difficulté liée à la complexité du modèle. Cette remarque s'applique à d'autres domaines que celui de l'ordonnancement comme par exemple le domaine des tournées de véhicules. Sur les tournées de véhicules beaucoup de méthodes travaillent sur un tour géant (solution du TSP) qui est ensuite découpé (optimalement) pour obtenir une solution du problème de VRP, de LRP etc. Dans le cadre du job-shop par exemple il est courant d'utiliser des vecteurs par répétition (séquence de Bierwith) et de définir une méthode permettant d'associer à une séquence une solution du problème. Cela revient à dire qu'on travaille sur deux espaces différents : l'espace des solutions et l'espace (ou ensemble) des objets qui par une fonction (que l'on cherche définir la plus efficace possible) donne une solution de l'espace. Ces méthodes alternent alors entre les deux espaces et il s'agit, le plus souvent de tirer profit des ces deux espaces pour obtenir un schéma algorithmique performant.

Dans cette thèse nous nous intéressons à un problème d'ordonnancement particulier : le problème du job-shop auquel nous ajoutons des contraintes de transport. Notre apport consiste en l'ajout de contraintes de deux types :

- la prise en compte de plusieurs moyens de transport de capacité unitaire ;
- la prise en compte de plusieurs moyens de transport de capacité non unitaire.

Pour la résolution de ces problèmes, nous généralisons la démarche classique appliquée au job-shop et qui repose :

- sur la notion de graphe disjonctif non orienté pour modéliser un problème ;
- sur la notion de graphe disjonctif orienté pour modéliser une solution ;

- sur la notion de plus long chemin pour le calcul des dates au plus tôt des opérations.

Dans la suite, nous utilisons le schéma ci-dessus pour proposer des heuristiques et des métaheuristiques pour les problèmes de job-shop avec transport.

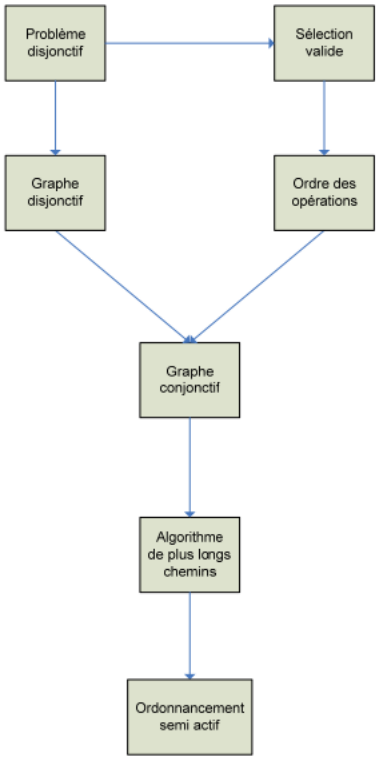


Figure 1-1 : Le schéma de modélisation du Job-Shop tel que résumé par Caumond (2008)

Cette thèse est structurée en 4 chapitres.

Le chapitre 1 présente le domaine d'étude : les systèmes flexibles de production. Après une présentation générale nous donnons une présentation des problèmes d'ordonnancement, et donnons quelques modèles théoriques d'ordonnancement.

Le chapitre 2 aborde le problème du job-shop en tant que problème théorique. Nous présentons pour ce problème, un état de l'art, les principaux algorithmes de la littérature. Nous rappelons la modélisation sous la forme de graphe disjonctif de ce problème et nous présentons les principes de base des principales méta-heuristiques.

Le chapitre 3 aborde le problème du job-shop avec transport. Une nouvelle modélisation est présentée. Tous les éléments d'une méthode d'optimisation sont proposés et présentés. Les résultats numériques concernent des instances de la littérature et permettent d'évaluer les performances des méthodes proposées.

Le chapitre 4 aborde le problème du job-shop avec des robots de capacité non unitaire. Une nouvelle modélisation est présentée. En particulier, nous présentons une modélisation linéaire en nombres entiers et une modélisation sous la forme d'un graphe conjonctif-disjonctif.

Cette thèse se termine par une conclusion dans laquelle nous rappelons les principales propositions que nous avons faites et dans laquelle, nous proposons des pistes de recherche dans la continuité de nos travaux.

Chapitre I : Présentation de la problématique

Ce chapitre présente la problématique de cette thèse : l'optimisation pour l'ordonnancement des systèmes de production.

Liste des tableaux

Liste des figures

1. Introduction

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

Pour faire face à une concurrence internationale de mieux en mieux organisée et à un marché de plus en plus fluctuant, les industriels ont développé la flexibilité de leurs systèmes de production afin d'améliorer leur productivité et leur réactivité, tout en réduisant le plus possible les coûts de production et en augmentant la qualité des produits fabriqués. C'est dans ce contexte qu'ont émergé les systèmes flexibles de production (*SFP*), objet de notre travail.

Ces systèmes ont une complexité fonctionnelle et structurelle : ils comportent de nombreuses entités en interaction et sont sujets à de multiples aléas. Cette complexité est une des raisons pour lesquelles les problèmes qu'ils posent – problèmes de dimensionnement, de planification, d'ordonnancement…- sont généralement reconnus comme très difficile à résoudre. Ils doivent être étudiés de manière méthodique et rigoureuse afin de détecter et de quantifier leur impact sur les performances quantitatives de SFP.

La plupart des problèmes de planification sont, ou se ramènent à, des problèmes d'optimisation. Pour la résolution de ces problèmes il est opportun de faire appel à la construction de modèles. C'est une solution très pertinente, pourvu qu'elle soit menée dans un cadre méthodologique. Compte tenu de leur complexité ces problèmes font parti de l'optimisation combinatoire (« Combinatorial Optimization Problems » ou COPs).

Nombre de ces problèmes sont très difficiles à résoudre, et il n'existe pas de méthodes mathématiques efficaces permettant de le faire dans des temps de calcul acceptable. De ce fait, et étant donné leur importance, les *COPs* constituent un domaine de recherche dynamique, dont l'intérêt n'a cessé d'augmenter durant les dernières décennies.

Ce chapitre est structuré en trois parties.

Dans la première partie (paragraphe 2), nous présentons le domaine d'étude : les SFP. Après une présentation générale des SFP qui porte sur leur définition et leur structuration, nous abordons les différents problèmes regroupés par rapport à la phase de conception des SFP et par rapport à la phase d'exploitation.

La seconde partie (paragraphe 3) contient une présentation des problèmes d'ordonnancement que nous définissons et dont nous donnons les éléments importants. Outre la notation et les principaux problèmes et modèles théoriques d'ordonnancement, la démarche de modélisation est présentée.

Dans la troisième partie (paragraphe 4), la notion de *POCs* est précisée, ainsi que quelques problèmes les plus représentatifs sont présentés. La difficulté des ces problèmes a abouti à la théorie de la complexité.

2. Le domaine d'étude : les systèmes flexibles de production

2.1. Présentation des Systèmes Flexibles de Production (SFP)

2.1.1. Spécificités et définition des SFP

Pour conserver, ou améliorer, leur position dans le marché international, les industriels doivent concilier deux objectifs considérés comme antagonistes, à savoir :

- Assurer une *productivité* la meilleure possible, essentiellement en termes de coût, de quantité et de qualité de production,
- Etre capables de s'adapter au mieux et rapidement à un environnement très fluctuant et souvent imprévisible, en se dotant de systèmes possédant une certaine *flexibilité* à l'égard non seulement d'un marché caractérisé par une demande variable en quantité et en nature, mais aussi dans la gestion de la production (gestion des pannes. . .).

Le concept de flexibilité est capital car il est intimement lié au concept d'incertitude qui apparait aussi bien dans des activités concernant le long terme, comme les prévisions des ventes, que dans des aspects plus opérationnels comme la gestion des pannes ou la mise en place de politiques de maintenance préventive. Un certain nombre de travaux traitent de la quantification de la flexibilité d'un système de production. Kochikar et Narendan (1992) propose notamment un cadre d'évaluation reposant sur quatre mesures de flexibilité pour un système de production donné :

- la présence d'alternatives de production proposées pour chaque type de produits (ainsi que la rentabilité de chaque alternative) ;
- la versatilité, ou flexibilité, des machines ;
- la nature du réseau des mouvements de produits entre unités de production ;
- la possibilité d'introduire de nouveaux types de produits dans la production. Cet aspect concerne typiquement le long terme.

Les réponses traditionnellement implantées par les industriels dans l'outil de production favorisent généralement un des deux objectifs antagonistes, et sont donc principalement de deux types :

- des systèmes, exemplifiés par les chaines de production, relativement automatisés, très productifs mais très peu flexibles : ils permettent une *production en masse* d'un nombre *très restreint* de types de produits, voire d'un seul type de produit ;
- des systèmes plus conventionnels de type *job-shop* qui sont très flexibles mais peu productifs : ils permettent de prendre en compte une demande dont la nature change, grâce principalement à une forte intervention humaine.

Le développement de nouvelles technologies dans le monde manufacturier a permis aux industriels de relever le défi sans cesse croissant relatif à ces deux objectifs antagonistes. Ainsi, dès le début des années 80, on a assisté à l'émergence d'une nouvelle classe de systèmes de production : les SFP.

La littérature propose différentes définitions des SFP. Par exemple, Kaltwasser et al., (1986) donne la définition fonctionnelle d'un SFP suivante :

« Un SFP est un système de production hautement automatisé capable de produire une grande variété de types de produits en utilisant le même équipement et le même système de contrôle. »

D'autres auteurs, comme Byrkett et al., (1988) ou Gunasekaran et al., (1993), définissent les SFP relativement à leurs composants :

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

Nous retenons la définition proposée par MacCarthy et Liu (1993) qui intègre les aspects fonctionnel et structurel :

« Un SFP est un système de production capable de produire différents types de pièces et qui est composé d'un ensemble de machines à commande numérique interreliées par un système automatisé de T/M. La gestion et le contrôle de ce système sont informatisés. »

Comme nous le verrons dans la section suivante, on peut ainsi définir plusieurs variantes, ou *types,* de SFP qui apportent des réponses *variables* aux objectifs des industriels. Il convient alors de choisir la solution la mieux adaptée au milieu industriel concerné.

2.1.2. Les différents types de SFP

Malgré l'existence de nombreuses définitions d'un SFP, il y a un consensus de la communauté nationale et internationale Brauner et al., (2005) pour admettre la présence des trois sous-systèmes suivants dans un SFP :

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

Cependant, le consensus s'arrête là, pour laisser place à de nombreuses classifications quant aux différents types de SFP. Cette absence d'homogénéité a bien entendu, des conséquences sur :

- la clarté de la communication entre chercheurs et/ou experts industriels,
- la facilité d'exploiter des relations entre types de SFP, notamment dans le cadre de décomposition de problèmes complexes.

Dans la suite, nous présentons une classification simplifiée proposée par MacCarthy et Liu (1993) et adoptée par la communauté scientifique Brauner et al., (2005). Cette classification nous semble pertinente en raison de sa clarté, de sa discussion critique par rapport aux classifications existantes et des relations (notamment hiérarchiques) qu'elle permet d'exhiber entre les différents types de SFP.

La démarche proposée par MacCarthy et Liu (1993) est une démarche de construction *progressive* des différents types de SFP. Plus précisément, les auteurs définissent quatre types de SFP :

Définition 1 (Type I : Machine flexible (MF)) *Une machine flexible de production est une unité de production dont le contrôle est informatisé et qui est composée d'une seule machine à commande numérique capable de changements autonomes d'outils, d'un moyen de T/M et d'une zone de stockage des pièces.*

Le moyen de T/M est typiquement un robot qui peut être absent lorsqu'une MF est intégrée dans une unité de production comportant un système de T/M capable d'assurer les opérations de T/M pour cette MF.

Définition 2 (Type II : Cellule flexible de production (CFP)) *Une cellule flexible de production est un SFP formé par un ensemble de MF où le terme ensemble signifie « au moins deux » qui partagent le même moyen de T/M.*

Définition 3 (Type III : Système flexible de production multi-machines (SFPMM)) *Un système flexible de production multi-machines est un SFP formé d'un ensemble de MF reliées par un système automatisé de T/M capable de servir au moins deux MF simultanément.*

Définition 4 (Type IV: Système flexible de production multi-cellules (SFPMC)) *Un système flexible de production multi-cellules est un SFP composé d'un ensemble de CFP (et, éventuellement, d'un ensemble de MF) interreliées par un système automatisé de T/M capable de servir au moins deux unités de production (MF ou CFP) simultanément.*

Des exemples de ces différents types de SFP sont donnés en Figure 2-1 (d'après MacCarthy et Liu (1993). Ces définitions permettent donc de classifier *clairement* les différents types de SFP en fonction de la nature de leurs composants de production et de leurs systèmes de T/M. La classification ainsi construite est cohérente avec les classifications proposées précédemment, et a été testée avec succès sur un grand nombre de SFP industriels MacCarthy et Liu (1993). Elle a l'avantage de proposer une structuration hiérarchique des SFP et mettre en rélation *composer – composant* les différents types de SFP.

Les topologies les plus souvent rencontrées des SFP d'après Kouvelis et Kim (1992) et communément acceptés par la communauté scientifique Brauner et al., (2005) sont :

- une topologie circulaire Figure 2-1 type I et II ;
- une topologie comportant des unités flexibles de production/stockage rangées en réseau circulaire unidirectionnel Figure 2-2 ;
- une topologie comportant des unités flexibles de production/stockage rangées en plusieurs lignes et colonnes Figure 2-1 type III ;
- une topologie linéaire comportant des unités flexibles de production/stockage rengées en une ligne Figure 2-3 ;
- une topologie linéaire comportant des unités flexibles de production/stockage rengées en deux lignes Figure 2-1 type IV.

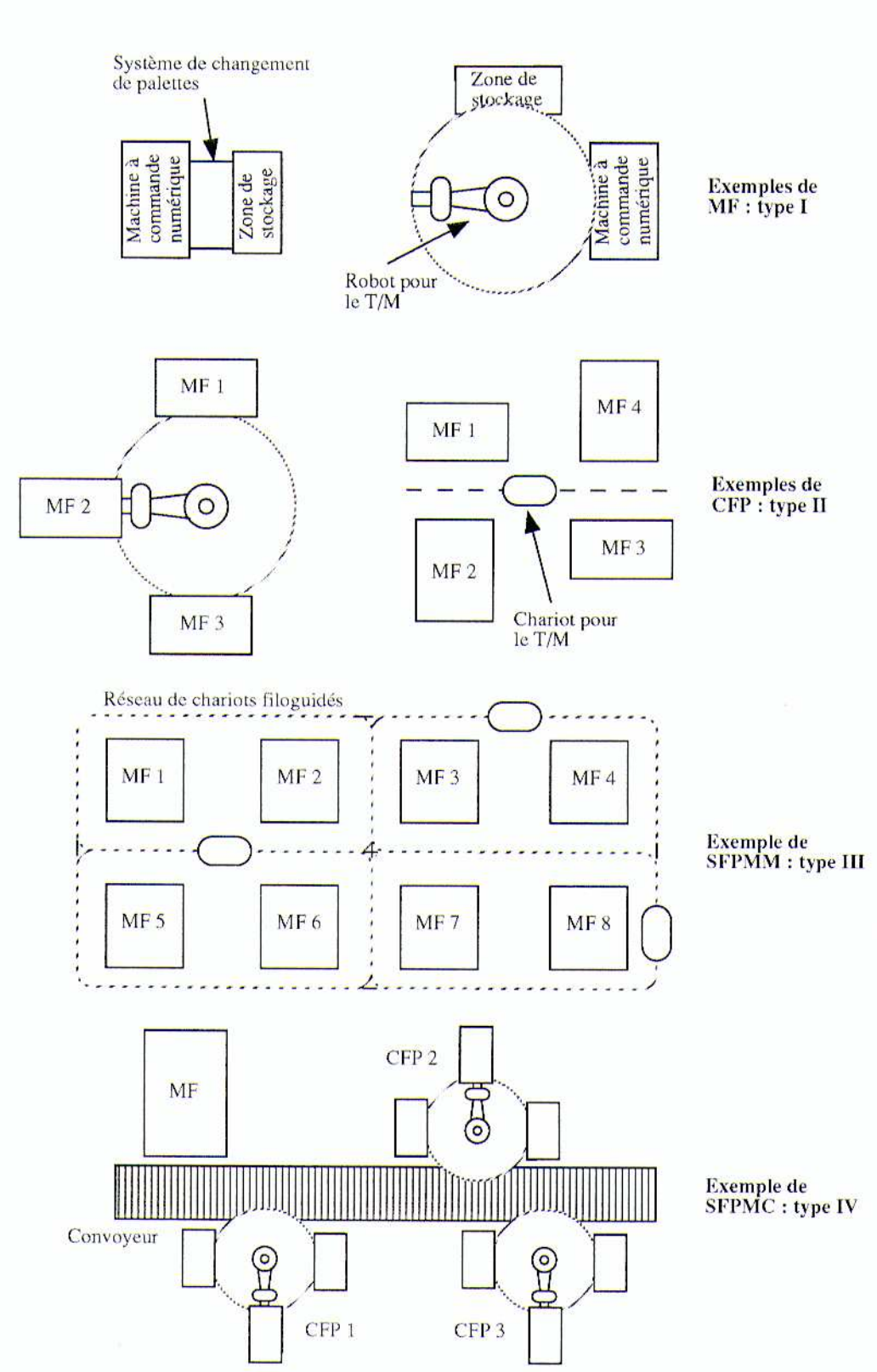


Figure 2-1 : Exemples de SFP de différents types (d'après MacCarthy et Liu (1993))

Suite à l'analyse des différentes topologies, on peut constater que le dénominateur commun est le moyen automatisé de transport/manutention. Des études montrent que l'efficacité d'un atelier flexible de production est essentiellement due à celle de ses moyens de T/M. Ces études, effectuées dans différents pays, ont montré que, si la part des coûts de transport/manutention

imputés dans les coûts de production est très variables selon les secteurs, ils sont, par contre très importants, et peuvent atteindre 15 à 40% des coûts de production Krampe et Lucke (1989). Cette situation conduit à penser qu'il est impératif de prendre en compte l'impact des systèmes automatisés de transport/manutention lors de la résolution des problèmes liés à la conception, à la planification et à l'ordonnancement des SFP.

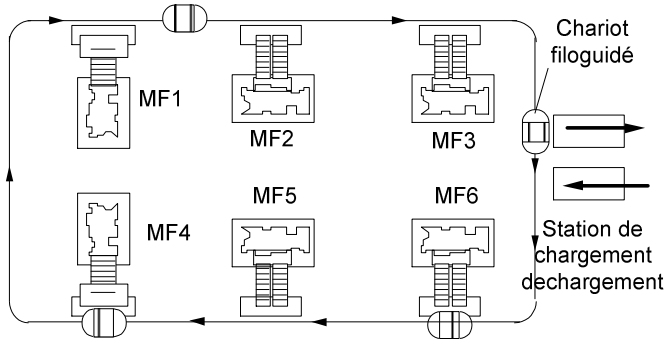


Figure 2-2 : Topologie circulaire unidirectionnel (d'après Brauner et al., (2005))

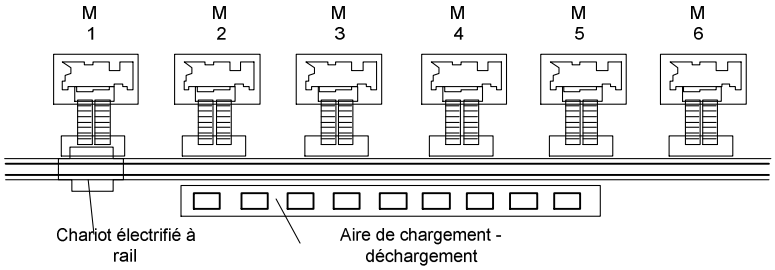


Figure 2-3 : Topologie linéaire (d'après Brauner et al., (2005))

D'autre part l'analyse de cinquante SFP existant, réalisé par Kusiak (1988), montre que les systèmes automatisés de transport/manutention les plus utilisés sont les systèmes avec des robots portiques ou des chariots automatiquement guidés (filoguidé ou optoguidé). Compte tenu de cette analyse nous allons concentrer nos efforts sur les SFP muni de dernier type de transport/manutention.

2.2. Les problèmes posés par les SFP

Tout au long du cycle de vie d'un SFP il faut résoudre des problèmes de planification de production. Looveren et al., (1986) classifie ces problèmes selon trois niveaux temporels :

- *le niveau stratégique* qui correspond aux problèmes de conception d'un SFP relativement aux objectifs retenus. Ce niveau induit clairement des décisions à long terme concernant notamment les composants du SFP.
- *le niveau tactique* qui comprend, entre autres, le choix des produits à usiner simultanément et conformément aux demandes de production. Plus globalement, ce niveau inclut les décisions à prendre à la suite du niveau stratégique avant qu'une exploitation ne puisse débuter. Dans ce contexte, il s'agit de décisions à moyen terme.
- *le niveau opérationnel* qui concerne les décisions liées au *pilotage* d'un SFP. Il s'agit donc d'un niveau décisionnel à court terme.

Soulignons que ces différents horizons correspondent à des niveaux de détails différents. Par exemple, le niveau opérationnel requiert une description détaillée d'un SFP qui n'est pas forcément nécessaire (voire même inconnue) au niveau stratégique.

Dans la suite de cette partie, nous précisons chacun de ces niveaux décisionnels, en reprenant les travaux présentés dans Stecke (1984) et Brauner et al., (2005). Les différents problèmes sont ainsi regroupés par rapport :

- à la phase de conception ;
- à la phase d'exploitation.

2.2.1. Phase de conception des SFP

Dans cette partie nous présentons les problèmes de conception des SFP avec des chariots automatiquement guidés. Ces problèmes peuvent être regroupés en deux catégories Brauner et al., (2005) :

- les problèmes de dimensionnement ;
- les problèmes d'agencement.

On s'intéresse donc au niveau stratégique au compromis cycle de vie d'un SFP/coût d'un SFP. En particulier, le premier aspect de ce compromis est lié à l'aptitude d'un SFP à prendre en compte une demande fluctuante, alors que le second englobe les coûts de construction, d'exploitation et de rendement d'un SFP. Les décisions à prendre étant nombreuses, l'étude du compromis est généralement décomposée de la manière suivante Stecke (1984) :

- *la sélection d'une famille de produits* susceptibles d'être produits par le SFP. Cette procédure requiert l'intégration de prévisions de production long terme ;
- *l'élaboration de gammes/nomenclatures* pour les types de produits concernés par l'étape précédente. En particulier, ces gammes doivent fournir des informations concernant l'ordre des opérations, leurs durées et les outils requis pour ces opérations. Ces informations sont exploitées pour déterminer la capacité de production et le nombre d'outils par type de machines du SFP ;
- *la spécification du type de système de transport/manutention automatiquement guidé.* Ce choix est extrêmement important car la topologie du SFP et l'agencement des machines dépend du type de systèmes de transport/manutention Kusiak (1988) ;
- *la sélection du type de SFP,* parmi les classes définies précédemment, qui semble être le plus pertinent relativement aux objectifs de flexibilité retenus. Ce choix a un impact au niveau de l'automatisation et donc, du contrôle en termes d'architecture informatique et de stratégies de contrôle du système considéré.

Ces points entrent dans le cadre de décisions initiales de conception. Une fois les décisions prises, les problèmes suivants de dimensionnement sont considérés Brauner et al., (2005) :

- *la détermination du nombre de machines.* Le nombre de machines dépend des gammes de fabrication, des taux souhaités de production pour chaque type de pièces, des implantations possibles pour les différentes machines, du coût des machines et de leurs caractéristiques (temps d'usinage, temps de réglages…). La détermination du nombre de machines et le choix de leur implantation influencent, notamment, l'emplacement de la ou des voies de circulation du réseau de guidage des chariots et le nombre de chariots à utiliser.
- *la détermination du nombre, de la capacité et de l'emplacement des zones de stockage des chariots, appelées également parking.* Lors de la production, les chariots ne sont pas tous nécessairement utilisés simultanément, aussi pour éviter des engorgements sur les

voies de circulation, il peut être très intéressant de prévoir des parkings pour les chariots. Ces parkings sont également utilisés pour recharger les batteries des chariots. Le choix du nombre de parkings et le choix de leur emplacement vont directement influencer le nombre de chariots qui devront et pourront être utilisés et vont également avoir un impact direct sur les méthodes d'exploitation qui pourront être envisagées.

- *la détermination de la capacité des zones de stockage des pièces*. Faut-il prévoir une zone de stockage centrale, ou bien des zones locales, ou bien encore aucune zone de stockage ? Qu'elle doit être la taille de ces zones ? Ce point nécessite une pondération des apports (réduction des congestions, des phénomènes de famine) et des inconvénients (coûts de stockage et, en termes de place, des zones de stockage). Plus les zones de stockage des pièces ont des capacités importantes plus la gestion des chariots est facilitée, car avec des capacités de stockage importantes, l'attente des chariots est réduite et certains engorgements peuvent ainsi être évités. Ces capacités de stockage ont donc une influence directe sur les performances du système.
- *la détermination du nombre de chariots*. Le nombre de chariots conditionne la fluidité de la circulation des pièces dans le système de production. En effet, si ce nombre est insuffisant par rapport aux niveaux de production des machines, les pièces vont saturer les zones de stockage en sortie de poste de travail et les temps d'attente pour obtenir un chariot pour le transport seront très importants. Si ce nombre est très important par rapport aux taux de production des machines, nous assistons dans ce cas au phénomène inverse, c'est-à-dire une importante attente de la part des chariots provoquant un engorgement des parkings et un engorgement des voies de circulation pouvant aboutir à des blocages du système. De plus les chariots ont un coût extrêmement élevé, il est donc crucial de déterminer leur nombre avec précision. Ceci dépend directement de la configuration du système flexible de production, des gammes, des temps d'usinage, des vitesses de déplacement à vide et à charge, du nombre de pièces qu'ils peuvent et de leurs modes de fonctionnement (monodirectionnel ou bidirectionnel) et de leurs règles de gestion.
- *la spécification de l'architecture* du système informatique de contrôle ;
- *la conception et la détermination du nombre* de palettes de chaque type, qui a un impact sur la production du SFP ;
- *l'élaboration de stratégies de planification et de contrôle globale*, cohérentes avec les objectifs fixés. Elles sont développées plus en détail et de manière progressive aux niveaux tactique et opérationnel,
- *la conception d'un cahier de charges* global pour la partie informatique de contrôle du SFP.

La résolution de ces problèmes peut être séquentielle ou bien certaines d'entre elles peuvent être abordées simultanément. Compte tenu des interactions entre ces niveaux de problèmes, le processus global de conception est itératif. En particulier, quand une alternative de conception a été définie, il est nécessaire de l'évaluer afin de juger si elle est satisfaisante. Une telle évaluation peut alors conduire à la définition d'une autre alternative de conception qui est évaluée à son tour.

Les problèmes d'agencement comprennent Brauner et al., (2005) :

- *l'agencement des machines et leurs regroupements en cellules* qui doivent tenir compte des gammes des pièces qui génèrent les flux de produits entre les différentes machines et cellules. L'objectif est alors de minimiser les temps de déplacement entre les différentes machines et cellules. Ceux-ci comprennent les temps de déplacement à charge et à vide des chariots.
- *l'agencement du réseau de guidage* (orientation optimale du réseau de guidage). La conception du réseau de guidage est de façon générale fortement contrainte par

l'implantation des postes de travail à desservir, par le mode de fonctionnement des chariots (monodirectionnels, bidirectionnel) et par la place disponible.

On trouve, pour les chariots, trois types de réseaux de guidage :

- les réseaux de guidage monodirectionnels ;
- *les réseaux de guidage bidirectionnels*. Sur ce type de réseaux de guidage circulent soit des chariots bidirectionnels, soit des chariots monodirectionnels pouvant faire demi-tour ;
- *les réseaux de guidage constitués de deux chemins monodirectionnels parallèles* fonctionnant en sens inverse. Cette solution est efficace car elle associe les avantages de la bidirectionnalité avec la simplicité de commande des circuits monodirectionnels mais elle nécessite une surface au sol importante.

La première solution est très répandue, très simple au niveau de la commande car le risque de blocage est limité. Elle conduit par contre à des performances globales bridées à cause de l'allongement des distances parcourues. Elle introduit aussi une contrainte au niveau de la conception du circuit. Ce dernier ne peut être constitué que de boucles fermées interconnectées. La seconde solution est certainement la plus performante. Une étude en simulation réalisée par Egbelu et Tanchoco (1986) a montré que le modèle bidirectionnel peut améliorer considérablement les performances du système de production. La commande est par contre plus délicate car elle doit éviter les blocages résultant du déplacement de deux chariots en sens inverses sur un même réseau de guidage.

Il est important de noter que les décisions liées à la conception prises au niveau stratégique ont un impact très fort sur la phase d'exploitation présentée dans le paragraphe suivant.

2.2.2. Phase d'exploitation des SFP

Cette phase concerne la planification au niveau tactique (moyen terme) et opérationnel (court terme).

Le problème de planification à moyen terme est reconnu comme très difficile. Son étude dans la littérature relève très souvent d'une décomposition progressive en cinq sous-problèmes proposée Stecke (1984) :

- *la sélection des types de pièces,* Ce problème peut être abordé en prenant en compte soit les contraintes temporelles concernant les demandes, soit les contraintes de production imposées par les types de produits demandés afin d'utiliser au mieux les ressources du SFP ;
- *le regroupement des machines,* qui consiste à déterminer des groupes de machines *fonctionnellement* équivalentes. Il est en effet reconnu que cette technique de regroupement peut améliorer la plupart des performances d'un système ;
- la détermination des quotas des campagnes de production des types de produits sélectionnés précédemment ;
- *l'allocation des ressources,* et en particulier des palettes, aux types de produits sélectionnés. Cette allocation peut mener à une révision des quotas de production déterminés à l'étape précédente ;
- *l'affectation des opérations* (*et des outils associés),* requises par les types de produits sélectionnés, entre les groupes de machines. Ce problème est également appelé *problème de chargement.*

Les problèmes soulevés au niveau opérationnel concernent les décisions à prendre le plus souvent en « *temps réel* » (d'où le terme de pilotage utilisé initialement), tandis que le SFP produit, afin de pérenniser les efforts entrepris dans les phases précédentes. En particulier, il s'agit d'un niveau décisionnel se situant *en aval* du niveau tactique qui sous-entend des décisions à prendre avant de lancer une campagne de production. Il est important de noter que la notion *« temps réel »* est toujours relative par rapport à une échelle de temps (semaine, jours, équipe, heure…).

On peut distinguer plusieurs types de problèmes à ce niveau et qui sont généralement ventilés selon deux problématiques Brauner et al., (2005) :

- les problèmes d'ordonnancement dans le SFP,
- les problèmes de contrôle des aléas de production.

Les problèmes d'ordonnancement concernent l'ensemble des décisions liées à la gestion du flux des pièces dans le SFP afin de rester cohérent relativement aux objectifs définis. Trois niveaux de décision sont généralement considérés (Stecke (1984)) :

- *la détermination d'une séquence (optimale) d'introduction des pièces* dans le SFP. Il s'agit alors d'un ordre d'introduction des pièces dans le SFP de telle sorte que des quotas de production, ou bien des quotas de présence de pièces dans le SFP, soient satisfaits. Un tel ordre peut être fixé à l'avance, ou bien être fonction de l'état du SFP (détermination en temps réel) ;
- *le développement de méthodes de gestion du flux des pièces* dans le SFP. En particulier, de telles méthodes visent à résoudre les problèmes de choix, typiquement nombreux dans un SFP, rencontrés lors d'un usinage : quelle machine choisir pour une opération quand plusieurs alternatives sont possibles ? La littérature propose généralement des algorithmes d'ordonnancement prédictif (off-line) répondant aux questions *où ?* et *quand ?*
- *la gestion de plusieurs pièces en attente d'un usinage sur une même machine.* Il s'agit alors de proposer des politiques de gestion des niveaux de priorité des pièces pour chaque machine relativement aux changements d'outils, par exemple.

Dans leur relation avec les objectifs fixés, ces problèmes sont peut-être plus concernés par l'amélioration du taux d'utilisation des SFP.

Les problèmes de contrôle des aléas de production regroupent les décisions liées à la surveillance du SFP et de la production, afin de rester cohérent avec les objectifs fixés (et toutes les méthodes développées précédemment pour atteindre ces objectifs). Ils concernent :

- *l'implantation de politiques de gestion des pannes de machines et/ou d'outils.* Ces politiques sont en principe déjà étudiées au niveau de la conception du SFP. En particulier, le regroupement des machines a une influence claire sur ces politiques. Ces politiques impliquent des adaptations, voire des changements, par rapport à un ordonnancement nominal. Il faut également prendre en compte la gestion des réparations ;
- la détermination de politiques de maintenance de l'équipement : maintenances à dates fixes, périodiques ou préventives ;
- *la détermination de procédures d'inspection* des pièces finies et/ou en cours d'usinage afin de contrôler la qualité de production.

Gérer efficacement un système flexible de production requiert la mise en œuvre des solutions proposant une résolution simultanée Brauner et al., (2005) :

- de la planification des mouvements des chariots ;
- de l'ordre d'entrée des pièces du système ;
- des conflits d'accès aux ressources (tronçons...).

Les solutions calculées doivent impérativement prendre en compte les contraintes suivantes :

- le nombre maximal de pièces simultanément autorisées dans le système est fixé ;
- la capacité des stocks d'entrée/sortie est limitée ;
- les déplacements à vide du chariot doivent être pris en compte ;
- la politique de gestion des stocks est très importante. La plus répandue est la politique FIFO car c'est la plus facile à utiliser dans le système ;
- la politique de gestion du chariot utilisée et implantée dans le superviseur de l'atelier est généralement une règle de priorité (FIFO, STT, . . .) permettant une implantation économe en ressources et en temps de calcul ;
- la politique de gestion du chariot alors qu'aucune pièce n'est disponible dans un stock de sortie consiste à immobiliser le chariot devant une station.

2.3. Bilan sur la problématique du domaine d'étude

Parmi les nombreux problèmes que posent les SFP, nous nous sommes intéressés particulièrement aux problèmes liés à leur conception et à leur exploitation. Ces problèmes sont généralement regroupés en trois catégories formant trois classes de problèmes, chacune ayant ses propres objectifs en fonction de son horizon : conception, planification, et pilotage.

Quel que soit le niveau auquel on s'intéresse, la charge du système est toujours une charge prévisionnelle : aux niveaux stratégique et tactique, la charge dépend des commandes et des prévisions clients ou des consommations des stocks qui sont fluctuantes par essence. Au niveau opérationnel, il est possible que la charge découle directement des activités internes de l'entreprise et donc d'activités planifiées par l'entreprise. Mais, même dans ce cas, des imprévus peuvent survenir : des retards peuvent surgir, des défauts peuvent retarder ou compromettre des livraisons, des opérations jugées plus prioritaires peuvent apparaître. Dans les trois niveaux de planification, on doit pouvoir répondre aux questions suivantes : dans quel ordre dois-je écouler la charge prévue, comment respecter la planification du niveau supérieur, comment puis-je intégrer telle ou telle charge non prévue dans le planning actuel ?

Le pilotage du système consiste à résoudre les problèmes éventuels lors de la réalisation des opérations. Ainsi, le pilotage doit déterminer comment les conflits d'accès aux ressources doivent être résolus et comment les événements aléatoires influencent ou entrainent la modification du planning prévu. Ainsi, si le planning prévoit la réalisation d'une opération et que cette dernière ne peut être réalisée par manque de matière ou de ressources, il faut déterminer quelle opération doit être effectuée à sa place : la prochaine opération prévue, une opération du même type ou d'un type proche...

Schématiquement, on peut dire que, durant la phase de conception, on résout les problèmes avant que le système ou son évolution n'existe. La planification détermine pour un système donné, la charge qui peut être écoulée. Enfin, pour un système et une charge donnés, la phase d'exploitation consiste à tenter d'écouler la charge de manière à suivre au mieux le planning.

Il est extrêmement difficile de mettre une frontière clairement définie entre planification et ordonnancement (lors de la phase d'exploitation) comme le montre la Figure 2-4 . C'est un exercice périlleux auquel nous ne souhaitons pas nous livrer. Néanmoins on peut dire que lors de la planification on cherche à améliorer l'efficacité, i.e. maximiser le degré d'atteinte des objectifs visés, et lors de l'ordonnancement on cherche à améliorer l'efficience, i.e. minimiser les ressources utilisées pour réaliser ces objectifs comme le montre la Figure 2-5.

Cependant, il est possible, en fonction de l'horizon temporel auquel on s'intéresse et du niveau de finesse de la modélisation du système en cours d'étude et celle des données liées à la charge du système, d'utiliser des méthodes issues de l'ordonnancement pour résoudre des problèmes de planification.

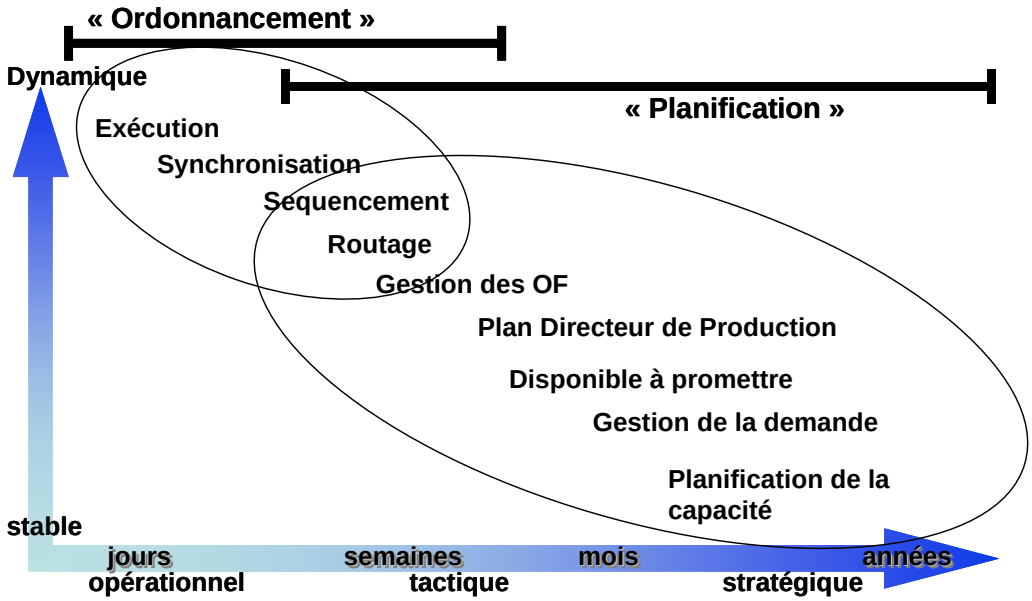
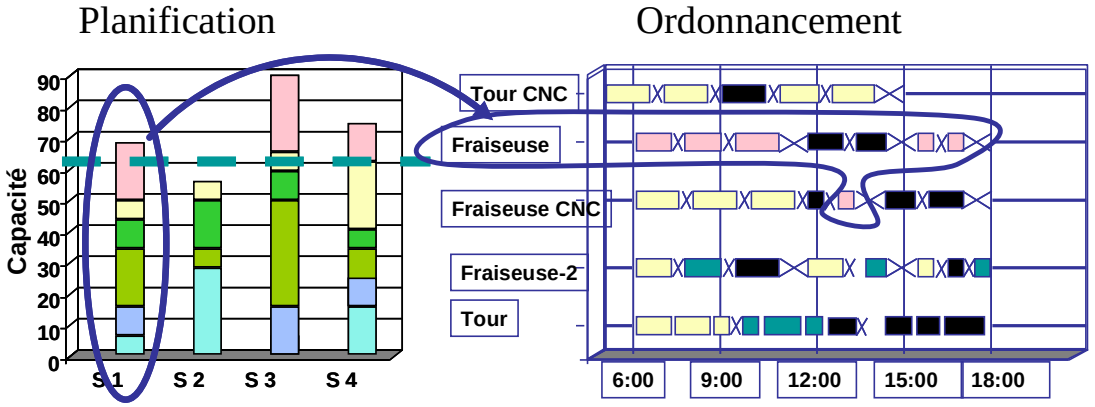


Figure 2-4 : Zone commune entre la planification et ordonnancement



Efficacité

- Comment faire
- Où le faire
- Combien faire
- Les matières nécessaires
- Les Ressources nécessaires

Efficience

- Comment le faire au mieux (routage)
- Séquencement (minimiser les changements)
- Synchronisation
- Priorités, contraints & conflits
- Pilotage et exécution
- Gestion des ordres prioritaires

Figure 2-5 : Objectifs de la planification et de l'ordonnancement

Les systèmes de production sont très variés et on pourrait penser que leurs modèles le sont aussi. Pourtant, la littérature a mis en évidence un nombre relativement faible de modèles théoriques pour l'ordonnancement des systèmes de production. Ces modèles sont très généraux et peuvent être appliqués à des problèmes variés, ils modélisent des parties de systèmes de production et / ou des systèmes de production. Même s'ils peuvent être appliqués à des domaines

plus larges, la terminologie qu'ils emploient est essentiellement tournée vers les systèmes industriels de production (machine, job, tâche, opération et ressources, …).

Tous ces modèles d'ordonnancement font ou peuvent faire appel aux techniques d'optimisation. Dans la suite de cette thèse, nous étudions particulièrement ces techniques. Ainsi, nous allons étudier, dans la prochaine partie les problèmes d'ordonnacement avant de s'intéresser aux problèmes d'optimisation.

3. Les problèmes d'ordonnancement

3.1. Définition du problème

« Ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution » Carlier et Chrétienne (1988).

Soit un problème d'ordonnancement. Les notations suivantes sont introduites pour définir l'énoncé du problème. Soit T l'ensemble de n tâches, $T=\{T1, T2, ..., Tn\}$, P l'ensemble de m processeurs (machines) $P=\{P1, P2, ..., Pm\}$ et R l'ensemble des ressources additionnelles $R=\{R1, R2, ..., Rs\}$. On peut définir le problème d'ordonnancement comme suit :

« L'ordonnancement, de manière générale, est l'affectation des processeurs de P et (éventuellement) des ressources de R aux tâches de T dans le but de réaliser toutes les tâches en respectant les contraintes » Blazewicz et al., (1996).

Lors de la recherche d'un ordonnancement, on en découvre généralement plus d'un répondant aux critères sélectionnés. En particulier, lorsque l'on recherche un ordonnancement de makespan minimal, il y en a généralement de nombreux ayant le même makespan. L'intérêt des classes d'ordonnancement est de limiter la recherche à un sous ensemble des ordonnancements dont on sait qu'ils sont meilleurs que les autres non explorés. Suivant les buts recherchés, par exemple en termes de degré d'exploitation des machines, on a trois catégories d'ordonnancement Baker, (1974) :

- *ordonnancement sans délai*, un ordonnancement O est dit sans délai lorsqu'aucune machine n'est restée inoccupée alors que dans le stock d'entrée se trouve une tâche en attente ;
- *ordonnancement actif*, un ordonnancement O est dit actif si tout décalage à gauche oblige à retarder l'exécution d'une autre opération ou à violer une contrainte.
- *ordonnancement semi actif*, un ordonnancement O est dit semi-actif si on ne peut pas décaler à gauche aucune opération sans modifier l'ordre de traitement des jobs sur les ressources (machines). En d'autre terme un ordonnancement O est dit semi-actif si on ne peut décaler à gauche aucune opération sans modifier l'ordre de traitement des jobs sur les ressources (machines) ce qui oblige à retarder l'exécution d'une autre opération ou à violer une contrainte.

La figure 3-1 présente les relations entre les classes d'ordonnancements. Les ordonnancements sans délai sont inclus dans les ordonnancements actifs qui sont inclus dans les ordonnancements semi-actifs et qui sont tous réalisables. La position de l'ensemble des ordonnancements optimaux dépend des problèmes. La figure 3-1 est décomposée en deux parties correspondant aux deux cas de figure possibles (A et B). En A, l'ensemble des solutions optimales coupe l'ensemble des solutions sans délai alors qu'il ne le coupe pas en B. En effet, on

montre qu'il existe au moins un ordonnancement actif optimal qui peut être obtenu par décalages à gauche successifs d'une solution optimale.

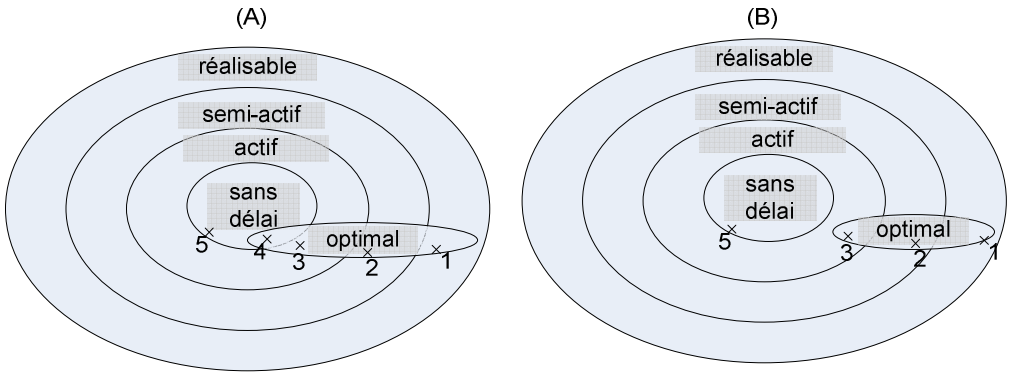


Figure 3-1 : Classes d'ordonnancements et optimalité Caumond (2006)

Une description détaillée est disponible dans Artigues et al., (2005)et Caumond (2006).

3.2. Les éléments d'un problème d'ordonnancement

Un ordonnancement est décrit, par les éléments qui le constituent. Quatre éléments sont retenus pour décrire d'une manière explicite un ordonnancement, ces éléments sont : les **tâches**, les **ressources**, les ***contraintes*** et les **critères** à prendre en considération lors du processus d'optimisation. Dans ce qui suit, on donnera une définition détaillée de chacun de ces éléments.

3.2.1. Les tâches

Une tâche (task en l'anglais) fait référence à l'exécution d'une opération, cette dernière étant caractérisée par des informations pertinentes nécessaires à la résolution du problème étudié. Ces informations peuvent être différentes selon l'objectif de l'étude et le point de vue. A titre d'exemple, le planneur s'intéressera au temps nécessaire pour réaliser l'opération sur une machine donnée, au type de ressources que cette opération consomme et à la quantité, tandis que l'opérateur va lui s'intéresser au procédé de réalisation de l'opération.

Dans le cadre des problèmes d'ordonnancement, parmi les informations utiles, on trouve le temps nécessaire pour la réalisation d'une tâche, la(les) ressource(s) qu'elle requiert (une ressource étant renouvelable ou non), une date de début au plus tôt, une date de fin souhaitée et une date de fin impérative. Des contraintes de précédence conditionnent l'exécution d'une tâche par rapport à la fin d'exécution d'autres tâches. Une tâche est alors considérée comme l'exécution d'une opération sur une certaine machine. Plusieurs taches peuvent être regroupées pour constituer un *« job »* pour lequel on donne la gamme de production. Il est également nécessaire de connaitre la nature de la tâche, par exemple savoir si une tâche est interruptible ou non, c'est-à-dire de savoir si le traitement d'une tâche sur une ressource donnée peut s'interrompre pendant une certainne durée et de le poursuivre ensuite, cela se traduit par l'autorisation de la préemption des tâches ou non. Les notations suivantes sont introduites pour décrire une opération. Chaque variable, représentera une information particulière se rapportant à la réalisation de l'opération :

- r_i pour représenter la date de disponibilité de la tâche i (release date en anglais) ;
- t_i pour représenter la date de début de la tâche i (start date en anglais) ;

- c_i pour représenter la date de fin d'exécution de la tâche i (completion time en anglais) ;
- d_i pour représenter la date d'échéance de la tâche i (due date en anglais) ;
- p_i pour représenter la durée opératoire de la tâche i (processing time date en anglais) ;
- $F_i = c_i - r_i$ pour représenter la durée de séjour de l'opération i sur la machine avant qu'elle redevienne disponible (flow time en anglais) ;
- $L_i = c_i - d_i$ exprime le retard algébrique (*lateness*) entre la fin d'exécution de la tâche i par rapport à sa date d'échéance ;
- $T_i = \max(L_i, 0)$ qui exprime le retard absolu de la tâche i (tardiness) ;
- $E_i = \max(-L_i, 0)$ qui exprime l'avancement (earliness) de la tâche i ;

Ces variables sont notamment retenues dans la littérature pour représenter une opération. La Figure 3-2 montre les relations entre les différentes variables représentant une tâche.

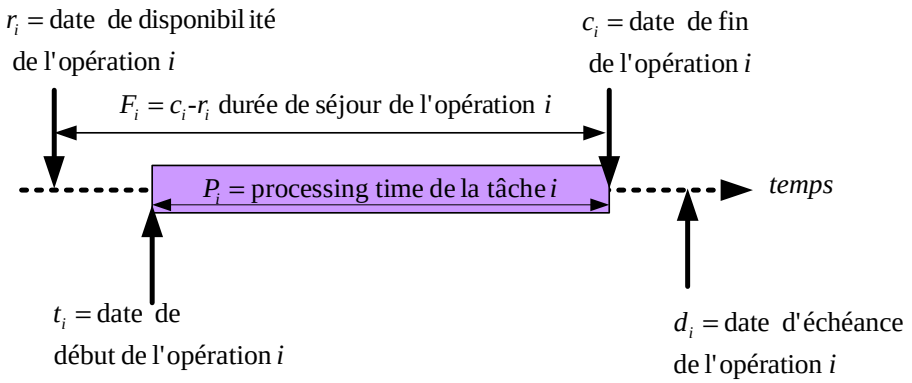


Figure 3-2 : Caractéristiques d'une tâche faisant référence à l'exécution d'une opération

3.2.2. Les ressources

Deux familles de ressources existent :

- *la première famille de ressources* est dite renouvelable c'est-à-dire que ces ressources sont réutilisables après la fin d'exécution des opérations. Cette famille est constituée de machines (processors en anglais) d'hommes et des outils de productions. Souvent des ressources additionnelles sont considérées comme par exemple l'utilisation des robots, ou des moyens de transport, ou des ressources d'un autre genre comme les buffers d'entrée/sortie. Parmi ces ressources on distingue aussi, des ressources disjonctives qui ne peuvent exécuter qu'une seule opération à la fois et des ressources cumulatives qui peuvent exécuter plusieurs opérations à la fois c'est le cas par exemple des machines parallèles ; notons qu'une machine fait partie de l'ensemble des ressources renouvelable : une machine de capacité 1 est une ressource disponible en un unique exemplaire.
- *la deuxième famille* est dite consommable et regroupe toutes les ressources qui deviennent indisponibles après une utilisation, c'est le cas des matières premières de l'argent, de l'énergie etc.

3.2.3. Les contraintes

Selon Lopez et Esquirol (1999) les contraintes expriment les restrictions que peuvent prendre conjointement les variables de décision. Donc les contraintes nous renseignent sur les limites imposées par l'environnement. On distingue plusieurs types de contraintes, qui sont (Baptiste (1998), Lopez et Esquirol (1999)) :

- *les contraintes de localisation temporelles*, Pour la réussite d'un projet ou d'un plan de production, on doit se soumettre aux impératifs de production (réalisation) traduits par le respect d'un échéancier fixé auparavant. Au niveau global on parlera d'une date de lancement d'un projet et d'une date de livraison. A un autre niveau de détail plus fin, pour une tâche ou une opération. Ce dernier niveau se traduit par la définition des dates suivante : *(i)* date de disponibilité : r_i ; *(ii)* date d'échéance : d_i .
- *les contraintes de précédence*, une contrainte qui lie le début d'une activité à la fin d'une autre est appelée contrainte de succession ou de précédence. Les gammes opératoires sont un exemple de ces contraintes, d'autres contraintes sont imposées dans certain cas telles que les contraintes de synchronisation, de simultanéité, ou de recouvrements etc. Lopez et Esquirol (1999).
- *les contraintes de ressources*, Les contraintes de ressources représentent le fait que les activités requièrent tout au long de leur exécution une certaine quantité de ressources Baptiste (1998). Les contraintes de ressources concernent les deux points suivants Lopez et Esquirol (1999) : *(i)* l'utilisation de ces ressources, (leur nature, la quantité nécessaire, et les caractéristiques d'utilisation) ; (ii) la disponibilité et la quantité de ces ressources.

Ces contraintes nous donnent une information précise sur la nature de l'atelier, et influencent le choix des méthodes d'optimization ;

Il est important de noter qu'en fonction des objectifs de l'étude, ces contraintes peuvent être strictes ou non. Selon Letouzey (2001), lorsqu'elles sont strictes, elles constituent des obligations à respecter. Lorsqu'elles ne sont pas strictes, on peut ne pas les satisfaire et on les appelle alors des contraintes de préférence.

3.2.4. Les critères d'optimisation

Les critères constituent les objectifs qu'on souhaite optimiser. En fonction du nombre d'objectifs on parle d'optimisation mono-objectif (on s'intéresse à l'optimisation d'un seul critère) et d'optimisation multi-objectifs (on s'intéresse à l'optimisation de plusieurs critères). L'optimisation s'exprime par une fonction de minimisation ou maximisation de ces critères. Dans Hentous (1999) on trouve la classification suivante :

- Les critères liés aux dates de fin de livraison ;
- Les critères liés aux volumes des encours ;
- Les critères liés à l'utilisation des ressources.

3.2.4.1. Les critères liés aux dates de fin de livraison

Les critères liés aux dates de fin de livraison constituent la catégorie des critères les plus étudiés en optimisation : on retrouve le *makespan* $C_{\max} = \max(c_i)$ qui représente la fin d'achèvement de toutes les tâches du problème.

Les notations ci-dessus sont également utilisées pour la définition de différents problèmes d'ordonnancement liés aux dates de fin :

- $C_{\sum} = \sum(c_i)$ représente la somme des dates de fin ;
- $\sum w_i C_i$ pour représenter la somme pondérée des dates de fin ;
- $\overline{C}$ pour représenter la moyenne arithmétique des dates de fin des tâches ;
- $F_{\max} = \max(F_i)$ pour représenter le temps maximal de séjour des tâches ;
- $L_{\max} = \max(l_i)$ pour représenter le maximum des retards algébriques de l'ensemble des tâches ;
- $T_{\max} = \max(T_i)$ pour représenter le maximum des retards absolus de toutes les tâches ;
- $E_{\max} = \max(E_i)$ pour représenter le maximum des temps d'avancement de toutes les tâches ;

On retrouve aussi d'autres critères comme les temps totaux, tels que $F_{\sum}, L_{\sum}, T_{\sum}, E_{\sum}$ pour représenter respectivement pour l'ensemble des tâches, le temps total de séjour, le retard total, le retard total absolu, et l'avancement total.

3.2.4.2. Les critères liés aux volumes des encours

Dans cette catégorie on s'intéresse au nombre de tâches en cours d'exécution à chaque instant t et on cherche généralement à minimiser le nombre de tâches en attente ou à maximiser le nombre de tâches terminées sur la dernière machine. Pour chaque opération on définit t_0 comme sa date de début d'exécution et p_0 son processing time. Ainsi il est possible de définir :

- *le nombre de tâches* en exécution à un instant t est donné par :
- $$N_p(t) = \sum_i^n e_i(t) \text{ avec } e_i(t) = \begin{cases} 1 \, si \, (t_0 \le t \le t_0 + p_0) \\ 0 \text{ sinon} \end{cases}$$
- *le nombre de tâches* en attente d'exécution est donné par :
- $$N_w(t) = \sum_i^n e_i(t) \text{ avec } e_i(t) = \begin{cases} 1 \, si \, (t_0 + p_0 \le t \le t_0 + 1) \\ 0 \text{ sinon} \end{cases}$$
- *le nombre de tâches terminées* sur la dernière machine est donné par :
- $$N_f(t) = \sum_i^n e_i(t) \text{ avec } e_i(t) = \begin{cases} 1 \, si \, (C_i \le t) \\ 0 \text{ sinon} \end{cases}$$

3.2.4.3. Les critères liés à l'utilisation des ressources

Les critères liés à l'utilisation des ressources permettent de satisfaire un objectif donné et d'autre part d'évaluer des critères de performance. Par exemple, pour les SFP on retrouve une panoplie de critères tels que :

- *maximiser l'utilisation de* : la machine la moins gourmande en énergie, de la machine ayant un bon rendement ou de la machine la plus simple à entretenir ;
- *minimiser la charge* d'une machine ou *le temps d'inactivité* de l'ensemble des machines ;
- *minimiser l'écart moyen* d'utilisation de l'ensemble des machines, critère qui permet d'équilibrer l'utilisation des machines ;

- …

On trouve également des critères de performance tels que Hentous (1999) :

- *l'utilisation moyenne des ressources* donnée par $\overline{U} = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} \right) \Big/ \left(\sum_{j=1}^{m} M_J \times C_{\max} \right)$, ce critère permet de mettre en évidence les périodes creuses et les périodes pleines de l'utilisation du système de production et ainsi de renseigner sur les taux d'utilisation des ressources.
- *le temps d'inactivité de l'atelier* $I = \sum_{j=1}^{m} \left(C_{\max} - \sum_{i=1}^{n} p_{ij} \right)$ correspondant à l'ensemble des tranches de temps perdus (en attente de travail (*idle time*)) pendant l'exécution des tâches. C'est un indicateur de l'utilisation des capacités de l'atelier qu'il faut minimiser pour obtenir une meilleure utilisation des ressources Hentous (1999).

La combinaison entre les différents critères est possible : maximisation des profits et minimisation de tous les coûts inhérents à la production. On note aussi que des similitudes et des équivalences entre critères et que des réductions permettent de déduire certains critères de certains d'autres. La Figure 3-3 donne les relations entre certains critères : le sens des flèches donne le sens de réduction avec la condition associée sur chaque arc.

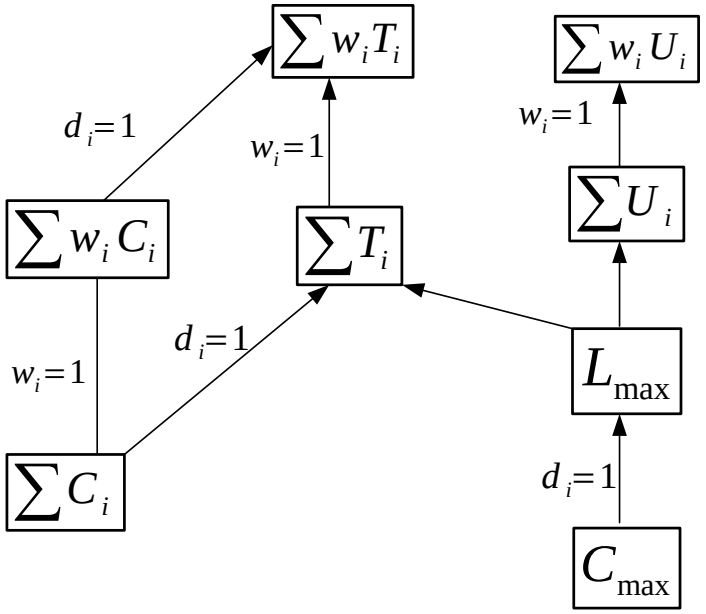


Figure 3-3 : Relation entre les critères d'optimisation Pinson (1995)

La section qui va suivre donne une notation des problèmes d'ordonnancement. Cette notation sera utile pour la compréhension et la classification des problèmes d'ordonnancement. Cette notation s'appuiera sur les trois éléments précités (tâches, ressources et critères).

3.2.5. Notation et définition des problèmes d'ordonnancement

L'introduction d'une classification des problèmes d'ordonnancement est justifiée par la grande variété de ces derniers et par la nécessaire homogénéisation. La description explicite de

ces trois éléments tâches, ressources et critères a donné naissance à une notation des problèmes d'ordonnancement. Ainsi c'est en 1979 qu'une notation a été proposée pour la première fois. Il s'agit le la notation $\alpha|\beta|\gamma$. Cette notation a été introduite par Graham et al., (1979) et fut reprise la même année par plusieurs auteurs dont Lageweg, Lawler, Lenstra, Rinnooy Kan dans leur article Lageweg et al., (1981) puis par toute la communauté scientifique du domaine. Cette notation constituée de 3 paramètres ayant chacun une signification différente : le paramètre α nous renseigne sur l'environnement d'usinage (manufactoring environment specification), le type d'atelier de production et le nombre de machines présentent dans le système, tandis que le champ β concerne les caractéristiques des problèmes d'ordonnancement et regroupe les contraintes imposées par l'environnement de production et par les ressources. Le champ γ concerne les critères de performance qui vont représenter les objectifs à atteindre pour le problème en question. Le Tableau 3-1, donne les différentes valeurs possibles pour les 3 paramètres $\alpha|\beta|\gamma$ de reprendre et d'étendre les propositions faites dans Varela et al., (2002). Les notations sont utilisées comme suit : les trois champs sont séparés par le caractère « | » et à l'intérieur de chaque champ on sépare les différents sous-champs par une virgule ; si un champ n'a pas de valeurs il est simplement omis.

Exemple des notations :

- ***Le job-shop*** : soit un problème de type job-shop où on a besoin de traiter 5 jobs sur 4 machines, avec les contraintes de précédence entre les opérations. Il se note comme suit : $\alpha_1 = J, \alpha_2 = 5|\beta_2 = prec, \beta_7 = 5|\gamma = C_{\max}$ la forme la plus simple est $J\|C_{\max}$, car les informations les plus pertinentes sont le type d'atelier et le critère à optimiser ; $\alpha_1 = J, \alpha_2 = 5|\beta_2 = prec, \beta_7 = 5|\gamma = C_{\max}$ est une instance particulière de $J\|C_{\max}$.
- ***Le Flow-shop*** : $F4|p_{ij} = p|\sum C_j$ représente un problème de Flow-Shop avec 4 étages $F4$. Tous les processing time valent ***P*** et ceci est exprimer par $p_{ij} = p$. $\sum C_j$ exprime le fait qu'on s'intéresse à la somme des dates de fin de tous les jobs.
- ***L'Open-shop* avec plusieurs robots identiques**: $O|p_{ij} = 1, tree, r_i; no-wait|C_{\max}$, O pour désigner qu'il s'agit d'un problème d'Open-Shop ; tous les processing time valent ***1*** et ceci est exprimer par $p_{ij} = 1$, la présence de r_j signifie que tous les jobs ont des dates de début au plus tôt et des dates de fin au plus tard. $no-wait$ pour signifier que les temps d'attente ne sont pas tolérés;
- ***Le job-shop* avec plusieurs robots différents** peut être noté : $J, MPR|t^r_{jkl}, t'^r_{kl}|C_{\max}$ où J désigne qu'il s'agit d'un problème de Job-Shop et MPR pour désigner l'utilisation de plusieurs robots différents ***r*** (***M***ulti-***P***urpose ***R***obots) et t^r_{jkl} pour représenter le fait que les temps de transport dépendent à la fois du job transporté, des machines et du robot utilisé ; t'^r_{kl} représente le fait que les déplacement à vide dépendent des machines et du robot utilisé. On s'intéresse à l'optimisation de la date de fin de la dernière opération.
- ***Le Flow-shop*** : $F4, R1|p_{ij} = p, r_j, t_j|\sum C_j$, représente un problème de Flow-Shop avec 4 étages $F4$ avec un seul robot $R1$. Tous les processing time valent ***P*** et ceci est exprimer par $p_{ij} = p$, la présence de r_j signifie que tous les jobs ont des dates de début au fin au plus tard. t_j signifie que les temps de transport dépendent des jobs et

non des machines. $\sum C_j$ permet d'exprimer le fait qu'on s'intéresse à la somme des dates de fin de tous les jobs.

Tableau 3-1 : Description détaillée de la notation $\alpha|\beta|\gamma$

Classe	**Factor**	**Description**	**Description en langue française**	**Valeur***
α	α_1	Manufacturing system type	Type d'atelier	$P, Q, R, X, G, O, J, F,\ PMPM, ...$
	α_2	Number of machines	nombre de machine	O, K
	α_3		Extension du champ α pour l'adapter à chaque problème	
	α_4		Extension du champ α pour l'adapter à chaque problème	
β	β_1	Job/operation preemption	Job/opération avec préemption	$O,\ pmtn$
	β_2	Precedence constraints	Contraintes de précédence	$prec, chain, tree, sp-graph$
	β_3	Ready times		O, r_j
	β_4	Restrictions on processing times	restriction sur les temps opératoires des jobs	$p_j = 1, p_{ij} = 1, p_j = p, p_{\inf} \le p_j \le p_{\sup}, ...$
	β_5	Due dates (deadlines)	date au plus tôt et date au plus tard	O, d_j
	β_6	Batches/families processing	traitement par lot ou par catégories	$O, batch$
	β_7	Number of jobs or tasks in a job	nombre de jobs ou nombre de tâches dans un job	O, n_j O
		(job shop case)	(cas de job shop)	
	β_8	Job/task priorities	Job/tâches avec priorités	O, w_j
	β_9	Dynamic machine availability	Disponibilité de machine dynamique	$O, avail$
	β_{10}	Additional/auxiliary resources	ressources additionnelle ou auxiliaires	O, aux
	β_{11}	Buffers	buffers	$O, no\ wait$
	β_{12}	Setup (changeover)	temps de reconfiguration	$O, setup*$
γ	γ	Performance measure	indice de mesure ou de performance	$C_{\max}, \sum C_{j,} \sum w_j C_{j,}\ L_{\max}, \sum T_j, ...$

3.2.6. Présentation des problèmes classiques

Tous les problèmes d'ordonnancement sont constitués d'un ensemble de tâches (jobs) à ordonnancer sur un ensemble de ressources (machines). Chaque job doit subir des traitements par une ou plusieurs ressources. La description des traitements à réaliser et leur éventuel ordre respectif sont appelés « gamme ». Suivant le type de problème d'ordonnancement étudié et l'organisation d'atelier, les gammes sont très différentes.

Les systèmes de production sont divers et variés et ont différentes organisations au niveau des ateliers de production. Ces organisations déterminent les problèmes d'ordonnancements, ainsi on trouve :

- **Les organisations à ressource unique** : c'est l'organisation, la plus simple, composée d'une seule machine, (un seul processeur). Par contre elle constitue un domaine de recherche pour les systèmes informatique où on partage souvent une seule ressource appelée ressource critique comme par exemple, un seul écran, un seul processeur une seule imprimante, un seul disque réseau etc… ;
- **Les organisations à ressource multiples** : dans ces organisations, on dispose de plusieurs ressources sur lesquelles s'exécutent plusieurs tâches. Le type de cheminement des tâches donne trois sous organisations connues sous les noms de Job-Shop, Flow-Shop et Open Shop ;
- **Les organisations à multi-étages parallèles** : Ces organisations regroupent dans chaque étage des machines capables de faire les mêmes opérations. Ces machines peuvent être identiques, ou uniformes ou différentes. Les étages sont disposés d'une manière parallèle afin d'avoir plusieurs cheminements possibles en même temps. La Figure 3-4 donne un schéma général d'une organisation multi-étages composée de trois étages parallèles.

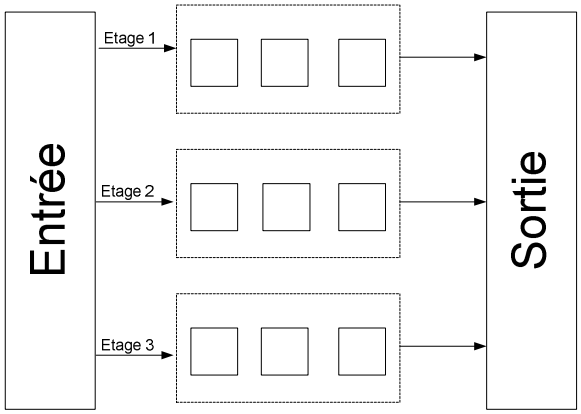


Figure 3-4 : Organisation multi-étage parallèles

Les problèmes classiques suivants ne diffèrent que par la définition de leur gamme et de leur organisation :

- **Pour le problème à une machine**, les gammes consistent à visiter l'unique machine durant un temps prédéterminé ;
- **Pour le problème de flow-shop**, Dans cette organisation, les machines sont disposées d'une manière linéaire (voir Figure 3-5). Il s'agit d'un problème d'ordonnancement avec n jobs et m machines. Les gammes opératoires sont toutes identiques, cela signifie que l'ordre de passage sur les machines est le même pour toutes les pièces, mais les temps de séjour peuvent être différentes. Le problème consiste à trouver un ordre de passage des pièces optimal, c'est-à-dire un ordre

permettant de minimiser la date de fin au plus tard de la dernière opération toutes pièces comprises. Il existe aujourd'hui des algorithmes permettant de résoudre des problèmes avec 2 machines de manière exacte et efficace (dans un temps polynomial). Au delà, il convient de passer aux méthodes approchées, telles que les métaheuristiques pour obtenir un résultat satisfaisant. La Figure 3-6 montre un flow-shop composé de 3 machines.

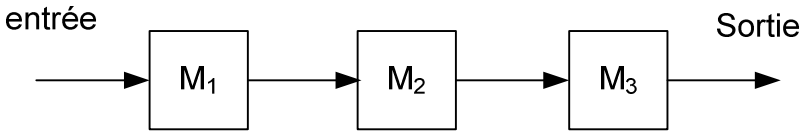


Figure 3-5 : Flow-shop à trois machines

- **Pour le problème de flow-shop hybride**, tous les jobs ont la même gamme. Ce problème est une extension du problème du flow-shop. Il est composé de ***m*** étages. L'étage *j (j=1,m)* est composé de M_j machines parallèles identiques, proportionnelles ou non identiques. La Figure 3-6 montre un Flow-Shop Hybride constitué de 3 étages. Les jobs doivent être traités sur une seule machine de chaque étage. L'ordre des étages est le même pour tous les produits (étage 1, étage 2, …, étage m). Au problème d'ordonnancement des jobs s'ajoute celui de l'affectation des jobs aux machines sur les différents étages.

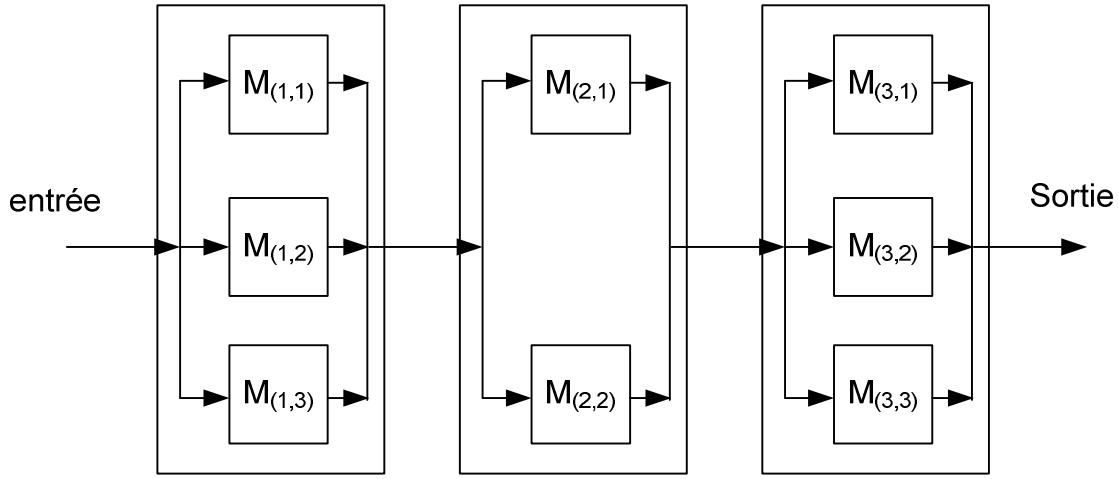


Figure 3-6 : Flow-Shop Hybride constitué de 3 étages

- **Pour le problème de job-shop**, chaque job a sa propre gamme. Une gamme consiste à visiter un ensemble de machines dans un ordre donné. Le Job-Shop ou **JSSP** pour (Job-Shop Scheduling Problem) est plus général que le Flow Shop car chaque job possède une gamme opératoire différente. Cela signifie qu'un problème de type Job Shop comporte n jobs à exécuter sur m machines. Les problèmes de type Job-Shop sont parmi les plus étudiés dans la littérature à cause de leurs ressemblances avec les problèmes réels ;
- **Pour le problème d'OpenShop**, chaque job a également sa propre gamme. Une gamme consiste alors à visiter un ensemble de machines dans un ordre à déterminer. En d'autres termes dans le problème de l'Open Shop, les gammes opératoires ne sont pas spécifiées. Seules le sont les différentes opérations à réaliser ainsi que leur différents temps de traitement. Typiquement cela peut se traduire par une fonction

qui peut être réalisée par différentes machines ou encore des pièces qui n'ont pas besoin de suivre un ordre précis de traitement.

- **Pour ce qui concerne le job-shop flexible**, on donne pour chaque opération un sous-ensemble de machine sur lesquelles l'opération peut être réalisée avec des temps de séjours souvent différents. La difficulté consiste alors à choisir le meilleur ordre et les meilleures affectations des machines aux opérations afin de minimiser le temps.
- **Pour le problème à machines parallèles**, tous les jobs ont la même gamme. Une gamme consiste à visiter une des machines parallèles, le choix de la machine est à déterminer.
- **Pour le problème du RCPSP** (Resource Constrained Project Scheduling Problem), chaque job doit réaliser une opération en consommant des ressources. Les ressources sont disponibles en nombre limité et sont typées, i.e. on ne peut pas, à priori, utiliser une ressource à la place d'une autre. On distingue les ressources renouvelables, qui sont disponibles pour une nouvelle opération après la fin d'une opération, et les ressources non renouvelables qui ne peuvent être utilisées qu'une seule fois.

La Figure 3-7 montre les relations qui existent entre les différentes organisations, et donne une vue générale de la typologie des problèmes d'optimisation.

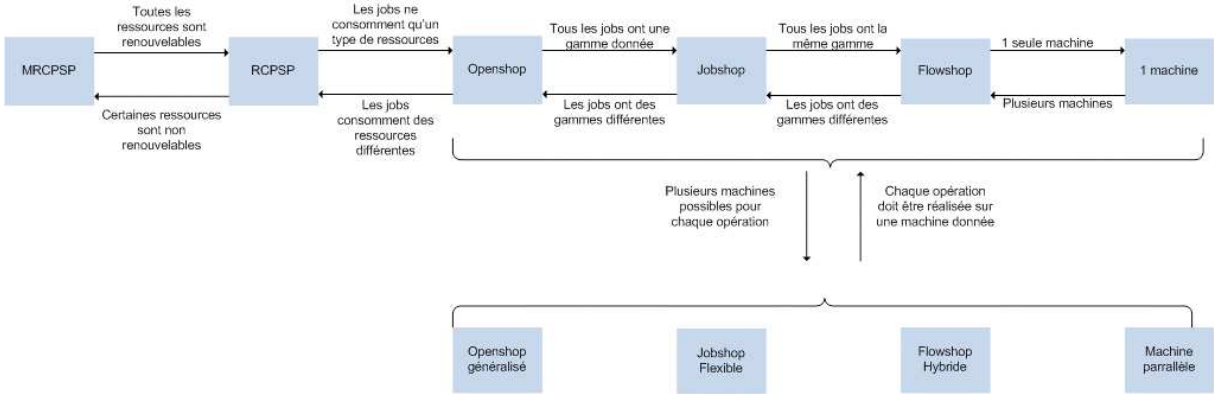


Figure 3-7 : Typologie des problèmes d'ordonnancement Bertel (2001)

Parmi les problèmes d'ordonnancement présentés, nous nous sommes particulièrement intéressés au problème de job-shop car il généralise la plupart des modèles d'atelier. Depuis la deuxième moitié des années 60, ce problème suscite de très nombreux travaux. Entre autres, des algorithmes d'optimisation particulièrement efficaces ont été développés.

Le problème de job-shop pose les hypothèses simplificatrices suivantes : capacité de stockage suffisante (considérée infinie), temps et ressources de transport négligés, disponibilités complètes des pièces et des machines, temps de traitement des opérations connus et déterminés. Mais ces hypothèses sont parfois trop restrictives, et des modèles plus complexes doivent souvent être envisagés. En particulier, les problèmes présentés dans le paragraphe 2 peuvent être vus comme des problèmes de job-shop avec extensions : les Systèmes Flexibles de Production se modélisent comme un problème de job-shop avec transport.

A priori, il n'est pas possible de réutiliser directement les algorithmes d'optimisation efficaces pour le job-shop. Ces algorithmes doivent être réadaptés pour prendre en compte des contraintes supplémentaires. Dans la suite, nous proposons une démarche d'optimisation et une démarche de modélisation qui permettent de construire un algorithme d'optimisation tout en tirant partie des algorithmes efficaces.

3.3. Modélisation des problèmes d'ordonnancement

Pour résoudre un problème d'optimisation P lié à un ordonnancement, il faut toujours, passer par une phase d'analyse. Ceci permet d'élaborer un modèle de fonctionnement $\mathcal{M}$, *cette démarche s'appelle la modélisation.* Plusieurs définitions existent pour le terme modèle, parmi ces définition, on a choisi deux d'entre elles que nous jugeons être les plus pertinentes.

3.3.1. Modèle

Selon Ferber (1995) :

« Un modèle, en science, est une image stylisée et abstraite d'une portion de la réalité /../.Ces modèles peuvent être très formels ou simplement fournir une représentation simple et pratique /../. L'intérêt d'un modèle est d'abord d'être plus explicite, plus simple et plus facile à manipuler que la réalité qu'il est sensé représenter. Les modèles éliminent ainsi un grand nombre de détails considérés comme inutiles par le modélisateur afin de mieux se consacrer aux données que celui-ci juge pertinentes relativement au problème qu'il désire résoudre /../. »

Les modèles sont ainsi *des images homomorphes de la réalité,* c'est à dire qu'il existe un homomorphisme entre l'objet d'étude et le modèle qui permet d'appliquer les résultats du modèle à l'objet lui même comme le montre la Figure 3-8. Les modèles sont donc plus que nécessaires pour cerner et représenter le domaine d'étude selon un certain nombre de critères.

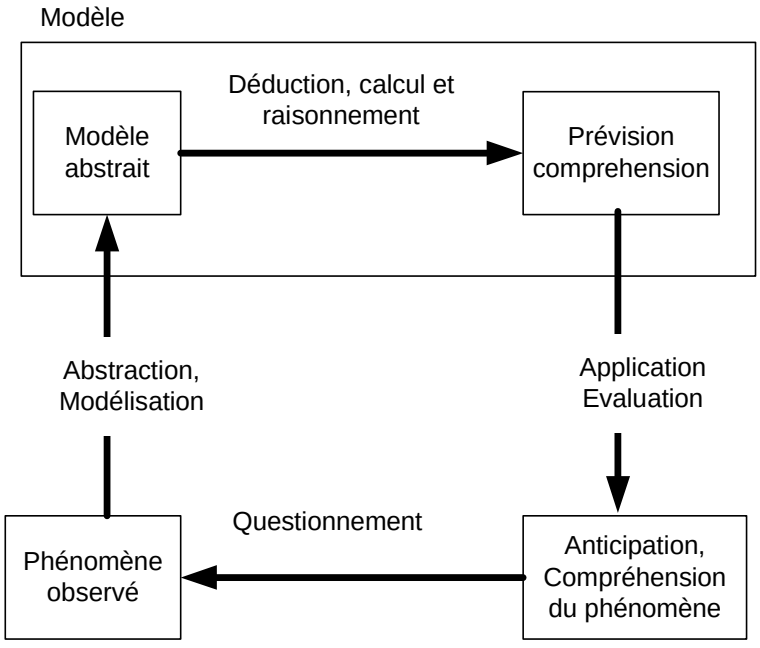


Figure 3-8 : La prévision et la compréhension de phénomène passe par l'élaboration de modèles Ferber (1995)

Minsky donne une définition de la notion de modèle Minsky (1968).

« Pour un observateur A, β *est un modèle du système* B *si* A *peut, à partir de* β *apprendre quelque chose d'utile sur le fonctionnement de* B *»*

Ces deux définitions se rejoignent sur le fait qu'un modèle est une représentation simplificatrice de l'univers réel. La démarche d'optimisation a souvent recours aux modèles de base les plus simples possibles pour résoudre des problèmes d'optimisation combinatoire. Une fois ce dernier maitrisé, on revient en amont avec une démarche incrémentale pour ajouter des extensions ignorées dans le modèle de base.

Ces extensions sont souvent l'effort de toute une communauté de chercheurs, fonctionnant sur le principe de réutilisabilité. L'adoption d'autres modèles pour le même problème représente une proportion faible par rapport au nombre de réutilisation et d'extension de modèles. Dans la suite de ce document et pour chaque problème étudié, on s'efforce de donner le modèle du problème ainsi que les extensions possibles.

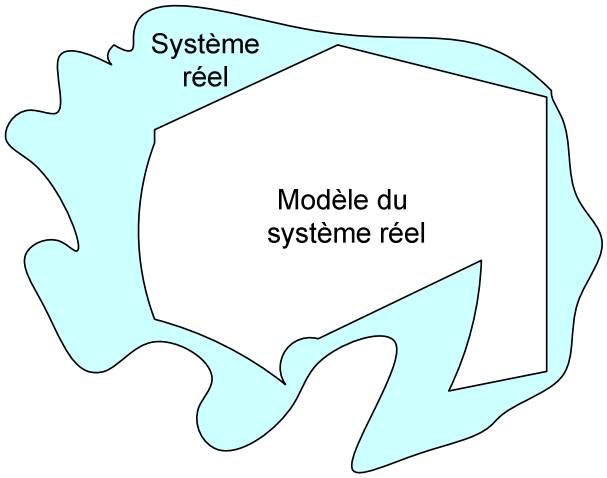


Figure 3-9 : Représentation simplifiée d'un modèle et son système

3.3.2. Processus de modélisation

La construction d'un modèle fiable est un processus laborieux et non trivial. Dans ce contexte pour réaliser un modèle, il est nécessaire de suivre un processus de modélisation. Pour y parvenir, nous en retenons un processus de modélisation basé sur la construction consécutive d'un modèle de connaissance (modèle de description du fonctionnement du système) et d'un ou plusieurs modèle(s) d'action (modèles informatique) Gourgand (1984). Ce processus comprend quatre étapes :

- Analyse et spécification ;
- Elaboration d'un modèle d'action ;
- Exploitation des résultats du modèle d'action ;
- Prise de décision et agir sur le système.

Dans la première étape on va concentrer tous les efforts pour comprendre le fonctionnement du système. Ceci a pour but l'élaboration d'un modèle de connaissance du système réel en passant par l'analyse de ce dernier afin de dégager toutes les informations pertinentes caractérisant le système. Puis en deuxième étape, on s'atèle à produire un modèle d'actions qui a pour but de résoudre le problème. L'élaboration de ce modèle consiste à la

traduction du modèle de connaissance dans un formalisme choisi (modèle mathématique, modèle de simulation dans un langage approprié, …).

En troisième étape on exploite le modèle d'action. Le réglage de méthodes proposées ainsi que l'exploitation des résultats sont fait à ce moment là.

Dans la quatrième étape on retrouve tous les mécanismes de prise de décisions et les éventuelles actions sur le système. La Figure 3-10 nous donne un aperçu du processus de modélisation. Tchernev (1997) et Gourgand et Tchernev (1998) propose un modèle fonctionnel du processus de modélisation.

Nous nous servons de ce processus comme cadre méthodologique pour proposer des solutions aux problèmes traités dans les chapitres suivants.

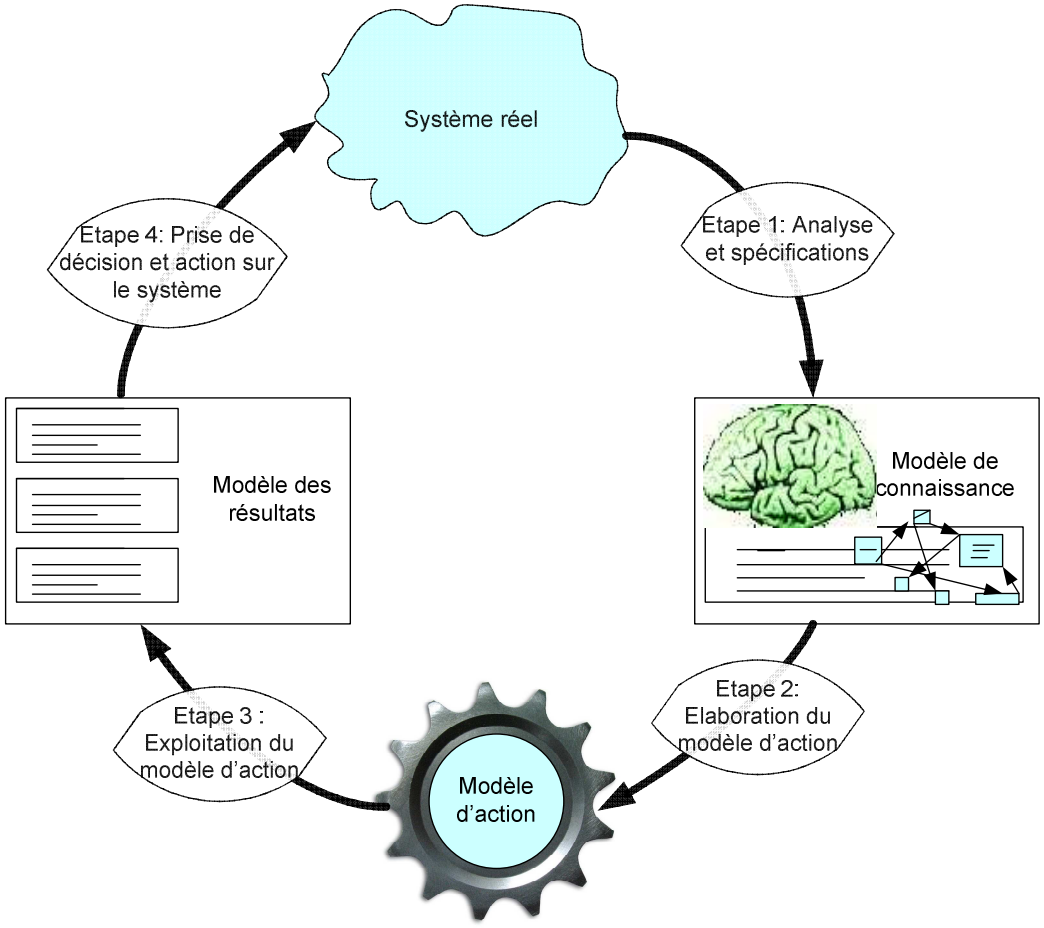


Figure 3-10 : Représentation simplifiée du processus de modélisation

3.4. Conclusion

Dans cette partie, nous avons présenté les problèmes d'ordonnancement. Après avoir donné la définition de l'ordonnancement et les différents types d'ordonnancement, nous avons décrit les éléments d'un problème d'ordonnancement et les critères d'optimisation. En se basant sur ces éléments nous avons présenté la notation pour les problèmes d'ordonnancement. Cette

notation basée sur trois paramètres ($\alpha|\beta|\gamma$) permet de décrire de manière non ambigu le problème étudié.

Nous avons présenté également le processus de modélisation utilisée qui permet d'aborder des problèmes complexes. Pour chaque problème d'ordonnancement dans les SFP, la démarche est la même. Moyennant des hypothèses simplificatrices, nous nous ramenons à un problème théorique. Face à la complexité, nous proposons alors des extensions de ce modèle théorique et des méthodes de résolution.

Dans la partie suivante nous présentons les problèmes d'optimisation.

4. Présentation des problèmes d'optimisation

4.1. Introduction

L'optimisation est une activité qui consiste à améliorer les procédés ou les techniques utilisées afin de les rendre plus performant. On la trouve dans tous les champs d'activité humaine. Cependant elle se heurte à des obstacles de taille. Cette situation se rencontre particulièrement lorsqu'on a à faire à des applications qui font intervenir plusieurs paramètres. La combinaison de ces derniers donne ce que l'on appelle communément l'explosion combinatoire qui rend difficile la recherche d'une solution. De nombreuses applications actuelles font face à ce problème. La recherche des solutions optimales est un processus très difficile dés que l'espace des solutions devient grand rendant ainsi l'énumération de ces dernières hors d'atteinte dans des temps raisonnables. Si on ne peut pas fournir une solution optimale à un problème, on se contente généralement de solutions approchées que l'on considère bonnes ou satisfaisantes sous certaines conditions. Cette situation apparait notamment dans le cas, des problèmes d'optimisation combinatoire *POC* (« Combinatorial Optimisation Problems » ou *COPs*). Dans la suite de ce chapitre, nous donnerons la notion de *POC*, et nous introduirons la notion de complexité.

4.2. Les problèmes d'optimisation combinatoire

On distingue les *POC* ou *COPs* des autres problèmes d'optimisation, plusieurs définitions existent dans la littérature. Nous les présentons de manière sommaire.

Définition (a) Garey et Johnson (1979)

Un *POC* est soit un problème de maximisation soit un problème de minimisation, qui se caractérise par les trois propositions suivantes :

- Un ensemble d'instances ;
- Un ensemble fini de solutions pour chaque instance ;
- Une fonction qui associe à chaque solution un nombre rationnel positif pour chaque solution $x \in S_p(I)$.

Sous ces conditions, une solution optimale pour une instance $I \in D_p$ est une solution $x^* \in S_p(I)$ telle que, pour $x \in S_p(I)$, on a $H(I,x^*) \leq H(I,x)$ si P est un problème de minimisation, et $H(I,x^*) \geq H(I,x)$ si P est un problème de maximisation.

Définition (b) Lacomme et al., (2003)

Un problème d'optimisation combinatoire (*POC*) consiste à chercher le minimum s^* d'une application f, le plus souvent à valeurs entières ou réelles, sur un ensemble fini S : $f(s^*) = \min_{s \in S}\{f(s)\}$.

Pour éviter des discussions sur les problèmes de précision, on suppose, sauf exception, que f est à valeurs entières. Par exemple, mesurer des durées de tâches avec des nombres entiers de secondes offre une précision largement suffisante pour un problème d'ordonnancement industriel. f est la fonction économique (objective function, cost function). La définition concerne la minimisation, mais il suffit de remarquer que maximiser f équivaut à minimiser $-f$. Parfois on s'intéresse seulement aux éléments de S vérifiant certaines contraintes, ce sont les solutions réalisables (feasibles solutions).

On peut reformuler cette définition de la manière suivante :

Définition (c)

Un problème d'optimisation combinatoire ***(POC)*** P *consiste à chercher le minimum ou le maximum* s^* *d'une application* f *, le plus souvent à valeurs entières ou réelles, sur un ensemble fini* S *tel que :*

$f(s^*) = \min_{s \in S}\{f(s)\}$ *si* P *est un problème de minimisation ;*

$f(s^*) = \max_{s \in S}\{f(s)\}$ *si* P *est un problème de maximisation.*

4.3. La théorie de la complexité

La comparaison entre deux approches ou algorithmes est souvent une chose délicate, car on se contentait au départ de ne tenir compte que du temps d'exécution pour décider de la performance des deux approches. Or cette manière est relativement biaisée car, d'autres paramètres rentrent en jeu et pas des moindres qui sont : la vitesse d'horloge (fréquence de calcul) de la machine sur laquelle s'exécutent l'une ou l'autre approche, le langage utilisé, le système d'exploitation tournant sur la machine, ainsi que la taille mémoire disponible. Pour une comparaison objective, il est impératif d'adopter des critères indépendants du matériel et des logiciels utilisés pour tester l'une ou l'autre méthode : c'est ainsi que la théorie de complexité est née.

4.3.1. La complexité algorithmique

Dans Lacomme et al., (2003) on trouve la définition suivante pour ce qui concerne la complexité d'un algorithme :

« La complexité d'un algorithme A *est une fonction* $C_A(N)$ *, donnant le nombre d'instructions caractéristiques exécutées par* A *dans le pire des cas, pour une donnée de taille* N *. »*

La complexité moyenne étant difficile à cerner, cette dernière tient en effet à comparer deux algorithmes uniquement sur le plan du nombre d'instructions qui peuvent être exécutées dans le pire des cas indépendamment de l'environnement d'implémentation (matériel et logiciel).

Pour représenter en mémoire la taille occupée par les données du problème, il faut aussi considérer les taille des données qui décrivent les données du problème, ainsi pour trier un tableau de taille *N*, la taille de la donnée doit comporter alors *N* fois, ce qu'occupe en mémoire chaque élément du tableau plus la taille qu'occupera en mémoire le nombre *N*.

Pour donner la taille d'une manière exacte, il faut que cette taille soit exprimée en bits mais, on se contente dans la plus part des cas de la donner en mots mémoire servant à sauvegarder ces données. En plus les données peuvent avoir plusieurs représentations, le nombre 16 est de taille 2 dans la représentation décimal mais sa taille sera de 4 dans une base binaire, pour éviter tout amalgame se rapportant à la taille d'une donnée, on se contente de notée par n la taille de la donnée indépendamment de sa représentation.

Réponse algorithmique :

Il s'agit de savoir si un algorithme est décidable ou calculable. Un algorithme est dit décidable lorsqu'on s'intéresse à savoir s'il existe ou non une solution au problème posé, dans ce cas, la réponse est (oui ou non). Par contre l'algorithme est dit calculable s'il consiste à trouver un élément répondant à certains critères dans un espace de recherche.

Dans Lacomme et al., (2003) on retrouve la définition suivante de l'ordre d'une fonction :

Ordre d'une fonction :

Soit f, g deux fonctions de $\Re$ dans $\Re$, on dit que f est d'ordre inférieur ou égal à g, ou d'ordre au plus g, si on peut trouver un réel x_0 et un réel positif c tels que $\forall x \geq x_0, f(x) \leq c \cdot g(x)$. En d'autres termes, g devient plus grand que f à partir d'un certain nombre x_0, à un facteur c prés. On écrit : f est $O(g)$, f est en $O(g)$, $f = O(g)$ ou $f \in O(g)$ (en interprétant $O(g)$ comme l'ensemble des fonctions d'ordre au plus g) et on prononce grand O de g.

Propriétés :

Si $\lim_{n\to\infty} \frac{f}{g} = c \succ 0$, f *et* g *sont de même ordre,* $f = O(g)$ *et* $g = O(f)$, *noté* $f = \Theta(g)$;

$\lim_{n\to\infty} \frac{f}{g} = 0$, *alors* $f = O(g)$ *mais* g *n'est pas* $O(f)$;

Si $\lim_{n\to\infty} \frac{f}{g} = +\infty$, f *est d'ordre supérieur à* g, *on note* $f = \Omega(g)$ *et* $g = O(f)$, *équivalent à* $g = O(f)$.

Ces propriétés sont intéressantes car on cherche souvent à considérer le pire cas, ce dernier constitue un majorant de la complexité réelle. $f = O(g)$ signifie que g est un majorant de la complexité réelle de f, les complexités sont souvent données en ordre près, alors un algorithme peut être en $\Theta(N^3)$ et un calcul grossier mais rapide peut évaluer la complexité à $O(N^3)$.

4.3.2. Les classes des problèmes

Le temps et l'espace ainsi que le type de la machine sur laquelle s'exécute le problème définissent quatre classes de complexité qui sont énumérées ci-dessous :

- *La classe* ***TIME(t(n))*** contient les problèmes de décision qui peuvent être résolus sur une machine déterministe en temps de l'ordre de grandeur de ***t(n)***.
- *La classe* ***NTIME(t(n))*** contient les problèmes de décision qui peuvent être résolus sur une machine non déterministe en temps de l'ordre de grandeur de ***t(n)***.
- *La classe* ***SPACE(s(n))*** contient les problèmes de décision résolus sur une machine déterministe nécessitant un espace de l'ordre de grandeur ***s(n).***
- *La classe* ***NSPACE(s(n))*** contient les problèmes qui pour être résolus nécessitent un espace de l'ordre de grandeur de ***s(n)*** sur une machine non déterministe.

Classes L et NL

Le temps requis pour résoudre les problèmes de la classe *L* évolue d'une manière *logarithmique* par rapport à la taille de ces problèmes. Avec les notations introduites plus haut, *L = SPACE (log(n))*. La classe *NL* s'apparente à la classe *L* mais sur une machine non déterministe (*NL = NSPACE(log(n)*). Par exemple savoir si un élément appartient à un tableau trié peut se faire en espace logarithmique.

Pour comprendre ce que est la complexité exponentielle on se propose de reprendre un exemple tiré de Lacomme et al., (2003). Dans cet exemple, les auteurs font l'hypothèse que la machine exécute 1 instruction de haut niveau par micro seconde c'est-à-dire une instruction toutes les 0.000001 secondes. Les cases vides ont des durées supérieur à l'âge de l'univers, estimé à 1000 milliards d'années, les machines actuelles sont ainsi impuissantes face à la complexité exponentielle malgré leur croissance des performances, en attendant l'ordinateur quantique ces limites font que les machines ne peuvent pas résoudre tous les problèmes mais néanmoins en résoudre un grand nombre.

Classe P

Cette classe contient des problèmes considérés faciles à résoudre. Les problèmes de la classe P sont résolus en temps polynomial par rapport à la taille de la donnée du problème sur une machine déterministe. Si la taille de la donnée est n, la complexité du problème sera alors $O(n^k)$ pour un certain k, on dit aussi que l'algorithme lui-même est *polynomial*. Cette classe a des propriétés très intéressantes comme par exemple : le fait que les algorithmes polynomiaux forment une classe *stable :* que la composition de deux algorithmes polynomiaux reste polynomiale, et qu'un algorithme construit polynomialement à partir d'appels à des procédures de complexité polynomiale, reste polynomial. Le Tableau 4-1 donne la croissance du temps de calcul pour les différences complexités Lacomme et al., (2003).

Tableau 4-1 : Croissance du temps de calcul pour les différentes complexités

Taille → *Complexité* ↓	**20**	**50**	**100**	**200**	**500**	**1000**
$10^3 n$	0. 02 s	0.05 s	0.1 s	0.2 s	0.5 s	0.1 s
$10^3 n \log_2 n$	0.09 s	0.3 s	0.6 s	1.5 s	4.5 s	10 s
$100n^2$	0.04 s	0.25 s	1 s	4 s	25 s	2 min
$10n^3$	0.02 s	1 s	10 s	1 min	21 min	27 h
$n^{\log_2 n}$	0.4 s	1.1 h	220 j	12500 ans	5.10^{10} ans	———
$2^{n/3}$	0.0001 s	0.1 s	2.7 h	3.10^6 ans	———	———
2^n	1 s	36 ans	———	———	———	———
3^n	58 min	2.10^{11} ans	———	———	———	———
$n!$	77 100 ans	———	———	———	———	———

Classe NP Co-NP (complémentaire de NP)

Le mot ***NP*** vient de l'abréviation de "***N***on deterministic ***P***olynomial time". Dans cette classe on retrouve tous les problèmes de décision dont on peut associer à chacun d'eux un ensemble de solutions potentielles (de cardinal au pire exponentiel) de façon à pouvoir vérifier en un temps polynomial si la solution trouvée satisfait la question posée. Les problèmes dans *NP* sont les problèmes qui peuvent être résolus en énumérant l'ensemble des solutions possibles et en les testant à l'aide d'un algorithme polynomial.

La classe NP renferme beaucoup de problèmes pour lesquels aucun algorithme polynomial déterministe n'est connu, de ce fait on peut dire que la classe NP est plus large que la classe P Dans les années 60, les chercheurs pensaient que P et NP étaient des classes différentes, mais sans preuve. En 1971, Cook (1971) démontre l'existence d'un problème NP qui, s'il est soluble par un algorithme polynomial, prouverait que tous les problèmes de la classe NP sont des problèmes de la classe P. Dans sa contribution Fortnow (2009) se pose la question « *what if* $P = NP$ *? » et si* $P = NP$? La réponse est en résumé « le monde est beau …. », mais depuis 30 ans de recherche active, cette réponse est négative et aucune démonstration de ce résultat négatif n'a été trouvée jusqu'à nos jours. Malgré la croyance que $P \neq NP$ la question reste toujours ouverte car les enjeux, concernant essentiellement la sécurité sur Internet, sont énormes. En effet, le système ***RSA***[1] qui permet le cryptage des messages électroniques dépend du fait que le décryptage du code ne soit pas un problème de la classe P . Si on démontrait que s'en était un, la sécurité de ce système serait compromise.

La classe duale de la classe ***NP***, quand la réponse est non, est appelée ***Co-NP.***

[1] ***RSA*** est un système de criptographie à clé publique inventé en 1977 par ***R***on Rivest, Adi ***S***hamir et Len ***A***dleman dont il porte les initiales, ce système est très utilisé dans des applications sur le web, qu'on estime à plus de 300 milions applications.

Classes EXPTIME

C'est la classe de problèmes ne pouvant être résolus que par des algorithmes de complexité exponentielle. Autrement dit cette classe renferme les problèmes décidables par un algorithme déterministe en temps exponentiel par rapport à la taille de son instance.

NC (Nick's Class)

Dans la classe ***NC*** les problèmes peuvent être résolus en temps **poly-logarithmique** (le temps nécessaire à la lecture des données en entrée est relativement plus grand que le temps nécessaire à leur résolution) sur une **machine parallèle** ayant un nombre **polynomial** (c'est-à-dire raisonnable) de processeurs.

Un problème est dans ***NC*** s'il existe un algorithme pour le résoudre pourrait être parallélisé, qui serait plus efficace que la version séquentielle. Le gain est significatif dans le cas où la version parallèle de l'algorithme (s'exécutant sur plusieurs processeurs) est plus efficace que la version séquentielle.

4.3.3. La réduction de Turing

Une réduction de Turing d'un problème Π_1 à un problème Π_2 est un algorithme A_1 qui résout Π_1 en faisant appel (éventuellement plusieurs fois) à un algorithme A_2 qui résout Π_2 tel que si A_2 est polynomial, alors A_1 l'est aussi. On notera $\xrightarrow{T}$ une telle réduction (donc ici $\Pi_1 \xrightarrow{T} \Pi_2$). La flèche $\xrightarrow{T}$ indique donc un sens de difficulté des problèmes, du plus facile au plus au plus difficile. Les réductions de Turing sont donc transitives.

Inclusions des classes

De la définition des classes de la complexité ressort les relations d'inclusions suivantes :

- $P \subseteq NP$, et symétriquement $P \subseteq Co - NP$
- $NP \subseteq PSPACE = NPSPACE$, et $Co\_NP \subseteq PSPACE$

En effet, un problème polynomial peut se résoudre en engendrant une solution et en la testant à l'aide d'un algorithme polynomial. Tout problème dans *P* est ainsi dans *NP*.

4.3.4. La NP-complétude

La théorie de la *NP-complétude* concerne la reconnaissance des problèmes les plus durs de la classe *NP*. La notion de la difficulté d'un problème qui est introduite dans cette classe est celle d'une classe de problèmes qui sont équivalents dans le sens où si l'un d'eux est prouvé être facile alors tous les problèmes de *NP* le sont. Inversement, si l'un d'eux est prouvé être difficile alors la classe *NP* est différente de la classe *P*.

Les problèmes NP-difficiles

Un problème *P* est dit *NP-difficile* si pour tout problème $P' \in NP$, P' est réductible à P , notons que cette définition n'impose pas l'appartenance des problèmes $NP - difficiles$ à P . La Figure 4-1 montre la représentation des classes *NP et P*.

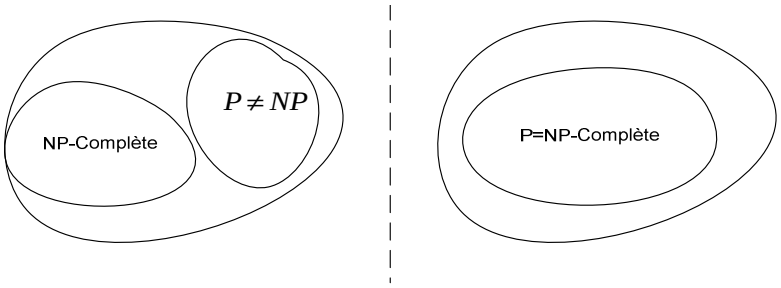


Figure 4-1 : Représentation de la classe *NP* et *P*

4.3.5. Récapitulatif des notations de la complexité algorithmique

Ainsi sur une machine déterministe, si un problème requière un temps *t(n)*, on parle alors d'un ordre de grandeur, ainsi pour un problème polynomiale on considère uniquement le degré de polynôme, si le degré est 2, la complexité est alors quadratique $O(n^2)$ et le problème appartient à la classe $TIME(t(n^2))$. Le Tableau 4-2 récapitule les différentes classes de complexité.

Tableau 4-2 : Signification des notations de la complexité

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

4.4. Quelques problèmes d'optimisation combinatoire

On vient de faire un petit tour d'horizon de la théorie de la complexité, ce qui nous aidera par la suite à comprendre le degré de difficulté des problèmes qu'on essaie de résoudre. Dans ce qui suit, on tentera d'élaborer un inventaire des problèmes les plus fréquents de l'optimisation

combinatoire en commençant par le plus connu d'entre eux qui est le problème du voyageur du commerce, une liste de centaines de problèmes est disponibles dans Garey et Johnson (1979). D'autres problèmes se sont ajoutés par la suite qui sont parfois des variantes de ce qui existent déjà.

4.4.1. Le problème du voyage de commerce

Parmi les problèmes les plus connus de la recherche opérationnelle, on trouve le problème de voyageur connu sous l'abréviation anglo-saxonne **TSP** pour **T**raveling **S**alesman **P**roblem. La popularité du TSP est due notamment à la simplicité de son énoncé et à sa difficulté de résolution. C'est William Rowan Hamilton qui a posé pour la première fois ce problème sous forme de jeu dès 1859. Sous sa forme la plus classique, son énoncé est le suivant : « Un voyageur de commerce doit visiter une et une seule fois un nombre fini de villes et revenir à son point d'origine. Trouvez l'ordre de visite des villes qui minimise la distance totale parcourue par le « voyageur », il s'énonce comme suit :

*« Etant donné **n villes** et les distances séparant ces villes, le but est de trouver un chemin de longueur totale minimale qui passe exactement une seule fois par chaque ville et revienne à la ville de départ. Les distances peuvent être des coûts dans ce cas on s'intéressera à minimiser le coût de la tournée ».*

Cet énoncé a une apparence simple, cependant le TSP est un problème très difficile à résoudre. La solution consiste à trouver un cycle hamiltonien passant par tous les points représentant les villes. Soit $c_{i,j}$ la distance séparant les deux points $(i \to j)$ représentant les deux villes (i, j), π une permutation de $\{1,2,...,n\}$ est et soit l la fonction qui donne la longueur de la tournée, on a alors :

$$l(\pi) = \sum_{i=1}^{n-1} c_{\pi(i),\pi(j)} + c_{\pi(n),\pi(1)}$$

Les auteurs dans Dantzig (1954) ont proposé la première formulation linéaire en nombres entiers TSP en utilisant une variable binaire $x_{i,j}$ dont la définition est la suivante :

$$x_{i,j} = \begin{cases} 1 \text{ si } (i,j) \text{ appartient à la solution optimale} \\ 0 \text{ sinon} \end{cases}$$

Formulation linéaire du TSP

$$Minimiser \sum_{i \neq j} c_{i,j} x_{i,j} \qquad (1)$$

Sous les contraintes suivantes :

$$\sum_{j=1}^{n} x_{i,j} = 1, \qquad i = 1,...,n \qquad (2)$$

$$\sum_{i=1}^{n} x_{i,j} = 1, \qquad j = 1,...,n \qquad (3)$$

$$\sum_{i,j \in S} x_{i,j} \leq |S| - 1, \qquad S \subset X, 2 \leq |S| \leq n-2 \qquad (4)$$

$$x_{i,j} \in \{0,1\}, \qquad i, j = 1,...,n, i \neq j \qquad (5)$$

La contrainte numéro *(1)* définit la fonction objectif qui nous donne le coût de la tournée ; les contraintes *(2)* et *(3)* indiquent respectivement que pour chaque nœud, ils existent un seul arc sortant et un seul arc entrant : elles sont appelées des contraintes de degré. Les contraintes *(4)* interdisent la formation des sous-cycles et les contraintes *(5)* indiquent que les variables $x_{i,j}$ sont binaires.

Les contraintes de type *(4)* génèrent un ensemble de contraintes non polynomiales ce qui rend l'utilisation de cette formulation en pratique inopportune.

La preuve que le ***TSP*** appartient à la classe des problèmes NP-difficile est donnée par Johnson et Papadimitriou (1985).

On trouve de nombreuses applications du ***TSP*** dans les problèmes de logistique, de transport des biens et de personnes et dans l'industrie pour l'optimisation de trajectoires de machines outils, dans la fabrication des cartes mère lors de perçage des points sur ces dernières, mais également la datation des sites archéologiques.

4.4.2. Le problème de sac à dos

Le problème du sac à dos ou ***KP*** (de l'anglais ***K****napsack* ***P****roblem*) modélise une situation analogue au remplissage d'un **sac à dos** Il s'énonce comme suit :

« Il s'agit de remplir un sac de capacité c *avec un sous-ensemble de* n *produits ; chaque produit* $i \in \{1,2,..,n\}$ *est cratérisé par un poids* w_i *(weight) et un profit* p_i*. Le but est de choisir un sous-ensemble de produits qui maximisent le profit, tout en n'excédant pas la capacité* c*. »*

Le *KP* appartient à la classe *NP*, qui est la classe des problèmes dont les solutions sont vérifiables en temps polynomial. En effet si l'on cherche à répondre à un problème de décision qui est de savoir s'il existe un sous-ensemble $s \subset S = \{1,2,....,n\}$ d'objets dont la somme (profits) est au moins égale à $K \in \mathbf{N}$, il suffit de vérifier si $\sum_{i=s} w_i \leq c$ et si $\sum_{i=s} p_i \leq k$ ce qui se fait en temps polynomial.

Dans Martelo et al., (2000) on trouve un état de l'art concernant la résolution de ***KP 0-1*** par les méthodes exactes, notons que le *KP 0-1* est la variante la plus classique de *KP* dans laquelle un produit est choisit {1} ou bien refusé {0}.

Il est formulé par un programme linéaire en nombres entiers comme suit :

$$Maximizer \sum_{i=1}^{n} p_i x_i \qquad (1)$$

Sous les contraintes

$$\sum_{j=1}^{n} w_i x_i \leq c \qquad (2)$$

Une particularité du problème de sac à dos est qu'il est lui-même un sous problème des problèmes plus généraux (c'est-à-dire qu'il constitue une réduction de plusieurs problèmes) on lui trouve des applications dans plusieurs domaines tels que :

- La version *NP-difficiles* de ce problème a été utilisée dans des primitives et des protocoles de cryptographie ;
- Dans le chargement de bateau ou d'avion : tous les bagages à destination doivent être amenés, sans être en surcharge ;
- La minimisation des chutes lors de la découpe de sections de longueurs diverses dans des barres en fer.

4.4.3. Le problème d'ateliers

Dans les problèmes d'atelier, on s'intéresse à la production d'un certain nombre de pièces appelées jobs (en Anglais) en utilisant les machines dont dispose l'atelier et ceci en consommant ou non des ressources. Dans la forme la plus simpliste on s'intéresse à la minimisation du temps de production, et on le donne en fonction de la date de fin de la fabrication de la dernière pièce : ce temps est communément appelé Makespan. Pour la minimisation du temps il s'agit de trouver le meilleur ordre de production qui donne le meilleur temps minimal de production. Chaque pièce est décrite par sa gamme de production.

Comme nous l'avons mentionné dans le paragraphe 3.2.6 la nature des gammes opératoires définissent plusieurs problèmes d'ateliers.

4.4.4. Le problème de RCPSP

De l'anglais ***R***esource-***C***onstrained ***P***roject ***S***cheduling ***P***roblem, est un problème d'ordonnancement à contraintes de ressources, c'est-à-dire qu'on cherche à trouver un ordonnancement, de durée minimale, des activités d'un projet entrant en compétition sur l'usage de ressources renouvelables, cumulatives et disponibles en quantité limité. C'est l'un des problèmes les plus étudié en ordonnancement car il couvre un grand nombre des problèmes théoriques d'ordonnancement et on lui trouve un très grand nombre d'applications dans l'industrie. Les instances de tailles réelles qu'on trouve dans le domaine de gestion de production ne sont pas encore résolues d'une manière optimale. Maniezzo et Mingozzi (1998) ont proposé une formulation linéaire et par la suite plusieurs chercheurs ont proposé des bornes inférieures en se basant sur la formulation soit par hybridation de PPC (Programmation Par Contraintes) et de génération de colonnes de Mingozzi dans les travaux Brucker et Knust (2000) soit par relaxation lagrangienne dans les travaux de Demassey et al., (2002a) et Demassey et al., (2002b).

4.5. Conclusion

Dans ce paragraphe, nous avons défini les problèmes d'optimisation (*COPs*), la complexité algorithmique des différentes classes de problèmes d'optimisation. Nous avons donné également quelques exemples de problème de ce type : le voyageur de commerce (*TSP*), le problème de sac à dos (*KP*), le problème d'ateliers et le RCPSP. Ces problèmes se distinguent des autres problèmes d'optimisation par le fait que les variables de décision qui composent une solution sont discrètes.

5. Conclusion

Ce chapitre a présenté trois thèmes principaux : les problèmes posés par les systèmes flexibles de production, les problèmes d'ordonnancement et leur modélisation, et les différents problèmes d'optimisation.

Cette présentation nous a permis de définir la classe des SFP que nous allons étudier. Nous nous intéressons plus particulièrement aux SFP qui utilisent des systèmes automatisés de transport/manutention basés sur l'utilisation (cf. Figure 2-1, Figure 2-2, Figure 2-3) :

- des chariots électrifiés à rail ;
- des chariots filoguidés ;
- des robots spécifiques pour le déplacement des pièces.

Lors de l'étude du domaine nous avons mis en évidence les problèmes de conception et d'exploitation des SFP. Nous nous intéressons plus particulièrement à la planification à court terme ou aux problèmes d'ordonnancement dans les SFP. De manière plus formelle, il s'agit, pour résoudre le problème, de construire une solution sans interblocages du système avec :

- l'ordre et les dates d'usinage des pièces sur les machines ;
- l'ordre et les dates de passage des pièces sur les stocks d'entrée / sortie de chaque machine ;
- l'ordre et les dates des transports en charge des moyens de transport / manutention ;
- l'ordre et les dates des transports à vide des moyens de transport / manutention.

Nous nous sommes intéressés aux problèmes d'ordonnancement, à leur description, à leur classification et à leur notation. Nous avons décrits les principaux modèles théoriques et nous avons donné des exemples.

La démarche de modélisation des problèmes d'ordonnacement préconise la construction conséquitive de deux modèles : modèle de connaissance et modèle d'action. En particulier, la démarche préconise de partir des modèles théoriques existants pour les enrichir progressivement. Ainsi, les méthodes efficaces de la littérature peuvent être réutilisées et étendues.

L'application de méthodes de recherhe operationnelle permet d'appréhender les problèmes complexes rencontrés dans les SFP reels, le but étant d'apporter une aide à la décision.

Dans ce contexte, dans ce chapitre, nous avons défini les problèmes d'optimisation combinatoire (POC ou COPs) et nous en avons donné quelques exemples.

D'autre part, le problème d'ordonnacement des Systèmes Flexibles de Production se ramène quand à lui à un job-shop avec transport et contraintes supplémentaires. Dans le chapitre suivant nous présentons le problème du job-shop.

Chapitre II : Le problème de job-shop

Ce chapitre présente le problème de job-shop et les principales techniques utilisées pour sa résolution. Entre autres, les méthodes exactes et approchées, le graphe conjonctif - disjonctif et les voisinages afférents sont présentés. Tous ces outils sont réutilisés et adaptés dans les chapitres suivants.

Liste des tableaux

Liste des figures

Liste des algorithmes

1. Introduction

Comme on vient de voir dans le chapitre précédent, on définit un problème d'ordonnancement comme un problème consistant à choisir un ordre des opérations s'exécutant sur un ensemble de ressources. Ce choix doit comporter l'affectation des ressources aux opérations, et doit permettre la minimisation ou la maximisation de plusieurs critères. Cette situation se rencontre dans tous les systèmes flexibles de production, c'est pour cette raison, que l'ordonnancement en tant qu'activité est très étudié dans le cadre des SFP. Lorsque l'ensemble des tâches à réaliser est connu, on dit que le problème est statique contrairement à un problème dynamique où les tâches ne seront donc connues qu'à mesure de l'avancement de la production.

Les SFP sont conçus pour être flexibles, ce qui offre une possibilité de production de plusieurs produits simultanément. D'autre part, dans les problèmes d'ordonnancement les opérations sont regroupées en jobs, chaque job possède sa gamme de production pour lequel on introduit d'autres contraintes. Vu sous cet angle, on peut facilement assimiler chaque produit à un job. Partant de cette analogie, le modèle qui représente le mieux les SFP est alors, le Job-Shop (*JS*). Ce modèle a été retenu dans sa forme classique (sans extensions) pendant près de trente ans (début des années 60 jusqu'au au début des années 90). Cependant, la communauté scientifique et les industriels ont trouvé, que le *JS* sous sa forme de base ne représente pas les contraintes réelles d'un SFP, comme mentionnés par Buzacott et al., (1988), Schutten., (1998) et Rogers et White (1990). Ceci amena les experts du domaine à inclure d'autres aspects dans le processus de modélisation telles que : le transport des jobs, les fenêtres de temps (time windows), les temps de reconfiguration et de préparation (setup times) etc. Bien qu'on trouve de plus en plus de travaux sur les extensions du job-shop, le Job-Shop sous sa forme classique continu à intéresser les scientifiques car il demeure un problème ouvert (on n'a pas encore trouvé une méthode efficace pour sa résolution).

Nous allons nous intéresser aux différentes méthodes d'optimisation utilisées pour la résolution du problème du *JS*. Compte tenu du temps d'exécution des méthodes exactes, nous nous intéressons aux méthodes approchées car la modification d'une solution apparaît naturellement dans la résolution des problèmes d'ordonnancement, de planification et d'allocation de ressources. Les principaux avantages des méthodes approchées sont leur universalité, puisqu'elles permettent d'optimiser un critère (ou même plusieurs simultanément) à peu près quelconque. En effet il suffit de pouvoir calculer la valeur du critère en tout point de l'espace dans lequel on recherche une solution. Leur bon fonctionnement repose sur la définition d'une notion de proximité telle que deux états « proches » possèdent des coûts « proches ». La définition de la notion de proximité influe sur leur fonctionnement, c'est à dire sur la qualité des solutions trouvées. Les principales critiques liées à ces méthodes concernent leur vitesse de convergence et les difficultés de réglage.

Ce chapitre est composé de cinq parties. La première présente le problème de job-shop en général (2), la difficulté de résolution et sa modélisation sous la forme d'un graphe disjonctif-conjonctif. La deuxième (paragraphe 3) présente les principales méthodes d'optimisation classifiées dans deux catégories : les méthodes exactes et les méthodes approchées. Une troisième partie (paragraphe 4) présente les principaux voisinages utilisés pour le job-shop. La dernière partie présente les extensions possibles.

2. Le job-shop : présentation

2.1. Introduction

Dans cette partie, nous présentons le problème de job-shop. Nous nous intéressons à ce problème parce qu'il est classique dans la littérature et car il s'agit d'un problème très « général » parmi les problèmes classiques d'ordonnancement d'atelier Jain et Meerran (1999).

Le problème de job-shop généralise les problèmes classiques d'ordonnancement tels que le problème à une machine, le flow-shop de permutation et le flow-shop général.

Le problème d'ordonnancement des SFP se décompose en deux sous problèmes : un problème d'affectation et un problème d'ordonnancement. Le problème d'ordonnancement est un problème disjonctif dont la forme la plus générale est le problème de job-shop.

2.2. Historique du Job-Shop

L'histoire du Job-Shop a commencé il y a plus de quarante ans. Cependant c'est dans les années 1960 qu'est apparu un problème d'ordonnancement aujourd'hui bien connu, celui de Fisher et Thompson avec 10 jobs et 10 machines Fisher et Thompson (1963). Ce problème résista pendant près de 25 ans et engendra une compétition entre les chercheurs du domaine Blazewicz et al., (1996). Dans cet article les auteurs dressent un état de l'art du JS et donnent une définition formelle du problème. Ils décrivent également les méthodes de résolution à l'aide d'équations linéaires ainsi qu'une modélisation sous forme d'un graphe disjonctif-conjonctif. Les méthodes de résolution sont séparées en deux familles, à savoir :

- les méthodes exactes pour lesquelles ils donnent tous les éléments nécessaires à la construction et l'exécution d'un algorithme exact et les méthodes de branchement et les inégalités valides ;
- les méthodes approchées pour les quelles ils décrivent quelques règles de priorité, la procédure à machine goulot, les algorithmes de recherche locale ainsi que des procédures d'ordonnancement opportunistes. Ils énnoncent également les principaux voisinages utilisés dans la littérature.

Dans Jain et Meerran (1999) on trouve, un bon état de l'art concernant le job-shop de son début jusqu'à la fin des années 90. Il liste les articles en les classant par méthode de résolution utilisée pour le JS. En plus de la description de la méthode, Jain et Meerran proposent une liste complète d'instances. Cette dernière sert de base de comparaison entre les méthodes de résolution. Ils présentent également des tableaux de synthèse dans lesquelles ils donnent une vue d'ensemble récapitulative des méthodes et des extensions ainsi que les références les évoquant.

Les méthodes exactes commencent avec les premiers travaux menés par Manne (1960), puis par Brooks et White (1965) suivi en 1968 par Greenberg qui utilisa une formalisation linéaire en nombre entiers Greenberg (1968). S'en suivi alors de nombreuses publications notamment celles de Ficher dans Fisher (1973a) et Fisher (1973b) qui utilisa les multiplicateurs de Lagrange. Dès les années 1975 McMahon et Florian développent un algorithme McMahon et Florian (1975) dépassant la performance des algorithmes de Fisher.

Durant les vingt dernières années furent créés des algorithmes plus performants utilisant entre autre la relaxation lagrangienne qui consiste en l'abandon de certaines contraintes pour résoudre des problèmes plus proches de la réalité avec plus d'une centaine de jobs et d'une cinquantaine de machines. Cette technique de relaxation fut travaillée afin d'associer à chaque

contrainte relâchée une pénalité appelée technique de relaxation lagrangienne augmenté. Plus tard, en 1989, Carlier et Pinson (1989) décrivent un algorithme basé sur le Branch and Bound, qui pour la première fois, résout de manière optimale le problème 10x10 de Fisher et Thompson ainsi que d'autres plus complexes. Plus récemment nous pouvons retenir les approches proposées dans Caseau et Laburthe (1995), Perregaard et Clausen (1995) et Blazewicz et al., (1998). Cependant bien que ces méthodes de résolutions des problèmes de type Job-Shop fournissent des résultats exacts, elles nécessitent souvent un temps d'exécution très long pour des problèmes de type FT 10x10. C'est pour cela qu'en parallèle de ces approches, d'autres, pouvant être qualifiées de méthodes approchées, ont été développées.

Les méthodes approchées offrent une alternative intéressante, car elles permettent d'obtenir des résultats rapidement. Ces dernières peuvent être envisagées pour le traitement de problèmes réels (généralement de grande taille). Comme il est très difficile d'établir un état de l'art exhaustif d'apparition des méthodes approchées pour le job-shop, on a choisi l'ordre chronologique. Les premières méthodes sont donc des heuristiques de construction apparues à la fin des années 50 et début des années 60 comme celles proposées par Giffler et Thompson (1960) et Fisher et Thompson (1963). Dans les années 70, malgré cet intérêt réel pour les méthodes exactes, on note la naissance des méthodes de relaxations, suivi dans les années 80, de l'apparition des métaheuristiques telles que les algorithmes génétiques, la recherche Tabou et le recuit simulé. Ensuite, on a vu l'émergence de méthodes telles que la programmation par contraintes, les réseaux de neurones, les systèmes expert ainsi que le recours à la simulation, afin de résoudre les problèmes d'ordonnancement.

2.3. Définition formelle

Le problème du job-shop consiste en l'exécution d'un ensemble de n jobs $J=\{J_1,\ldots,J_n\}$ sur un ensemble de m machines $M=\{M_1,\ldots,M_m\}$. Chaque job J_i est composé d'un ensemble ordonné de n_i opérations notées $O_{i,1}, O_{i,2}, \ldots, O_{i,n_i}$ pour lesquelles des contraintes de précédence sont décrites. A chaque opération machine $O_{i,j}$ est associé un temps de traitement p_{ij} correspondant à sa durée d'exécution. Chaque machine ne peut traiter qu'un job à la fois et un job ne peut s'exécuter que sur une seule machine en même temps. Les jobs exécutés sur une machine M_i doivent attendre dans le stock de sortie. De la même manière chaque machine M_i dispose d'un stock d'entrée où les jobs attendent leur exécution sur cette machine. Le temps de transfert des jobs n'est pas pris en compte.

On suppose que les valeurs p_{ij} sont des entiers non-négatifs. L'objectif est de trouver un ordonnancement faisable qui minimise le makespan $C_{\max}=\underset{j=1,n}{Max}\{C_j\}$ où C_j dénote la fin d'exécution de la dernière opération O_{j,n_j} du job *j*. Le résultat de l'ordonnancement est constitué des dates de début de toutes les opérations machine.

Les hypothèses suivantes sont communément retenues pour le problème de job-shop. Certaines variantes du problème de job-shop consistent à supprimer certaines de ces hypothèses :

- Tous les jobs sont disponibles dès le début de l'ordonnancement.
- Les machines sont disponibles sur tout l'horizon d'ordonnancement (pas de panne, pas d'état initial non vide).
- Les temps de traitement p_i sont déterministes et connus à l'avance.
- Les temps de montage et de démontage sont inclus dans les temps de traitement
- Les temps de transport sont négligeables.

- Chaque machine ne peut réaliser à un instant donné qu'une seule opération.
- Un job peut être traité au plus par une machine à un instant donné.
- Les produits peuvent attendre dans les zones de stockage de capacité illimitée.

2.4. La preuve de l'appartenance à la classe NP-difficile

La connaissance de la classe à laquelle appartient un problème est très important afin de bien l'appréhender. Pour ce qui concerne le job-shop composé de n jobs à exécuter sur m machine il existe $(n!)^m$ solutions possible. Le job-shop est un problème *NP-Difficile* et *NP-complet* et plusieurs démonstrations existent. Pour ce qui concerne le Job-Shop on a les résultats suivants :

- les variantes de Job-Shop ***J2//Cmax*** et ***J3/pij=1/Cmax*** sont prouvés ***NP-Difficiles***, dans Lenstra et al., (1979) ;
- les variantes suivantes sont polynomialement solvables :
  - ***n*** jobs (avec au plus ***2*** opérations) et ***2*** machines Jackson (1956)
  - ***J2/pij=1, ri/ Cmax*** Timkovsky et Rubinov (1956) ;
  - le job-shop avec ***2*** jobs et ***m*** machines Brucker (1988

Par principe, quand on démontre qu'un problème ***A*** est ***NP-Difficile,*** et que ce problème est une réduction d'un problème plus complexe ***B***, alors ***B*** est forcement ***NP-Difficile***. Le Job-shop avec *n* jobs et *m* machines est donc ***NP-Difficile***, il suffit de le réduire à la variante J2||Cmax qui est déjà ***NP-Difficile***.

On peut aussi réduire le job-shop au ***TSP*** avec les considérations suivantes : chaque ville est un job; le voyageur de commerce est la machine ; chaque job va passer une et une seule fois sur la machine. Comme le TSP est *NP complet* et *difficile* alors le Job-Shop l'est aussi.

Le tableau 2-1 extrait de Drozdowski (1996) donne une vue d'ensemble sur la complexité du Job-Shop avec :

- n pour le nombre de jobs ;
- m pour le nombre de machines ;
- n_j pour le nombre d'opérations par job ;
- $d(o)$ pour la durée d'opération ;

Tableau 2-1 : Complexité du Job-shop Drozdowski (1996)

<table>
<tr><th>P</th><th colspan="2">NP</th></tr>
<tr><td>$m=2,\forall J \quad n_j \le 2$</td><td>$m=2,\forall J \quad n_j \le 3\left[\exists k, n_k = 3\right]$</td><td rowspan="2">$m=2$
autre cas</td></tr>
<tr><td>$m=2,\forall o \quad d(o)=1$</td><td>$m=2,\forall o \quad d(o) \le 2\left[\exists k, d(k)=3\right]$</td></tr>
<tr><td rowspan="2"></td><td>$m=3,\forall J \quad n_j \le 2$</td><td rowspan="2">$m \ge 2$</td></tr>
<tr><td>$m=3,\forall o \quad d(o)=1$</td></tr>
<tr><td>$n=2$</td><td colspan="2">$n \ge 3$</td></tr>
<tr><td>$C_{\max} \le 3$</td><td colspan="2">$C_{\max} \ge 3$</td></tr>
</table>

2.5. Modélisation sous forme d'un graphe disjonctif-conjonctif

Le graphe disjonctif a été introduit pour la première fois en 1964 par Roy et Sussmann (1964). La particularité de ce graphe est qu'il utilise des arêtes non orientées qui peuvent prendre les deux sens, l'évaluation du graphe n'est possible qu'une fois toutes les arêtes sont orientées : on parle alors d'un graphe conjonctif. Ce graphe a reçu pas mal d'extensions et d'adaptations pour représenter les différents problèmes d'ordonnancement même s'il a été principalement étudié pour les problèmes de Flow-Shop, de job-shop et de RCPSP.

D'une manière formelle le graphe disjonctif est un graphe dont les nœuds représentent des opérations, les arcs et des arêtes des contraintes temporelles de la forme $t_j \ge t_i + a_i$, entre les opérations (i, j) où t_i, t_j sont leurs dates respectives de début et a_i durée de l'opération i. Pour modéliser une contrainte de précédence entre l'opération i et j on introduit une arrête de i vers j d'un coût égal à la durée de l'opération i.

Pour orienter ce graphe il suffit donc de fixer l'ordre des opérations au préalable et ainsi obtenir un graphe conjonctif orienté qu'il suffit d'évaluer avec un algorithme de plus long chemin de type Bellman. Le résultat obtenu est alors les dates de début au plus tôt des opérations. Ainsi le modèle du graphe est divisé en deux modèles distincts : le graphe disjonctif qui modélise le problème en général et qui contient les contraintes indépendantes de l'ordre des opérations et le graphe conjonctif qui modélise toutes les contraintes pour un ordre donné des opérations.

Dans la suite, on détaille la construction des deux graphes : le graphe disjonctif (Figure 2-1) qui modélise le problème et dans lequel à priori aucune disjonction n'est arbitrée et le graphe conjonctif (Figure 2-2) qui modélise une solution.

On note $G=(V,A,E)$ un graphe disjonctif, où V est un ensemble de nœuds, A est un ensemble d'arcs et E est un ensemble d'arêtes. Les ensembles V, A et E sont construits de la manière suivante :

- Pour chaque opération du problème de job-shop, on crée un nœud dans V. Pour faciliter les explications, on pourra donc confondre les nœuds et les opérations. On

crée de plus deux nœuds 0 et $*$ qui modélisent deux opérations fictives. Le nœud 0 représente une opération fictive réalisée avant toutes les autres opérations, on dit que ce nœud est la source du graphe. Le nœud $*$ représente une opération fictive réalisée après toutes les autres opérations, on dit que ce nœud est le puits du graphe.

- On crée un arc dans A entre deux opérations consécutives dans la gamme du job. Ainsi, pour chaque opération de nœud i, on crée un arc du nœud i vers le nœud $i+1$ où $i+1$ est l'opération suivante dans la gamme du job. Cet arc est de longueur p_i (i.e. la durée de traitement de l'opération i).
- On crée une arête dans E entre deux opérations traitées par la même machine. Cette arête représente la disjonction entre ces deux opérations et devra être orientée pour construire un graphe conjonctif. Il y a donc un ensemble d'arêtes reliant toutes les opérations à réaliser sur la même machine.

Pour illustrer la construction du graphe disjonctif, le Tableau 2-2 propose une instance de job-shop et la Figure 2-1 présente le graphe disjonctif associé.

Tableau 2-2 : Instance exemple *J1*

Job	Opération 1	Opération 2	Opération 3
Job 1	(M 3, 1)	(M 1, 3)	(M 2, 7)
Job 2	(M 3, 5)	(M 4, 4)	(M 1, 1)
Job 3	(M 2, 5)	(M 1, 5)	(M 3, 3)

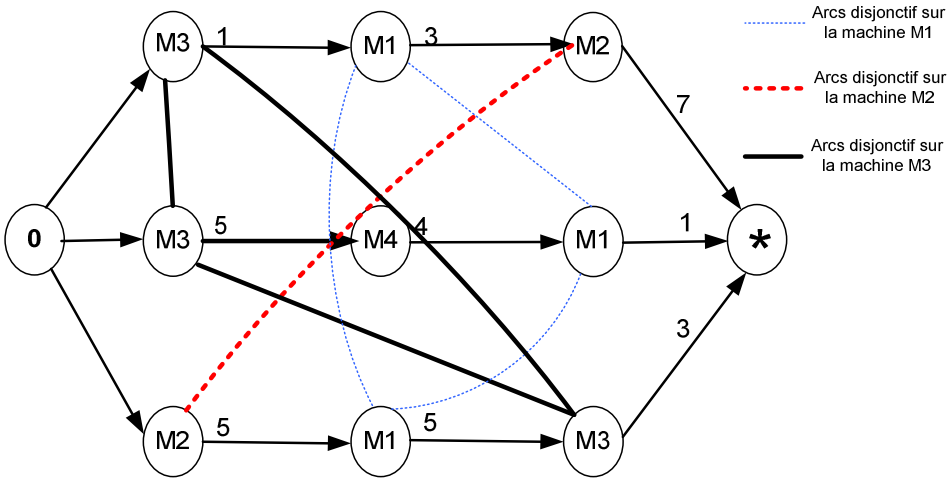


Figure 2-1 : Graphe disjonctif du problème

L'étape suivante est l'orientation des arêtes du graphe disjonctif. Orienter une arête entre deux opérations consiste à remplacer cette arête par un arc dans le sens de l'orientation. La longueur de cet arc est la durée de traitement de l'opération de départ de l'arc.

Le modèle de graphe conjonctif-disjonctif permet de calculer l'ordonnancement semi-actif. L'obtention de l'ordonnancement semi-actif passe donc par les étapes suivantes :

- analyse du problème de départ pour construire le graphe disjonctif ;
- un ordre d'opération est considéré. Cet ordre peut être fourni par un expert, peut être calculé ou obtenu par un algorithme d'optimisation.

- le graphe disjonctif et l'ordre des opérations permettent de construire le graphe conjonctif. Ensuite, un algorithme de plus longs chemins permet de calculer l'ordonnancement semi-actif associé.

L'exemple de la solution *S1* est évalué dans le graphe de la Figure 2-2. La date de début de l'opération * est la date de fin de la dernière opération, c'est-à-dire le makespan. Le chemin le plus long donne le makespan de cette solution qui est donc 20.

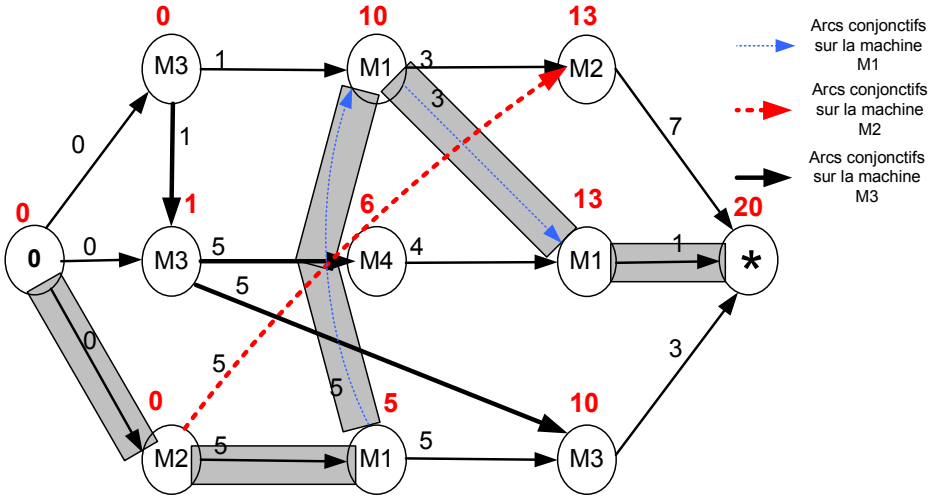


Figure 2-2 : Graphe conjonctif et valué de l'instance *J1*

Le chemin encadré en gris constitue le chemin critique et la succession des 3 opérations sur la machine M1 s'appelle le bloc critique « bloc machine ». Ce chemin est remarquable car tout retard d'une opération le long de ce chemin retarde la date de fin de l'ordonnancement. Le chemin critique est le chemin le plus long du graphe conjonctif et c'est lui qui définit le makespan de l'ordonnancement correspondant à ce graphe. Les blocs sont définis par rapport au chemin critique. Un bloc est une suite (de longueur maximale) de sommets faisant référence à la même machine. La notion de bloc est proposée par Grabowski et al., (1996) : elle a permis de définir des systèmes de voisinage très efficace pour le Job-Shop.

Sur le graphe de la Figure 2-2 le chemin critique est représenté en gris et se compose des opérations suivantes :

Job 2 / M1 Job 1 / M1 Job 3 / M1 Job 3 / M2

Le chemin critique se compose donc de deux blocs : un bloc qui fait référence à la machine M1 et qui comprend trois opérations et un bloc d'une seule opération qui concerne la machine M2.

2.6. Diagramme de Gantt

Le diagramme de Gantt tient son nom de son créateur Henry Laurence Gantt (1861-1919), ingénieur en mécanique et consultant en management. Il est surtout connu pour avoir mis au point en 1910 son célèbre diagramme très utilisé en gestion de projets. Il fut créé en 1917. Le but de ce diagramme est de représenter de manière claire et rapide la solution à un problème d'ordonnancement. Plusieurs variantes ont vu le jour depuis mais, au sens large, il se représente par un ensemble de lignes horizontales tracées dans un graphe avec le temps en abscisses. Chaque ligne est composée de différents rectangles qui sont pour chacun la représentation du temps

nécessaire à la réalisation d'un travail : le temps de réalisation est proportionnel à la longueur du rectangle. Dans le cadre d'un problème d'ordonnancement, le diagramme de Gantt prend en ordonnée les différentes ressources utilisées (cf. figure Figure 2-3) et donne une représentation sous forme de diagramme de Gantt d'un problème du job-shop.

Soit un problème de job-shop consistant à usiner sur 4 machines 3 pièces (3 jobs). Le Tableau 2-3 donne les gammes opératoires des jobs.

Tableau 2-3 : Gammes opératoires du l'instance *J2*

Job	Opération 1	Opération 2	Opération 3
Job 1	(M 3, 1)	(M 1, 3)	(M 2, 7)
Job 2	(M 3, 5)	(M 4, 4)	(M 1, 1)
Job 3	(M 2, 5)	(M 1, 5)	(M 3, 3)

Une solution de ce problème consiste à définir un ordre des opérations sur les machines. Le tableau 2-4 définit un ordre de passage des différents jobs sur les différentes machines.

Tableau 2-4 : Solution de problème du Job-Shop de l'instance *J2*

M1	Op. 2 du Job 3	Op. 2 du Job 1	Op 3 du Job 2
M2	Op. 1 du Job 3	Op. 3 du Job 1	
M3	Op. 3 du Job 3	Op. 1 du Job 1	Op. 1 du Job 2
M4			Op2 du Job 2

Si on suppose que chaque opération est calée le plus à gauche (elle définit alors une solution semi-active), on obtient le diagramme de la Figure 2-3.

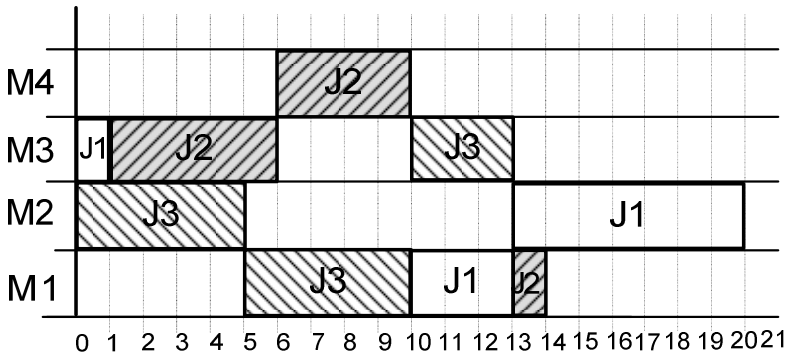


Figure 2-3 : Diagramme de GANTT de la solution de l'instance *J2*

2.7. Formalisation linéaire

Plusieurs formulations linéaires existent pour le job-shop et certaines d'entre elle se basent sur la formulation de Manne. Dans Pham (2008), une évaluation des formulations linéaires du job-shop est proposée.

Pour un problème de Job-Shop de n jobs et m machines, on considère les notations suivantes :

$\tilde{J}$ l'ensemble de tous les jobs; $\tilde{J} = \{1;2;...;n\}$;

I l'ensemble de toutes les opérations ;

M l'ensemble de machines $M = \{1;2;...;m\}$;

A_j l'ensemble de tous les couples d'opérations consécutives pour le job $j \in \tilde{J}$;

B l'ensemble de tous les couples d'opérations $(i, j) \in I, i \neq j$ exécutées sur la même machine ;

I_k l'ensemble des opérations exécutées sur la machine $k \in M$;

p_i le temps opératoire de l'opération $i \in I$;

H un nombre entier positif suffisament grand ;

x_i la date de début de l'opération $i \in I$;

x_τ la date de début de l'opération * ;

Pour chaque couple d'opérations $(i, j) \in I$ s'exécutant sur la même machine $k \in M$

$Y_{ij} = \begin{cases} 1 \text{ si l'opération } i \text{ est exécutée avant l'opération } j \\ 0 \text{ sinon} \end{cases}$.

La formulation de Manne consiste alors à Minimiser le makespan x_τ sous les contraintes suivantes :

$$x_j - x_i \geq p_i \qquad \forall (i, j) \in A_k, \forall k \in \tilde{J} \qquad (1)$$

$$x_j + H(1 - y_{ij}) - x_i \geq p_i \qquad \forall (i, j) \in B \qquad (2)$$

$$x_i + Hy_{ij} - x_j \geq p_j \qquad \forall (i, j) \in B \qquad (3)$$

$$x_\tau - x_i - p_i \geq 0 \qquad \forall i \in I \qquad (4)$$

$$y_{ij} \in \{0,1\} \qquad \forall (i, j) \in B \qquad (5)$$

$$x_i \geq 0 \qquad \forall i \in I \qquad (6)$$

$$x_\tau \geq 0 \qquad (7)$$

Les contraintes (1) assurent qu'aucune opération ne peut commencer avant la fin d'exécution de l'opération qui la précède. Les contraintes (2) et (3) sont des contraintes de disjonction machine et assurent que deux opérations s'exécutant sur la même machine doivent

être ordonnées grâce à l'utilisation de la variable binaire $y_{ij} \in \{0,1\}$ de la contrainte (5). Les contraintes (4) fixent la date de début de l'opération $*$, ceci est assuré par le fait que x_τ est supérieur à toutes les dates de début plus le processing time $x_i + p_i$ de chaque opération par $i \in I$. Les contraintes (6) et (7) imposent que toutes les dates de débuts des opérations sont positives ou nulles. Le Tableau 2-5 donne les caractéristiques de cette formulation.

Tableau 2-5 : Caractéristiques de la formulation de Manne Pham (2008)

Nombre de variables binaires	Nombre de variables continues	Nombre de contraintes
$mn(n-1)/2$	$mn+1$	$3mn^2/2+3mn/2-n+1$

La formalisation linéaire ci-dessus permet de résoudre tous les problèmes de job-shop. Mais, les temps de calcul obtenus peuvent rapidement devenir prohibitifs. La littérature du job-shop s'attache donc à développer des méthodes d'optimisation approchées dédiées au problème de job-shop de manière à proposer les solutions pour des problèmes de taille moyenne ou importante.

Pour ce qui concerne les méthodes approchées, plusieurs méta-heuristiques dédiées au problème ont été proposées. En plus du fonctionnement général des méta-heuristiques, la fonction de voisinage constitue un élément principal. Dans ce qui suit, nous présentons une description concise des principaux voisinages utilisés dans les problèmes d'atelier.

3. Les méthodes d'optimisation

On peut séparer les méthodes d'optimisation en deux familles distinctes : les méthodes exactes et les méthodes approchées. Sur la Figure 3-1, nous proposons une vue d'ensemble des méthodes d'optimisation et de leur schéma global d'optimisation. Le schéma fait apparaitre qu'après une modélisation du problème intervient une étape de modification, où l'on adapte la modélisation du problème à la méthode soit l'inverse ou soit les deux en même temps. On voit aussi qu'on peut utiliser plusieurs méta-heuristiques en même temps ou bien encore, qu'on peut combiner les méthodes approchées avec des méthodes exactes. Ce schéma d'optimisation suppose qu'après la satisfaction d'un critère d'arrêt, toutes les méthodes s'arrêtent en retournant la meilleure solution trouvée (solution optimale, borne supérieure ou borne inférieure). Dans ce qui suit on donne une description des méthodes les plus employées en recherche opérationnelle.

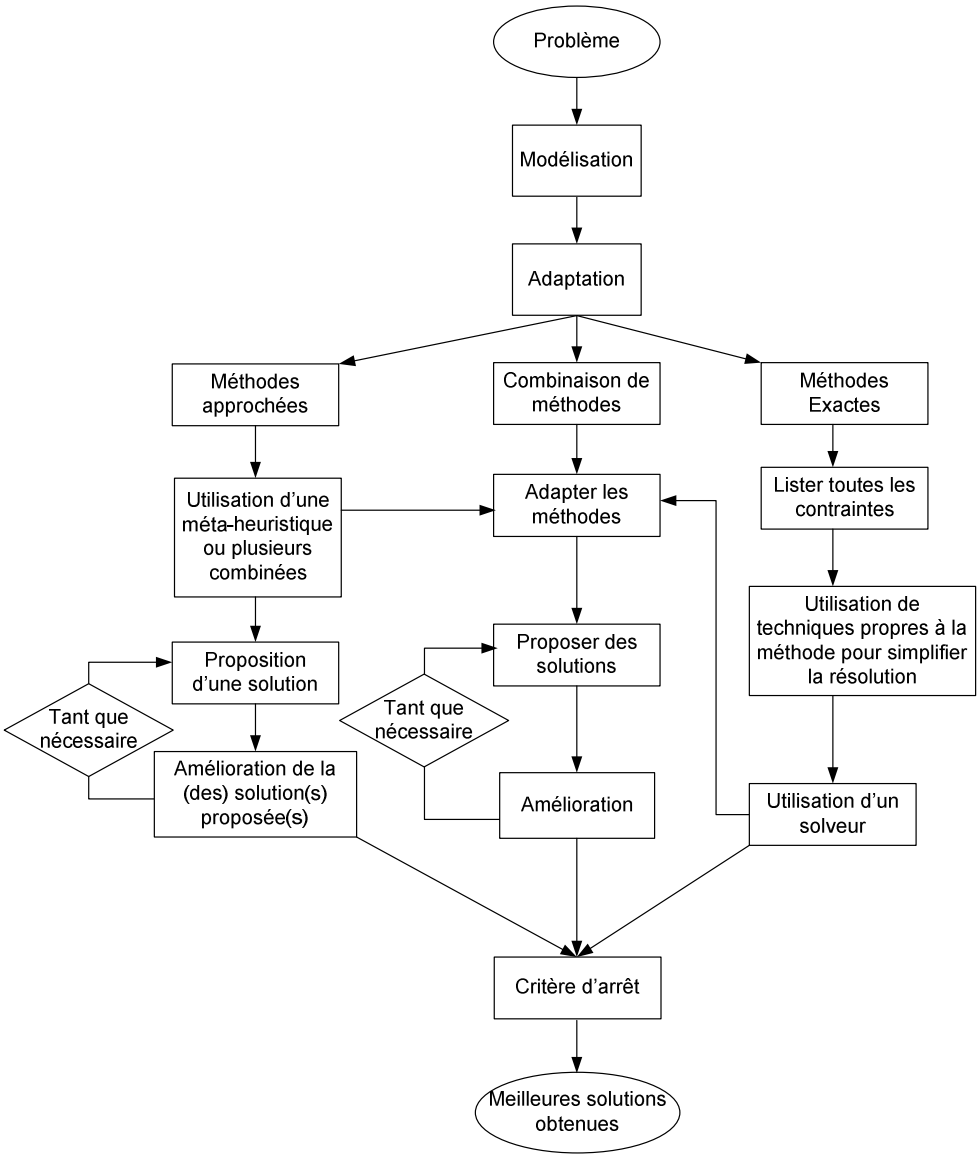


Figure 3-1 : Schémas d'optimisation

3.1. Les méthodes exactes

Les problèmes d'optimisation combinatoire ont un ensemble de solutions fini. Une manière intuitive de résoudre les problèmes d'optimisation discrets consiste à énumérer tout ou une partie des solutions. Néanmoins, pour un grand nombre de problèmes, cette énumération s'avère impossible à cause des temps de calcul prohibitifs entrainé par l'énumération. C'est pourquoi des techniques de relaxation de contraintes telle que la relaxation lagrangienne et d'autres comme la génération de colonnes ont été mises en œuvre pour les *COPs*.

3.1.1. La programmation linéaire

Il y a trois grandes familles de programmes linéaires : les programmes linéaires simples, les programmes linéaires en nombres entiers, et les programmes linéaires mixtes. Tous ces programmes sont des programmes dit linéaires, ils sont donc composés de contraintes sous forme d'équation (égale) ou d'inéquation (supérieure/égale et inférieure/égale) faisant intervenir des variables de manière linéaire.

Les programmes linéaires simples ont des variables dites "continues". Ces programmes linéaires peuvent être efficacement résolus par la méthode du simplexe ou la méthode du point intérieur. La méthode du point intérieur est polynomiale, la méthode du simplexe est exponentielle dans le pire des cas. En pratique, la méthode du simplexe est plus performante et plus répandue dans les outils de résolution.

Les programmes linéaires en nombres entiers font intervenir des variables entières. Ces programmes peuvent paraître plus simples que les précédents dans la mesure où le nombre de solutions possibles est fini. Pourtant, c'est ce type de problème qui est le plus difficile à résoudre. D'ailleurs certains programmes linéaires en nombres entiers sont reformulés (sous certaines réserves et sous certaines conditions) en programmes linéaires simples.

Les programmes linéaires mixtes comportent à la fois des variables continues et des variables en nombres entiers. D'un point de vue résolution, ces programmes linéaires ressemblent fortement au programme linéaire en nombre entiers et on ne retient en général que la taille du sous problème en nombre entiers pour mesurer la taille du problème mixte.

De nombreux solveurs génériques sont disponibles dans le commerce, et la résolution du programme linéaire peut être faite de manière relativement efficace. Pourtant, toutes les tentatives de résolution de problèmes disjonctifs (problèmes apparaissant en planification) à l'aide de ce type de solveurs ont été décevantes Jain et Meerran (1999).

3.1.2. Les procédures de séparation/évaluation (PSE)

La technique par séparation/évaluation (également rencontrée sous la dénomination Branch&Bound) réalise une énumération implicite de toutes les solutions : elle consiste à trouver une solution optimale sans parcourir toutes les solutions du problème. En d'autres termes de démontrer l'optimalité d'une solution en recherchant des bornes inférieures et supérieures de la solution optimale et en utilisant un schéma d'énumération arborescent.

Pour y parvenir, deux fonctionnalités sont introduites : la séparation et l'évaluation. La séparation consiste à partitionner un problème en deux ou plusieurs sous-problèmes. Les sous-problèmes engendrés sont des problèmes, plus faciles à résoudre, et dont la réunion des solutions de ces sous problèmes est une solution du problème maître. Les séparations successives réalisées durant la procédure créent un arbre appelé « arbre de recherche ». L'évaluation fournit pour chaque nœud de l'arbre une valeur appelée « borne inférieure ».

Ce processus est réitéré sur chacun des sous problèmes jusqu'à leur résolution (i.e. leur borne inférieure est égale à leur borne supérieure) ou leur rejet (i.e. leur borne inférieure est supérieure à la meilleure solution réalisable trouvée jusqu'ici). Un arbre est ainsi construit, dans lequel chaque nœud correspond à un problème et les fils de ce nœud représentent les sous problèmes générés.

Tous les nœuds sous le nœud évalué (et donc tous les sous-problèmes du problème considéré) sont des solutions de coût supérieur à la borne inférieure. C'est grâce à cette propriété que l'on peut éviter de parcourir certaines solutions de l'arbre (i.e. faire une énumération implicite plutôt qu'explicite). En effet, si une solution a déjà été trouvée, tout nœud dont la borne inférieure est supérieure à la solution trouvée n'a pas besoin d'être exploré, on dit que ce nœud et tout son sous-arbre ont été coupés.

Le succès d'une PSE réside dans la construction d'un arbre de taille raisonnable, alors que la difficulté principale consiste à obtenir une borne inférieure de bonne qualité.

Les méthodes exactes sont exigeantes en ressources, et elles requièrent des temps d'exécutions conséquents. Dans ce contexte on a souvent recours aux méthodes approchées. L'utilisation conjointe des méthodes approchées et des méthodes exactes existe. Souvent on utilise les méthodes exactes en limitant le temps pour obtenir des bornes inférieures qui donneront un point de comparaison pour les méthodes approchées. Plusieurs chercheurs essaient de combiner ces méthodes pour profiter de leurs avantages. Dans Dréo et al., (2003) on trouve les caractéristiques et les avantages des méta-heuristiques que l'on peut reprendre ici :

«

….*Ces méthodes ont en commun, en outre, les caractéristiques suivantes :*

- *elles sont, au moins pour partie, stochastiques : cette approche permet de faire face à l'explosion combinatoire des possibilités ;*
- *généralement d'origine discrète, elles ont l'avantage, décisif dans le cas continu, d'être discrètes, c'est-à-dire qu'elles ne recourent pas au calcul, souvent problématique, des gradients de la fonction objectif ;*
- *elles sont inspirées par des analogies : avec la physique (recuit-simulé, diffusion simulé…), avec la biologie (algorithmes évolutionnaires, recherche avec tabous…) ou avec l'éthologie (colonies de fourmis, essaims particulaires…).*
- *elles partagent les mêmes inconvénients : les difficultés des réglages des paramètres de la méthode et le temps de calcul élevé.*

Ces méthodes ne s'excluent pas mutuellement : en effet, dans l'état actuel de la recherche, il est plus souvent impossible de prévoir avec certitude l'efficacité d'une méthode donnée, lorsqu'elle est appliquée à un problème donné. De plus la tendance actuelle est l'émergence des méthodes hybrides, qui s'efforcent de tirer parti des avantages spécifiques d'approches différentes en les combinant.

».

Dans ce qui suit nous présentons les principales méthodes approchées utilisées en recherche opérationnelle.

3.2. Les méthodes approchées

La connaissance de l'espace de recherche des solutions d'un problème permet de guider le choix d'une méthode. Souvent ces espaces sont énormes et multidimensionnels et le processus de recherche est fastidieux, piégeant ainsi de nombreux algorithmes dans des solutions qui ne sont pas optimales. Ces solutions sont appelées *minimums locaux*. Le but d'une méthode d'optimisation est donc d'éviter de rester bloquer dans ces endroits (minimums locaux) et d'atteindre des endroits plus prometteurs afin de trouver des solutions optimales, car souvent ils en existent plusieurs. Ces solutions représentent se qu'on appelle communément des *minimums globaux*. La Figure 3-2 donne un aperçu de l'espace de recherche et on remarque sur le graphique l'existence de plusieurs minimums locaux et globaux.

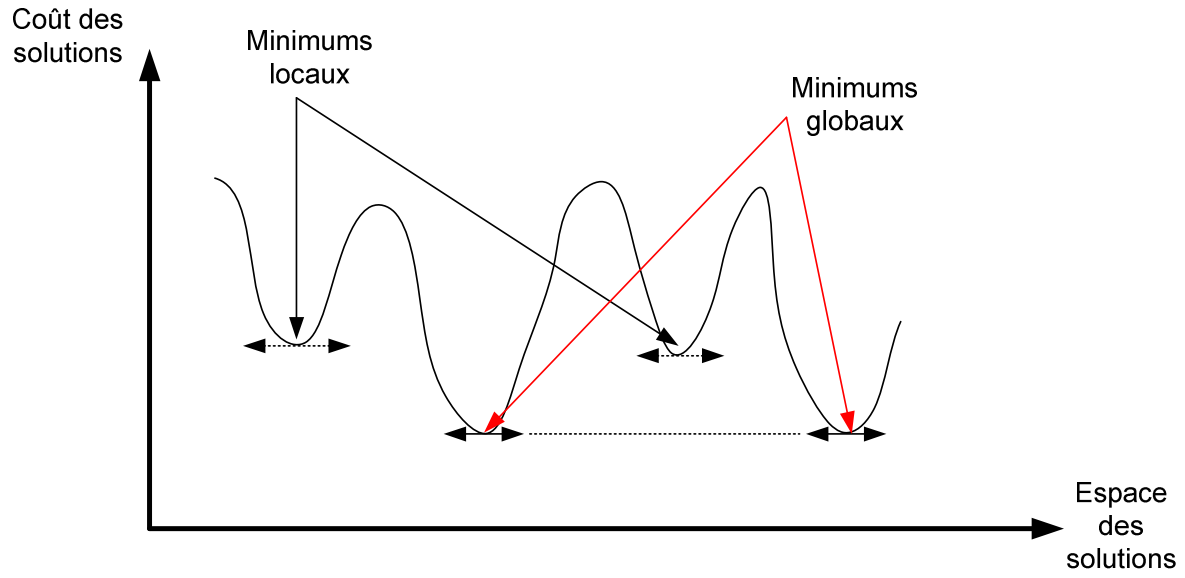


Figure 3-2 : Espace de recherche de solutions aux *COPs*

Nous ne souhaitons pas discuter ici le choix des méthodes à employer, tant il paraît dépendre de paramètres difficiles à appréhender. Par contre, nous pouvons constater que la plupart des méthodes approchées et plus particulièrement les métaheuristiques reposent sur la notion de système de voisinage. Le choix du voisinage influence grandement la performance des ces techniques.

3.2.1. Définitions

Définition : voisinage

On définit une fonction V de voisinage sur un espace de solutions E comme suit :

$$V : E \to E$$

$$x \in E \xrightarrow{V} V(x) = x' \in E$$

De cette définition, on note que la solution x' est accessible depuis la solution x par le biais de la fonction V, on nomme alors x' voisin de x. Pour chaque solution x on peut avoir un sous espace $E \subset E$ de voisins possibles de x par le biais du voisinage V. Théoriquement et sous certaines conditions une recherche itérative par voisinage peut conduire aux minimums globaux si la fonction de voisinage est bien définit et si l'espace est accessible à partir de chaque solutions. La Figure 3-3 montre qu'une succession de voisinages conduit à un minimum globale.

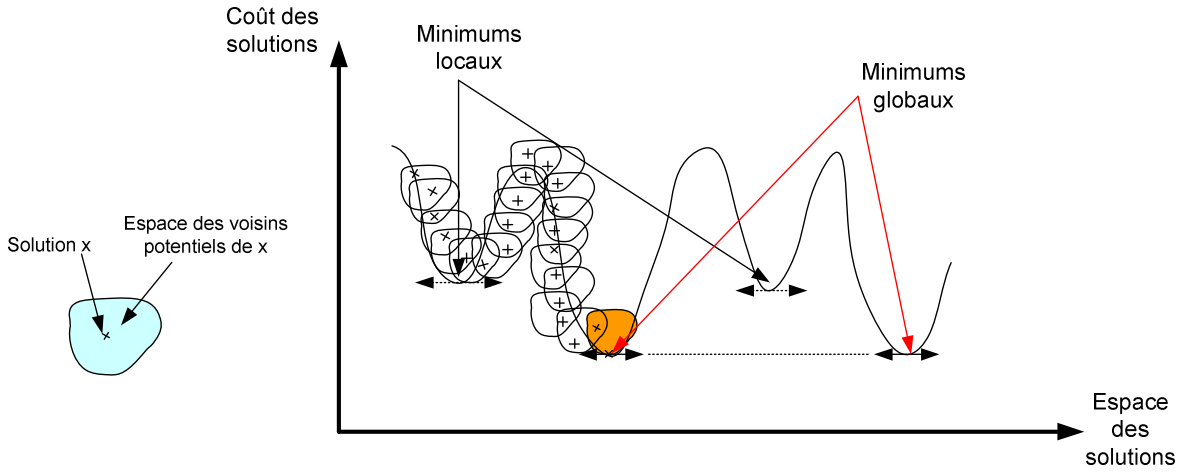


Figure 3-3 : Voisinage d'une solution x

Quand un minorant (respectivement majorant) d'un ensemble E existe on peut avoir les bornes inférieures (respectivement les bornes supérieures).

Définition : borne inférieure

Soit $\begin{matrix} f : E \to E \\ x \in E \xrightarrow{f} f(x) = y \in E \end{matrix}$ où E est un espace de recherche, f est une fonction du coût, une borne inférieure est le plus grand des minorant de la fonction f qu'on note LB tel que :
$x \in E \Rightarrow f(x) \geq LB$

Définition : borne supérieure

Soit $\begin{matrix} f : E \to E \\ x \in E \xrightarrow{f} f(x) = y \in E \end{matrix}$ où E est un espace de recherche, f est une fonction du coût, une borne supérieur est le plus petit des majorants de la fonction f qu'on note UB tel que :
$x \in E \Rightarrow f(x) \leq LB$

Définition : qualité d'une solution

La qualité d'une solution S qu'on notera $Q(S)$ est exprimée par l'écart de cette solution à sa borne inférieure divisé par sa borne inférieure LB, $Q(S) = \frac{S - LB}{LB}$ avec $LB \neq 0$.

Si le $Q(S) = \frac{S - LB}{LB} = 0$ alors, la solution S est optimale. La qualité augmente quand $Q(s) \to 0$.

3.2.2. Les heuristiques de construction

Ce sont des méthodes très simples à mettre en œuvre et qui dépendent du domaine d'utilisation. Elles sont souvent utilisées pour avoir rapidement des solutions admissibles (réalisables) de qualité relativement bonne, bien que, en générale leur performance ne puisse être garantie.

Ces heuristiques construisent des solutions selon des critères d'optimisation locaux. C'est-à-dire les heuristiques déterminent une solution selon une règle de construction donnée, mais n'essaient pas d'améliorer la solution obtenue. Une fois le choix de règle effectué, on ne le remet jamais en cause. De ce fait leur efficacité est relativement limitée. En effet, un bon choix local ne conduit pas nécessairement à un bon choix au niveau global.

Les heuristiques de construction sont souvent couplées avec d'autres méthodes. Elles sont utilisées pour obtenir des solutions de départ relativement de bonnes qualités, qu'on tente d'améliorer par la suite par des mécanismes de recherche locale par exemple.

3.2.3. Les méta-heuristiques

Meta-heuristique est un mot composé de méta et d'heuristique qui viennent du grec (au-delà) pour méta qui signifie à un plus haut niveau, et heuristique, (***heuriskein***), qui signifie ***trouvé***. Les méta-heuristiques sont des algorithmes généraux pouvant être utilisé pour une large gamme de problèmes. Pour toutes méta-heuristiques que nous citons dans cette partie, on donne leurs pseudo-codes en essayant de mettre en évidence leurs points forts et leurs points faibles.

Les méthodes de recherche locale

On les trouve aussi sous le nom de méthodes de trajectoire ou de voisinage dans Blum et Andrea (2003). Elles se basent sur le principe d'amélioration d'une solution de départ x à chaque pas de leur progression, on définit alors un voisinage $N(x)$ propre au problème étudié.

La recherche locale peut être guidée et ceci se manifeste par des critères à respecter dans le système de voisinage utilisé, ce qui donne la recherche locale guidée (***G***uided ***L***ocal ***S***earch ou ***GLS***).

Les méthodes de recherche locale itérée (***I***terated ***L***ocal ***S***earch ou ***ILS***) sont parmi les méta-heuristiques les plus simples. Le principe général de *ILS* est de construire itérativement un ensemble de minima locaux. L'Algorithme 3-1 décrit le principe des méthodes de type *ILS*.

Algorithme 3-1 : Recherche locale itérée

```
Nom de procédure : recherche_locale_itérée
   x :      Solution initiale
   y :      Recherche_locale(x)
    Début
      Tant que nécessaire faire
                x'     :=perturbation(y)
                y'     :=recherche_locale(y')
               y    := best(y,y')
      Fin Tant que
    Fin
```

Pour améliorer de manière significative la méthode de recherche locale, les perturbations ne doivent être ni trop faibles, ni trop fortes. Si elles sont trop faibles, elles ne permettent pas de s'extraire efficacement des minima locaux, restreignant l'exploration de l'espace de recherche. Si au contraire les perturbations sont trop fortes, la mémoire de la solution y est insuffisante et x' se rapproche d'une solution générée aléatoirement.

Le choix de l'état courant (procédure **best** dans l'algorithme de principe) peut être simplement l'état record (i.e. le meilleur minimum local trouvé jusque là). Les méthodes de type ILS offrent de nombreux avantages. Elles sont simples à programmer, comportent très peu de paramètres et s'adaptent très facilement à de nombreux problèmes.

Dans la suite nous détaillons les principales métaheuristiques. Ces méthodes d'optimisation illustrent parfaitement les principales méthodes de recherche locale existantes.

La descente stochastique

C'est une méta-heuristique très simple à mettre en œuvre. Elle part d'un principe qui est basé sur le voisinage d'une solution ***X***. Le voisin ***Y*** obtenu est acceptée si et seulement si ***Y*** est meilleur que ***X***. Le seul paramètre important d'un algorithme de descente stochastique est la méthode de sélection ou de génération d'un voisin. La descente a en revanche le principal inconvénient de rester bloquée dans un minimum local car elle n'autorise pas les voisins de moindre qualité que la solution en cours. L'Algorithme 3-2 présente le pseudo-code de la descente stochastique et la Figure 3-4 présente une trajectoire « classique » d'une descente stochastique.

Algorithme 3-2 : Descente stochastique

```
Nom de procédure : descente_stochastique
   X :   Solution initiale
   H :   Fonction de coût
    Début
      Tant que nécessaire faire
               Y:=voisin(X);
          Si  (H(Y)≤H(X)) alors
               X:=Y;
          Fin Si
      Fin Tant que
   Fin
```

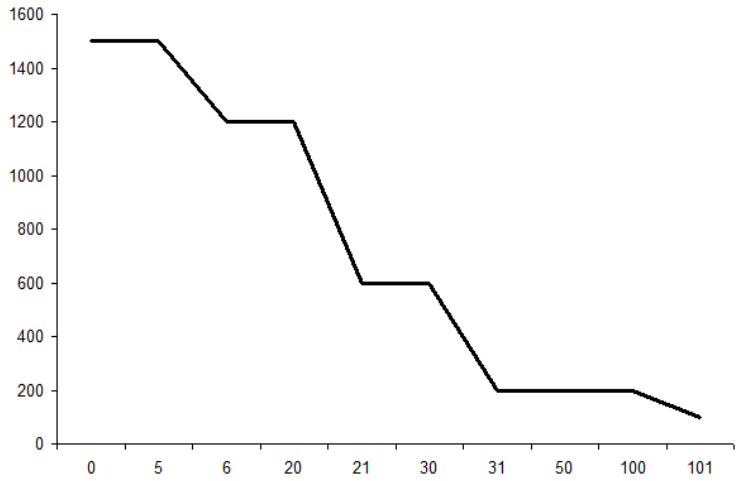


Figure 3-4 : Trajectoire d'une descente stochastique

Cet algorithme est facile à programmer et il permet d'améliorer rapidement une solution. Le choix de ***Y*** dans le voisinage de ***X*** peut ne pas être fait entièrement aléatoirement. On peut

envisager de mettre en place des méthodes exactes ou heuristiques pour choisir préférentiellement des éléments dans le voisinage de ***X***. Notons toutefois que la qualité du comportement d'un algorithme de descente stochastique dépend exclusivement du choix du système de voisinage.

Les descentes stochastiques successives

Cet algorithme consiste à effectuer plusieurs descentes stochastiques en partant de plusieurs solutions de départs différentes. Chaque descente nous conduisant dans un minimum local fait de cette méthode un algorithme aisément parallélisable.

L'algorithme des descentes stochastiques successives converge très lentement, en probabilité, vers l'optimum. Les descentes stochastiques effectuées sont indépendantes et sont réalisées à partir de solutions initiales différentes qui ne prennent pas en compte d'éventuelles informations sur les précédents minimas locaux trouvés. L'Algorithme 3-3 présente le pseudo-code des descentes stochastiques successives, qui, rappelons le, peuvent être parallélisées.

Algorithme 3-3 : Descentes stochastiques successive

```
Nom de procédure : descentes_stochastiques_successives
   X :   Solution initiale
   H     :  Fonction de coût
   X0:   Etat initial
    Début
      Tant que nécessaire faire
          Choisir un état X0
          X :=X0
          Tant que nécessaire faire
                      Y:=voisin(X);
             Si   (H(Y)≤H(X)) alors
                   X:=Y;
             Fin Si
          Fin Tant que
      Fin Tant que
    Fin
```

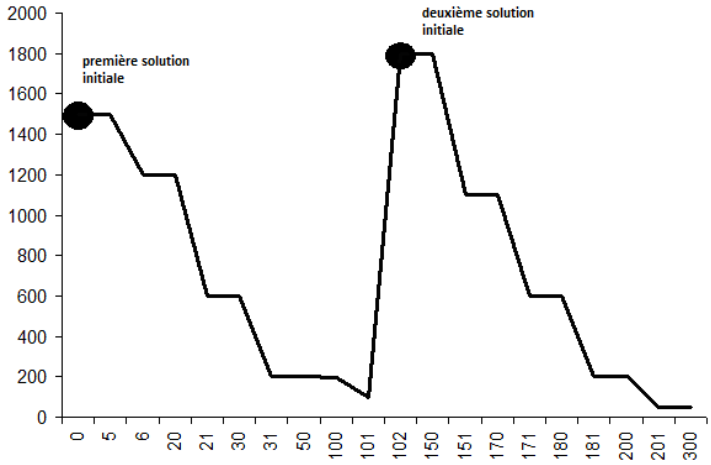


Figure 3-5 : Trajectoire de descentes stochastiques successives

La méthode de recuit simulé (en anglais Simulated Annealing)

Cette méthode s'appui sur un principe physique, qui est celui du recuit utilisé par les forgerons pour donner de la résistance et améliorer la structure du fer en alternant des périodes de chauffe (recuit) et de refroidissement, elle est également utilisée pour simuler le filtre de Metropolis, Metropolis et al., (1953). Les travaux de trois chercheurs, d'IBM présentent une version du recuit qui évite les minimums locaux Kirkpatrick et al., (1983). Cette méthode et sous certaines conditions peut atteindre l'optimum global avec une probabilité proche de 1 ce qui est démontré dans Aarts et Laarhoven (1985), Rinnooy Kan et Timmer (1987a) et Rinnooy Kan et Timmer (1987b). Cette propriété est un grand avantage par rapport aux autres méthodes qui ne garantissent pas d'arriver à un optimum global.

La recherche du minimum par l'algorithme du Recuit Simulé se fait à partir d'une solution (ou état) initiale ***X***. Si l'on considère un état ***X***, on choisit dans le voisinage de ***X*** noté V(X), un état voisin (dans un sens qui reste à préciser) nommé ***Y***. On accepte de transiter vers cet état ***Y*** si cet état est d'énergie (coût) inférieure ou égale à celle de ***X***. Si l'on note ***H(X)*** et ***H(Y)*** l'énergie respective des états ***X*** et ***Y***, cela revient à dire que ***H(Y)*** doit être inférieur ou égal à ***H(X)***. Dans le cas contraire, la transition est acceptée, bien qu'elle soit défavorable (augmentation de l'énergie), avec une probabilité ***P*** dépendant de la température ***T***, de ***H(X)*** et de ***H(Y)*** (Algorithme 3-4) avec $P = \exp(-\frac{H(Y) - H(X)}{T})$. On peut constater que la température ***T*** est un paramètre très important.

La probabilité P permet à l'algorithme d'admettre des transitions défavorables et donc de sortir d'un éventuel minimum local. Cette probabilité P dépend de la température T. Elle est d'autant plus grande que cette dernière est élevée. Nous constatons donc qu'un maximum de transitions défavorables sera accepté au début de l'algorithme et ceci afin de perdre la 'mémoire' de l'état initial. L'amélioration, à l'aide du recuit simulé, d'une solution de départ fourni par une heuristique ne présente donc pas d'intérêt puisque très rapidement l'algorithme perd la « mémoire » de l'état initial.

Algorithme 3-4 : Recuit simulé

```
Nom de procédure : recuit_simulé
     T   :  Température initiale
     X   :  Solution initiale
   Début
     Tant que nécessaire faire
              Y:=voisin(X);
         Si (H(Y) < H(X))alors
            X:=Y;
         Sinon
            Si random < P alors
              X:=Y
         Fin Si
         Modifier_Temperature(T)
     Fin Tant que
   Fin
```

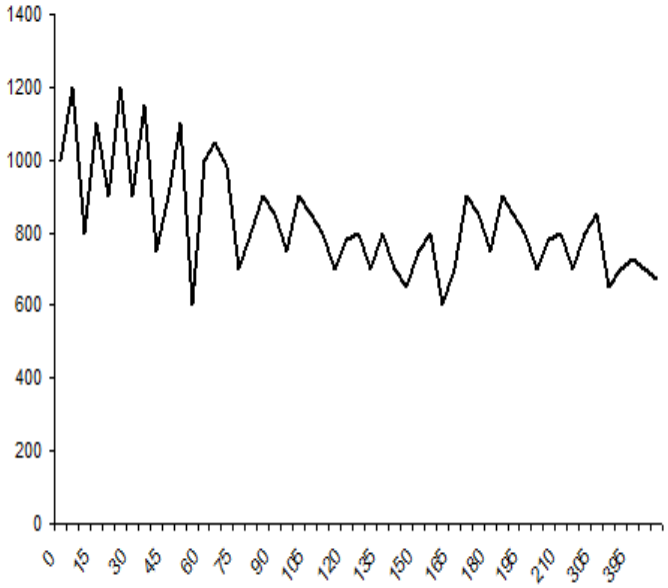


Figure 3-6 : Trajectoire d'un recuit simulé

Le principal inconvénient de cette méthode consiste dans le réglage des nombreux paramètres qui la composent, qui sont souvent choisis de façon empirique.

Les paramètres à régler dans le recuit simulé sont alors :

- la température initiale (température au début du processus d'optimisation) ;
- la température finale (température à laquelle le processus est considéré comme gelé) ;
- le critère de changement de température. Plusieurs façons de changer la température sont possibles, citons principalement les paliers de température (la température est constante pendant un certain nombre d'itérations puis décroît) et la décroissance géométrique (la température décroît à chaque itération grâce à un coefficient proche de 1 ou inférieur à 1) ;
- le critère d'arrêt du programme ;

- le choix d'un système de voisinage. Il s'agit sans aucun doute du paramètre le plus important.

Des considérations théoriques permettent de fixer chacun de ces paramètres afin que l'algorithme du recuit conduise à l'optimum Laarhoven et Aarts, (1987). Ces valeurs accroissent notablement les temps de calcul de sorte que l'on a souvent recours à l'expérience...

La méthode TABOU (en anglais Tabu (ou Taboo) search)

La méthode taboue ***TS (T***abu ***S***earch) a été formalisée pour la première fois par F. Glover Glover (1986), qui fonctionne avec une mémoire. A chaque mouvement on met à jour une liste de mouvements interdits dits tabous d'où le nom de la méthode. Les implémentations de la ***TS*** varient avec la taille, la variabilité et l'adaptabilité de la mémoire du problème étudié. On renvoi le lecteur intéressé par cette méthode à l'article Glover et Laguna (1997) qui fournit beaucoup de détails concernant la ***TS***.

Cette méthode élabore une liste des voisins avec *n* mouvements non tabous ; on sélectionne le meilleur voisin qui deviendra par la suite la solution courante, on insère alors le mouvement qui a donné le meilleur voisin dans la liste tabous, et on réitère ce processus jusqu'à la satisfaction d'un critère d'arrêt. L'Algorithme 3-5 donne le pseudo-code de la méthode TABOU.

Algorithme 3-5 : Méthode TABOU

```
Nom de procédure: TABOU_Search
   X     :  Solution initiale
   E(x)  := ø // liste de voisins de x par n mouvements ;
   L     =  ø // liste tabou vide au départ
    Début
      Tant que nécessaire faire
                E(x):=voisins(X);
          Selection(Y,E(x))
          Mettre_à_jour (meilleur_solution)
          Mettre_à_jour (L en insérant le mouvement (transition) de X➔Y)
          X :=Y
      Fin Tant que
   Fin
```

La méthode de Kangourou (en anglais Kangaroo)

Cette méthode étend et améliore *la descente stochastique* en autorisant la détérioration de la solution en cours pour ainsi éviter des blocages prématurés dans des minimums locaux. Elle se comporte à la manière d'un kangourou. Le principe de cet algorithme est d'effectuer une descente stochastique, et lorsque l'état minimal actuel est de même coût depuis trop longtemps, d'accepter une transition défavorable dans un voisinage de l'état actuel (pas nécessairement le même que celui utilisé pendant les descentes) et ceci quel que soit le coût. Ceci est appelé un saut. Ensuite on débute une nouvelle descente stochastique. L'Algorithme 3-6 donne le pseudo-code de la méthode Kangourou.

Cette méthode est publiée la première fois dans Fleury (1995). L'auteur souligne quelques problèmes pour lesquels le recuit simulé et le tabou ne sont pas adaptés. La Figure 3-7 illustre les sauts que réalise la méthode afin de sortir des minimums locaux.

Soit :

- ***V*** le système de voisinage utilisé lors de la descente ;
- ***W*** le système de voisinage utilisé lors des sauts ;
- ***A*** le nombre maximum d'itérations à effectuer avant un saut.

Le deuxième système de voisinage ***W*** utilisé par l'algorithme n'est pas nécessairement le même que ***V***. Le voisinage ***W*** doit être (afin que l'algorithme converge en probabilité vers l'optimum) tel que tout élément puisse être joint à tout autre par une chaîne finie d'état deux à deux voisins au sens de ***W***. Les Descentes Successives ne sont de fait qu'un cas particulier de l'algorithme du Kangourou, pour lequel ***W(x)*** pour tout ***x***, représente l'espace de recherche complet. Le fait d'effectuer des 'sauts' permet à l'algorithme de sortir d'une vallée c'est à dire d'un minimum local. A titre d'exemple la Figure 3-7 illustre la trajectoire suivie par un algorithme du Kangourou. L'avantage principal d'un algorithme du Kangourou est qu'il permet de minimiser des fonctions pouvant prendre des valeurs infinies. Lorsque l'algorithme se trouve dans une vallée contenant un optimum local, il effectue après A itérations sans amélioration un « saut ». Il existe donc une probabilité non nulle que l'algorithme effectue un saut prématurément, c'est-à-dire qu'il peut exister un point (ou plusieurs) de la vallée (partiellement explorée) qui corresponde à une valeur plus petite de la fonction objectif.

Algorithme 3-6 : Algorithme du Kangourou

```
Nom de procédure : KANGOUROU
     X   :  Solution initiale
    Début
      c   :=0
      Choisir un état X0
      X   :=X0
      Tant que nécessaire faire
               Si (c<A) Alors
               Choisir Y dans le V(X)
                      Si H(Y)≤H(X) Alors
                             Si H(Y)<H(X) Alors
                                 c:=0
                             Fin Si
                             X := Y;
                      Fin Si
                      c := c + 1
               Sinon (* Faire un saut*)
                      Choisir Y dans W(X)
                      Si H(Y)≠H(X) Alors
                             c:=0
                      Fin Si
                      X:=Y;
               Fin Si
      Fin Tant que
    Fin
```

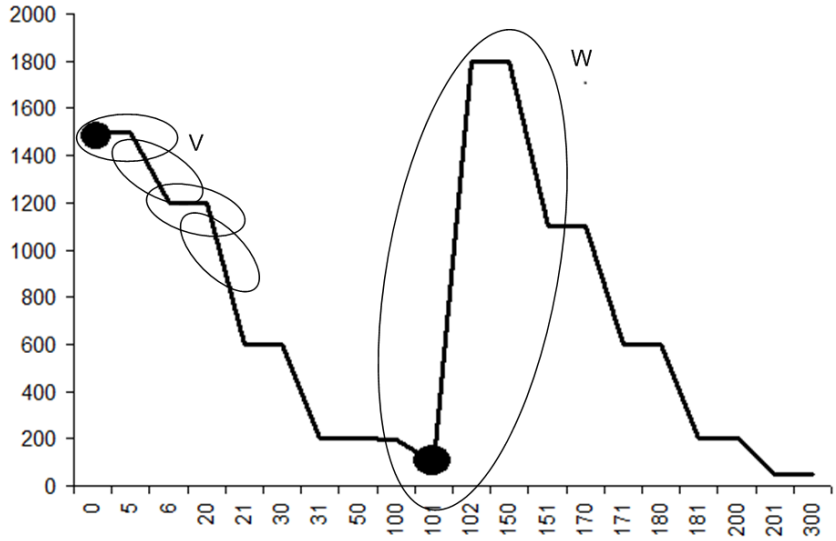


Figure 3-7 : Comportement typique d'un algorithme du Kangourou

L'algorithme génétique/ mémétique (en anglais Genetic/Memetic Algorithm)

Plus récente que les autres méta-heuristiques précédemment citées, elle s'appuie sur le célèbre principe énoncé Darwin (1859) celui de la sélection naturelle, de l'évolution et de l'hérédité. C'est une méthode à populations, où chaque individu appelé chromosome, représente une solution du problème. Par le mécanisme de croisement et d'hérédité on obtient de nouveaux chromosomes appelés fils (enfants). Avec une certaine politique on remplace les chromosomes parents par les chromosomes fils et ainsi cette population évolue vers une population contenant les meilleurs éléments héritant des meilleures qualités des individus quelle contient à chaque génération. Pour éviter un appauvrissement génétique de la population c'est-à-dire que la population converge vers les mêmes solutions (plusieurs individus identiques ou donnant la même solution) un mécanisme permettant l'injection de nouveaux individus est appelé à chaque fois que la nécessité se présente.

Cette méthode a été utilisée pour la première fois par Holland (1975) pour résoudre un problème d'optimisation. Elle a été adaptée la première fois pour le problème de job-shop par Lawrence (1985), puis par Goldberg (1989). On remarque un intérêt croissant pour la méthode ces dernières années, on peut citer notamment les travaux de Croce et al., (1995) , Abdelmaguid et al., (2004), Lacomme et al., (2009a) et Lacomme et al., (2010).

Dans un algorithme génétique, le croisement est utilisé comme un mécanisme de convergence de l'algorithme, alors que la mutation est utilisée pour créer de la diversité dans la population afin d'éviter un appauvrissement génétique. Par contre dans un algorithme mémétique les croisements jouent le rôle de création de la diversité, tandis que la mutation et la recherche locale sont utilisées pour créer la convergence de l'algorithme.

Ces deux algorithmes ont le même schéma d'optimisation sauf que l'algorithme mémétique possède une recherche locale. Le fonctionnement de l'algorithme génétique/mémétique ainsi que leurs implémentations seront expliqués en détail dans la suite, les résultats obtenus seront commentés et analysés. L'Algorithme 3-7 donne le schéma d'optimisation *de base* de *l'algorithme génétique*.

Algorithme 3-7 : Algorithme génétique

```
Nom de Procédure : Algorithme_génétique
        nmi    :   nombre maximal d'itérations de l'algorithme
        np     :   nombre maximal d'itérations avant le restart
        pm     :   probabiliter de mutation
        p      :   pourcentage de la population à remplacer
        pop    :   population
    Début
        Tant que ni<nmi faire
          Selection_des_parents (P1,P2)
          enfant := Croisement_de(P1,P2)
          muter(enfant) avec une_probabilité pm
          InsertSolution(Pop, enfant)
          trier(Pop)
          Si (npi=np) alors
             Restart(Pop,p)
             npi :=0
          Fin Si
          ni :=ni+1
        Fin Tant que
    Fin
```

La méthode de colonie de fourmis (en anglais Ant Colony)

Elle s'appuie sur le principe de l'auto-organisation : une suite de mouvements désordonnés localement qui contribuent à une organisation sans faille globalement. C'est une organisation où chaque individu a un rôle bien déterminé à jouer. Les colonies de fourmis, les abeilles sont des exemples pertinents de ces systèmes. Ces organisations ont passionné un grand nombre de chercheurs en biologie puis en éthologie pour percer le mystère de leur organisation réglée comme une très bonne horloge. Enfin des chercheurs en optimisation ont adapté le fonctionnement de ces systèmes pour aboutir à une méthode qui porte le nom de Colonie de fourmi ou (Ant Colony en anglais). Pour une lecture plus approfondie sur cette méthode voir Dréo et al., (2003) qui contient aussi d'autre références traitant spécialement des colonies de fourmis dont on peut citer par exemple : Hölldobler et Wilson (1990), Bonabeau et al., (1996), Bonabeau et al., (1998) et Bonabeau et al., (1999).

La méta-heuristique tente de simuler le fonctionnement d'une colonie de fourmis. Les fourmis sortent de leur nid pour chercher de la nourriture et déposent une substance chimique appelée phéromone tout au long de leur parcours. Les premières fourmis ayant trouvé la nourriture reviennent au nid et marquent deux fois leurs parcours. Cette substance guide le choix des trajets que vont effectuer les autres fourmis par deux mécanismes qui sont le renforcement et l'évaporation de la phéromone. Les fourmis sont capables de dépasser certains obstacles en faisant des ponts (constitué de plusieurs fourmis) et elles choisissent les branches les plus courtes comme le montre la Figure 3-8, ce qui constitue une propriété très importante en optimisation qu'est celle de la convergence. Cette méta-heuristique est beaucoup employée pour résoudre le problème de voyageur de commerce, et l'Algorithme 3-8 donne le schéma d'optimisation de la méthode de colonies de fourmis dans ce cas, par souci de clarté les fonctions ***Fville, Fevaporation et f_deposer_pistes*** ne sont pas détaillées. Pour une lecture plus approfondie on renvoie le lecteur à Dréo et al., (2003)

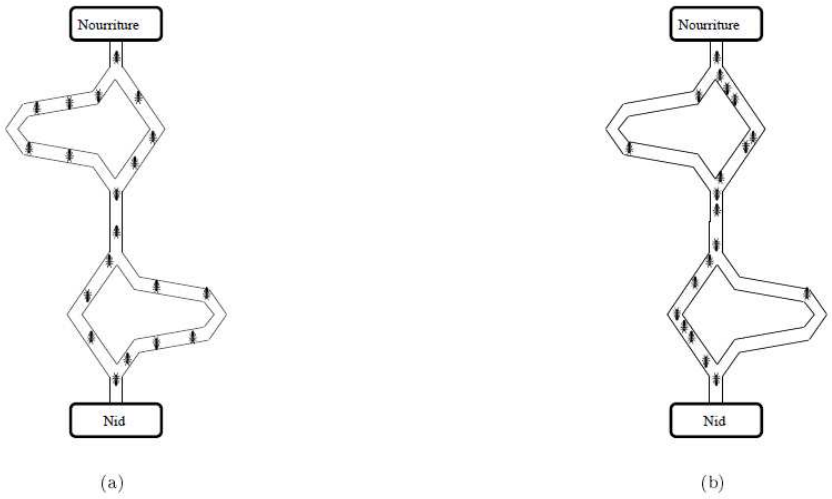


Figure 3-8 : Expérience de sélection des branches les plus courtes par une colonie de fourmis (a) au début de l'expérience, (b) à la fin de' l'expérience Dréo et al., (2003)

Algorithme 3-8 : Algorithme de colonies de fourmis de base : le « Ant systeme »

```
Nom de Procédure : Ant_system
  t                    : compteur d'itération de l'aglorithme ;
  tmax                 : nombre maximal d'itérations de l'aglorithme ;
  m                    : nombre de fourmis ;
  k                    : une fourmis k=1 ..m ;
  Fville               : formule de choix d'une ville ;
  Fdépose_piste        : formule de dépôt d'une piste sur un trajet ;
  Fevaporation         : formule d'évaporation des pistes ;
        Début
          Pour t=1 à tmax faire
           Pour k=1 à m faire
              Choisir une ville au hasard ;
                  Pour t=1 à tmax faire
                    Choisir une ville j dans la des villes
                    restantes selon la formule fville ;
                  Fin Pour
                    Déposer une piste sur un trajet selon la formule Fdépose_piste
           Fin Pour
             Evaporer les piste selon la formule Fevapoartion
          Fin Pour
        Fin
```

The greedy randomized adaptive search procedure (GRASP)

De l'anglais ***G***reedy ***R***andomized ***A***daptive ***S***earch ***P***rocedure (***GRASP***) est une méta-heuristique très employée pour résoudre les ***POCs***. Le ***GRASP*** est une méthode itérative où chaque itération est composée de deux phases : une première phase de construction où on fournit une solution admissible, suivie d'une phase de recherche locale où on tente d'améliorer la solution de la première phase. Pendant les itérations de la méthode le meilleur minimum local

obtenu est sauvegardé. Cette méta-heuristique a été introduite pour la première fois par Feo et Resende (1989). On trouve une bibliographie commenté du ***GRASP*** dans Feo et Resende (1995) et Festa et al., (2001). Les différentes implémentations sont données dans Resende et Ribeiro (2002)**.** Cette méthode a permis de donner de bons résultats pour beaucoup de ***POCs***. Trois paramètres (ListSize, MaxIter, Seed) sont à définir pour l'algorithme de GRASP. Ces paramètres sont respectivement la taille de liste des solutions candidates, le nombre maximal d'itérations est donné par la graine (seed, en Englais). Actuellement, on recense beaucoup de ***POCs*** où le GRASP est adapté avec succès et a permis d'obtenir de bons résultats ; on peut citer par exemple : Aiex et al., (2003) pour le problème du job-shop, et Prins (2009) pour le ***VRP.*** L'algorithme du principe du GRASP est décrit par algorithme 3-9.

Algorithme 3-9 : GRASP

```
Nom de procédure : GRASP (ListSize,MaxIter,Seed);
     Debut
         Initialiser la seed
         Pour i = 1 à MaxIter faire
            Pour j = 1 à ListSize faire
               Genere_sol(sol);
               Evaluer_SOL(sol, coût);
               Local_Search_Sol(sol);
               Si  Sol_record>sol alors
                   Sol_record:=sol;
               Fin Si
            Fin Pour
         Fin Pour
         Retourner Sol_record ;
     Fin
```

D'autres méthodes existent, on peut citer la méthode d'essaims particulaire qui se rapproche de la méthode des colonies de fourmis, et les approches utilisant l'intelligence artificielle. Actuellement on trouve plusieurs initiatives utilisant les systèmes multi-agents profitant du l'évolution de la performance des machines.

3.3. Conclusion

La Figure 3-9 tirée de Hao et al., (1999) donne un classement des méthodes selon leur appartenance à la recherche opérationnelle ou à l'intelligence artificielle, le classement donne aussi les années d'apparition des méthodes. Pour de plus amples informations concernant ce classement on vous renvoi vers l'article source.

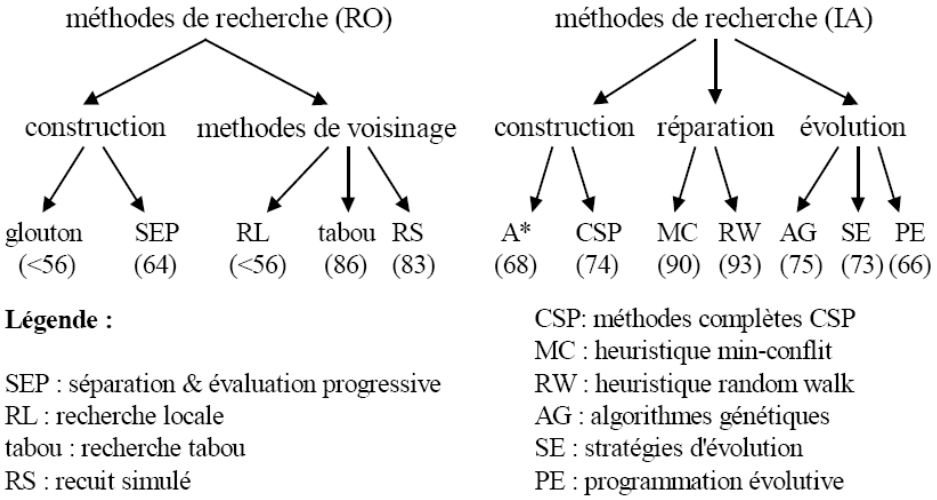


Figure 3-9 : Classement des méthodes de résolution Hao et al., (1999)

De nombreux méthodes et d'outils performants permettent de résoudre de manière efficace le problème du job-shop. Ces méthodes peuvent se subdiviser en deux grandes catégories, les méthodes exactes et les méthodes approchées. Les méthodes exactes fournissent pour un problème donné une solution optimale tandis que les méthodes approchées fournissent pour un problème donné une solution de *bonne qualité* qui est souvent non optimale. Dans cette deuxième catégorie, on peut distinguer les heuristiques et les métaheuristiques.

Nous avons vu que les méta-heuristiques utilisent la notion de voisinage et même parfois plusieurs, simultanément ou itérativement, comme le Kangourou. L'efficacité d'une méthode dépend en grande partie de son système de voisinage qui doit lui permettre d'explorer rapidement de bonnes solutions. Dans une situation idéale, le système de voisinage ne devrait contenir que des solutions meilleures que la solution courante, ce qui, associé à des mécanismes de diversification donnerait aux méta-heuristiques un pouvoir d'exploration élevé.

Dans le cadre du Job-Shop, il n'existe pas de voisnages fournissant uniquement des solutions de cout inférieur au cout de la solution courante, mais des voisinages évitant de générer des solutions dont on est certains qu'elles soient pires.

4. Les principaux voisinages proposés pour le Job-Shop

Comme nous l'avons mentionné le chemin le plus long du graphe conjonctif définit le makespan associé à ce graphe. Pour modifier le makespan d'une solution, il est nécessaire de modifier le chemin le plus long. Toute modification du chemin critique le casse et permet d'obtenir un nouvel ordonnancement, mais rien ne garanti que ce dernier soit égal, meilleur ou pire. Les voisinages présentés dans ce paragraphe exploitent cette propriété.

La notion de voisinage s'appuie sur des opérations ou des mouvements élémentaires pour passer d'une solution à une autre. Ces mouvements sont effectués sur des opérations appartenant à des blocs. Ces derniers sont formés d'une succession d'opérations s'exécutant sur une même machine. Dans la littérature plusieurs voisinages existent. On peut citer les voisinages Blazewicz et al., (1996) : N1 Laarhoven et al., (1992), N2 Matsuo et al., (1988), N3, N4 Dell'amico et Trubian (1993), N5 Nowicki et Smutnicki (1996). Ces voisinages permettent d'obtenir des solutions voisines en faisant des opérations simples sur le chemin critique. Dans le cas où on fait une seule opération le voisinage est dit en ***1 OPT***, il est en ***2 OPT*** s'il requière 2 opérations et en ***3 OPT*** trois mouvements élémentaires sont faits pour avoir la solution voisine et ainsi de suite. Une brève description de ces voisinages est donnée ci-après.

4.1. Voisinages *N1* et *N2*

Le voisinage de type ***N1*** consiste à permuter deux opérations consécutives traitées par la même machine et le long du chemin critique. Soit deux opérations O_1 et O_2 appartenant au bloc critique d'une solution S_1. O_1 est avant O_2 dans S_1 et ceci est schématisé par un arc $O_1 \rightarrow O_2$. Le voisinage de type ***N1*** permet d'avoir une solution S_2 en inversant l'arc $O_1 \rightarrow O_2$ en $O_1 \leftarrow O_2 \Leftrightarrow O_2 \rightarrow O_1$. La réévaluation de l'ordonnancement est ainsi obtenue avec ce seul changement. Ce voisinage est introduit par Laarhoven et al., (1992), la démonstration de la connectivité ou l'accessibilité de ce voisinage est donnée dans Hurink et Knust (2005). Ainsi, il est possible de trouver la solution optimale en appliquant itérativement ce voisinage.

Laarhoven et al., (1992) ont utilisé ce voisinage pour mettre en œuvre un recuit simulé. De même Lourenço (1995) propose une optimisation locale utilisant une descente stochastique et des grands sauts (large step optimization).

Pour restreindre le nombre de voisins dans le système de voisinage ***N1*** Matsuo et al., (1988) cité par Blazewicz et al., (1996) proposent un voisinage dérivé de ***N1*** appelé ***N2***. Ce dernier, utilisé dans un recuit simulé, consiste à inverser les opérations qui se situent sur les extrémités des blocs. La réduction de nombre de voisins est faite au détriment de la connectivité car ce dernier ne la vérifie pas Schmidt, (2001). La procédure de séparation et évaluation proposée par Brucker et al., (1994) utilise le même principe pour le branchement lors de l'exploration de l'arbre. Les voisinages ***N1*** et ***N2*** sont des voisinages en ***1 OPT***.

4.2. Voisinages *N3* et *N4*

Le voisinage ***N3*** proposés par Dell'Amico et Trubian Dell'amico et Trubian (1993) est en ***3 OPT***. Dans le bloc critique on fait trois inversions différentes pour obtenir le voisinage ***N3***. Soit *(i, j)* un arc disjonctif sur le chemin critique. Nous définissons respectivement *p(i)* et *s(i)* comme le prédécesseur et le successeur de *i* sur la même machine *m*. Toutes les permutations nécessaires des trois opérations {*p(i), i, j*} et {*i, j, s(j)*} sont réalisées pour inverser l'arc *(i, j)*. Le voisinage ***N4*** est une variante du voisinage ***N3*** en ne travaillant que sur les extrémités des blocs et consiste à

déplacer une opération à l'intérieur du bloc vers le bord d'un bloc. Ce voisinage est utilisé par Dell'amico et Trubian (1993) dans un algorithme tabou et ils ont montré que ce voisinage était lui aussi faiblement connecté (i.e. qu'il permet toujours d'obtenir l'optimum en un nombre fini d'étapes). Ce voisinage génère beaucoup de solutions possibles car pour chaque opération du bloc critique hormis les opérations des extrémités du bloc deux inversions sont possibles soit au début soit à la fin du bloc.

4.3. Voisinage *N5*

Le voisinage ***N5*** est proposé par Nowicki et Smutnicki Nowicki et Smutnicki (1996). Il essai de remédier aux défauts des voisinages précédents. Il est démontré qu'il domine le voisinage ***N1*** et ***N2***. Cela signifie que toute amélioration par un voisin des voisinages ***N1*** et ***N2*** est dans le voisinage ***N5***. Les auteurs ont montré que ce voisinage est faiblement connecté, c'est-à-dire qu'il permet d'atteindre l'optimum en un nombre fini d'étapes.

Dans le système ***N5*** on génère arbitrairement un seul chemin critique ; puis on définit tous les blocs et inverse les opérations comme suit :

- Si le bloc n'est ni premier n'est ni dernier on inverse les deux extrémités ***2 OPT*** ;
- si le bloc est premier dans le chemin critique on inverse les deux opérations de la fin du bloc ***1 OPT*** ;
- si le bloc est dernier dans le chemin critique on inverse les deux opérations de début de bloc ***1 OPT***.

La Figure 4-1 montre les différentes opérations à réaliser pour obtenir le voisinage ***N5*** sur un bloc composé de 5 opérations machines.

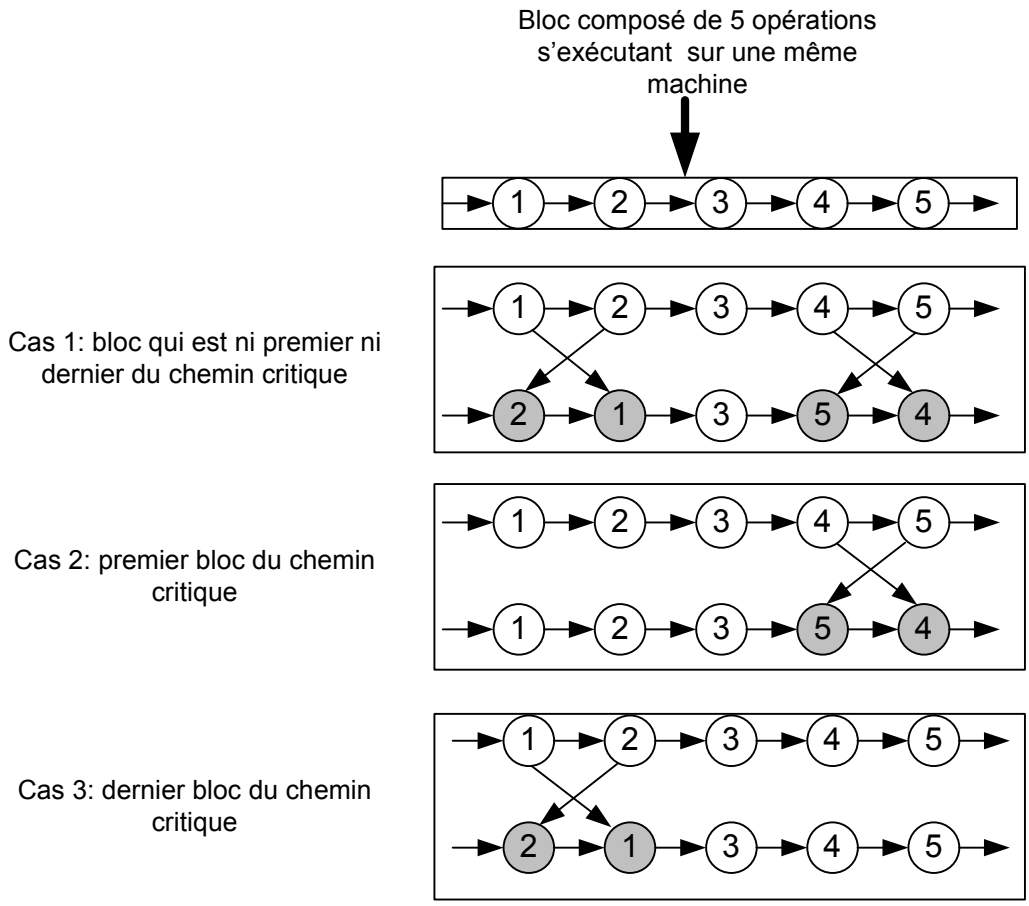


Figure 4-1 : Les différents mouvements à réaliser selon le cas dans le voisinage NS

Nowicki et Smutnicki (1996) ont utilisé le voisinage ***N5*** pour réaliser un algorithme tabou. Jusqu'à aujourd'hui leur algorithme est considéré comme étant un des algorithmes les plus performants pour le problème de job-shop Blazewicz et al., (1996), Caumond (2006).

Pour tester et comparer les méthodes entre elles, des benchmark sont requis le paragraphe suivant liste les différents benchmark qui existent pour le problème de Job-Shop.

4.4. Les benchmarks

Les benchmarks sont utiles pour connaître les performances des méthodes de résolution. Dans la littérature plusieurs benchmarks existent pour les problèmes de recherche opérationnelle. L'essentiel d'entre eux sont regroupées sur le site[2] de l'OR Library où il est possible de les télécharger. On y trouve aussi des résultats et des références des auteurs des benchmarks. Ainsi pour ce qui concerne le Job-Shop, on peut télécharger un fichier contenant 82 instances de différentes tailles regroupant les principaux benchmarks de la littérature[3] et donnant les références des sources de ces instances. Les instances portent des noms composés des lettres qui représentent souvent les initiales des noms de leurs auteurs et des chiffres pour les différencier entre elles. Elles sont composées de n jobs et de m machines et leur taille est donnée par $n \times m$.

[2] Site de l' OR Library :http://people.brunel.ac.uk/~mastjjb/jeb/info.html,

[3] Page sur laquelle sont disponibles les problèmes du Job-Shop :http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt

Les instances utilisées pour les benchmarks de la littérature sont :

- abz5 à abz9 introduites par Adams et al., (1988),
- ft06 (6X6), ft10 (10X10), ft20 (20X5) introduite par Fisher et Thompson (1963)
- la01 à la40 constituent 40 instances de différentes taille (10x5, 15x5, 20x5, 10x10,15x10,20,10,30x10, et 15x15) proviennent de Lawrence (1985) ;
- orb01 à orb10 proviennent de Applegate et Cook (1991),
- swv01-swv20 introduite par Storer et al., (1992),
- yn1-yn4 introduite par Yamada et Nakano (1992).

5. Les extensions possibles

Comme on vient de voir ci-dessus le job-shop figure parmi les premiers problèmes à être étudié. Sous sa forme de base le modèle théorique du job-shop ne couvre pas les contraintes réelles des SFP. Pour rendre ce modèle applicable aux SFP plusieurs extensions ont été apportées, chacune d'elles tente de couvrir un aspect pratique de ces derniers. Dans ce qui suit nous essaierons de donner les extensions les plus importantes du job-shop et pour chacune d'elles une brève description ainsi que les références de quelques articles traitant l'extension en question. Ainsi on trouve les extensions suivantes :

- sequence-dependent ou setup times ;
- les machines parallèles (Multiprocessors),
- date de fin au plus tard (Job release times),
- les fenêtres de temps (Processor time windows),
- buffers d'entrée/sortie limités et des contraintes bloquantes (Limited buffers and blocking constraints),
- la généralisation de sans attentes (Generalized no-wait constraints) ,
- la flexibilité des machines (Processor flexibility) ;
- l'inclusion du transport et les temps de transfert (Transport and transfert times) ; le troisième chapitre est entièrement dédiée à cette dernière extension qui constitue notre problématique.

5.1. Sequence-dependent setup times

Cette extension inclue les temps non opératoires tels que la préparation des outils, la maintenance et le réglage des machines. Souvent après chaque opération une phase de réglage ou d'alimentation des machines en matière première ou consommable est requise. Dans le milieu industriel, tous ces temps sont regroupés sous l'appellation ***SETUP-TIME***, en informatique par exemple un calcul scientifique exécuté par un ordinateur requière un temps de chargement/déchargement des données du problème. Ces temps étaient négligés dans le job-shop classique. Le setup-time est dit séparable si ce dernier peut être effectué pendant que le prédécesseur de l'opération en question est en exécution sur la machine, dans les autres cas il est inséparable du processing time. Le setup time dépend du job s'exécutant sur une machine, mais il peut aussi dépendre du job prédécesseur sur la même machine et la on parle de séquence dépendant setup times.

5.2. Multiprocessors

Lorsqu'une opération requière plusieurs machines (processors) pour son exécution, on parle du multi processor job-shop, cela se rencontre par exemple dans le cas d'une opération chirurgicale nécessitant la coopération de plusieurs chirurgiens, d'anesthésistes et d'infirmiers, ou encore c'est le cas d'une opération qui pour être réalisée, nécessite un ou plusieurs opérateurs utilisant des machines, pour plus d'information on renvoie le lecteur à Drozdowski (1996).

5.3. Processor time windows

Cette extension se rencontre lorsqu'on fourni pour chaque machine une ou plusieurs fenêtres de temps pendant lesquelles elles sont disponibles pour exécuter les opérations. En dehors de ces fenêtres les machines sont considérées comme indisponibles.

5.4. Limited buffers and blocking constraints

En limitant la capacité des buffers d'entrée/sortie des machines, on augmente la complexité du problème car cette extension ajoute des contraintes bloquantes. Si elles sont prises en considération des machines peuvent se trouver dans l'état de blocage car elles ne peuvent pas transférer les jobs ayant fini leurs exécutions vers des buffers de sorti qui sont pleins, cette situation peut générer aussi la saturation des buffers d'entrées des machines en question. Le challenge est alors de trouver des solutions évitant le blocage des machines tout en minimisant le Makespan.

5.5. No-wait constraints

Dans le No-wait job-shop, il se peut y avoir du temps d'attente entre les opérations successives d'un même job, ceci se manifeste par des contraintes technologique à satisfaire, par exemple en métallurgie, la pièce du métal est travaillée à une certaine température, donc entre l'opération de chauffe et le passage à la presse on ne tolère pas de temps d'attente, c'est le cas aussi de l'industrie chimique, ou certaines réaction ne tolère pas de délai d'attente pour garantir la qualité attendue.

5.6. Job-Shop flexible

On parle du Job-Shop flexible lorsqu'on dispose de machines capables d'effectuer la même opération, même si leurs performances ne sont pas équivalentes (c'est-à-dire que la durée opératoire peut être différente d'une machine à une autre pour une même opération), ainsi pour chaque opération on donne un ensemble non vide de machines capables de la réaliser avec les temps opératoires relatifs à chacune d'entre elle. Donc, aux contraintes du Job-Shop classique, s'ajoutent des contraintes d'affectation des machines aux différentes opérations.

5.7. Transport and transfert times

Les contraintes de transport constituent l'extension la plus naturelle du job-Shop, car dans sa version classique, le Job-Shop ne tient pas compte du temps de transport des jobs entre les différentes machines et les jobs sont supposés naissant sur les machines. Cette extension génère de nouvelles contraintes disjonctives entre les ressources de transport, qui sont souvent des robots transporteurs, on les trouve aussi sous l'appellation d'AGV (pour ***A***utomated ***G***uided ***V***ehicule) ou sous l'appellation de chariots filoguidés. Dans le cas où on dispose de plusieurs transporteurs, des contraintes liées à l'affectation des opérations de transport aux transporteurs

s'ajoutent aux contraintes du Job-Shop classique. Cette extension est étudiée en détail dans le chapitre suivant.

6. Conclusion

Dans notre étude de l'ordonnancement nous avons accordé notre attention aux ateliers flexibles de production, et plus particulièrement au problème de job-shop comme problème modélisant le mieux les SFP. Ce problème a été étudié en donnant un historique de son apparition, ainsi que sa formalisation linéaire tout en recensant les différentes contraintes du problème. Sa modélisation sous forme de graphe disjonctif a également été évoquée ansi que de nombreuses méthodes permettent de résoudre le problème du job-shop.

Dans ce chapitre, nous avons présenté les méthodes approchées et en particulier mis en évidence que le graphe disjonctif et les voisinages basés sur la notion de blocs et sur le chemin critique sont des approches très efficaces.

Nous avons également présenté les benchmarks qui existent et les différentes extensions possibles. La dernière extension se rapportant au transport constitue notre sujet d'étude à laquelle le chapitre suivant est entièrement consacré.

Chapitre III : Le problème de Job-shop avec transport

Ce chapitre est consacré à la présentation et à la résolution du problème de job-shop avec transport et plusieurs chariots.

Liste des tableaux

Liste des figures

Liste des algorithmes

1. Introduction

Dans ce chapitre, nous nous intéressons au problème du job-shop avec transport. En appliquant une démarche de modélisation au problème d'ordonnancement des systèmes flexibles de production, ce problème apparaît naturellement comme un raffinement du modèle de job-shop classique lors de l'étude des systèmes flexibles de production. Or, le problème de job-shop est central en ordonnancement car c'est un problème d'atelier général et qu'il est largement étudié dans la littérature. Ainsi, de nombreuses méthodes efficaces ont été proposées pour sa résolution (cf.§2.2). Nous proposons donc d'étendre les méthodes existantes pour le problème de job-shop afin de prendre en compte le transport avec plusieurs chariots.

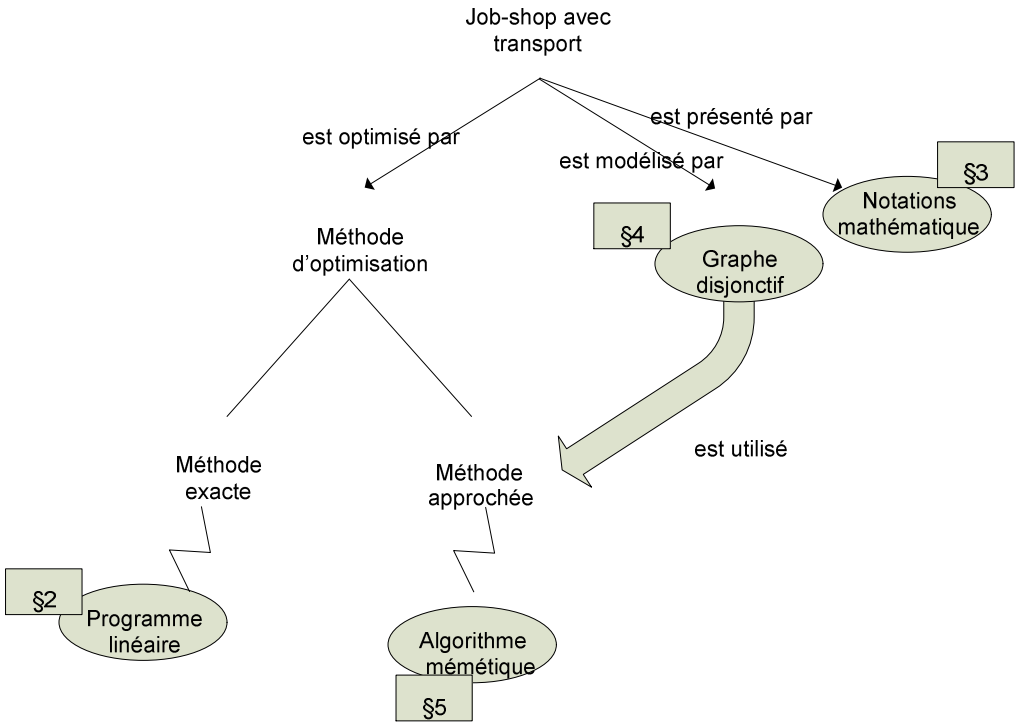


Figure 1-1 : Synoptique de nos propositions pour le job-shop avec transport

La Figure 1-1 est un synoptique de nos propositions pour le problème de job-shop avec transport. Ces propositions concernent principalement :

- un premier programme linéaire en nombres entiers (cf. §2) permettant de trouver la solution optimale des instances à moins de 5 jobs,
- un modèle de graphe conjonctif - disjonctif pour la conception de méthodes efficaces d'optimisation (cf. §3),
- et un algorithme génétique pour la résolution approchée d'instances de grande taille (cf. §5).

L'extension que nous proposons concerne l'inclusion du transport des jobs entre les différentes machines dans le problème du job-shop. Le transport constitue une extension naturelle du job-shop classique. Du fait de la difficulté que cet ajout génère, on ne trouve pas beaucoup de publications dans la littérature. Dans la suite de ce chapitre on essaie d'étendre les modèles existant du job-shop, tels que le graphe disjonctif, le chemin critique et les notions de blocs. Nous introduisons aussi une modélisation mathématique du problème, une modélisation du problème sous forme d'un graphe disjonctif, et un schéma de résolution basé sur l'utilisation

des heuristiques et des méta-heuristiques. Ce chapitre se termine par une évaluation numérique de nos approches par rapport aux résultats de la littérature.

2. Le job-shop avec transport : présentation

2.1. Définitions et notations

On considère un problème $\wp$ de job-shop avec transport et plusieurs robots. On dispose de m machines pour réaliser le traitement de n jobs. Le système est muni d'un ensemble de k robots qui réalisent les opérations de transport/manutention. Le job-shop avec transport consiste à ordonnancer l'ensemble des n jobs sur l'ensemble des m machines, les jobs étant transportés entre les différentes machines par les k robots transporteurs. Chaque job J_i est composé de n_i opérations notées $O_{i,1}$, $O_{i,2}$,…, O_{i,n_i} à réaliser dans un ordre donné (contraintes de précédence), appelé la « gamme du job » : $O_{i,1} \to O_{i,2} \to ... \to O_{i,n_i}$. Chaque opération $O_{i,j}$ du job J_i doit s'exécuter sur la machine $\mu_{i,j}$ sans préemption pendant une durée $p_{i,j} > 0$. Chaque machine ne peut exécuter qu'une seule opération à la fois. On considère que chaque machine dispose de stocks (buffers) illimités d'entrée/sortie.

De plus, on prend en compte le transport entre deux machines. Soit deux opérations successives $O_{i,k}$ et $O_{i,(k+1)}$ d'un même job exécutées sur deux machines $\mu_{i,k}$ et $\mu_{i,k+1}$. Entre ces deux opérations a lieu le transport du job entre le stock de sortie de la machine $\mu_{i,k}$ et le stock d'entrée de la machine suivante $\mu_{i,k+1}$. La durée du déplacement à charge du robot r de la machine $\mu_{i,k}$ vers la machine $\mu_{i,k+1}$ est donnée par $t^r_{\mu_{i,k},\mu_{i,k+1}}$. Chaque robot ne peut transporter qu'un seul job à la fois. Entre deux déplacements à charge le robot se déplace à vide de la machine i vers la machine j. Ce temps de déplacement à vide est donné par $v^r_{i,j}$. On suppose que $v^r_{i,i} = 0$ et $t^r_{i,j} \geq v^r_{i,j}$ pour chaque robot r.

Ainsi, chaque job J_i a un instant donné peut être soit : *(i)* traité par une machine ; *(ii)* transporté par un robot ; *(iii)* dans le buffer d'entrée d'une machine ; *(iv)* dans le buffer de sortie d'une machine.

Les jobs attendent dans les buffers de sortie des machines la disponibilité des robots qui leurs sont affectés. De même les jobs déposés sur les machines doivent attendre dans les buffers d'entrée la disponibilité de la machine pour être traités. On note que l'ordre d'arrivée des jobs peut différer de l'ordre d'exécution par les machines et les robots. Les temps de transport satisfont l'inégalité triangulaire et les temps de chargement (respectivement déchargement) sont inclus dans les temps de déplacement. Toutes les variables $p_{i,j}$, $t^r_{i,j}$, $v^r_{i,j}$, $C_{\max} = \max_{j=1,\ldots,n}\{C_j\}$, C_j, $j = 1...n$ étant la date fin de la dernière opération, sont supposées être des variables entières et non négatives.

D'après la notation $\alpha / \beta / \gamma$ le problème étudié peut être noté $JR\left|t_{k,l}, t'_{k,l}\right|C_{\max}$ (nous invitons le lecteur à consulter Knust (1999) pour plus d'informations concernant la complexité des problèmes d'atelier avec transport). Dans cette notation, J est utilisé pour spécifier qu'il s'agit d'un problème de job-shop ; R qu'il s'agit d'un nombre limité de robots identiques ; $t_{k,l}$

indique que les temps de transports ne dépendent pas des jobs mais de l'emplacement des machines et $t'_{k,l}$ indique que les déplacements à vide dépendent de la topologie de l'atelier et des robots, le critère à optimiser est la minimisation du temps de fin de la dernière opération dans le système appelé makespan $C_{\max}$.

Pour ce qui nous concerne, les robots peuvent être différents dans le sens où leurs vitesses de déplacement pour le même job et le même déplacement peuvent être différentes. L'utilisation des robots identiques est un cas particulier de notre problème.

2.2. Etat de l'art

De nombreuses études tendent à inclure le transport, lors de la modélisation et de la résolution des problèmes d'ordonnancement, en se ramenant à des problèmes de flow-shop ou de job-shop. Le problème du job-shop avec transport a été, pour la première fois, formalisé par Knust (1999) et étudié dans le cas d'un seul robot dans Hurink et Knust (2002), Hurink et Knust (2005). Plusieurs travaux proposent des contraintes additionnelles pour prendre en compte le transport. Dans Strusevich, (1999) les auteurs supposent qu'il existe un décalage (lag) connu et déterministe entre la fin d'une opération et le début de l'opération suivante d'un même job. Ce décalage (lag) entre deux opérations consécutives du même job est appelé temps de transport. Cependant, il est considéré que le moyen de transport est toujours disponible. D'autres études prennent en compte le fait qu'un transporteur ne peut déplacer qu'un seul job à la fois dans le contexte d'un flow-shop Hurink et Knust (2001) et dans le cadre du job-shop avec transport Hurink et Knust (2002), Hurink et Knust (2005) où les auteurs introduisent une modélisation sous la forme d'un graphe disjonctif avec deux types de nœuds : des nœuds représentant des opérations sur les machines et des nœuds représentant les opérations de transport. L'étude des problèmes d'ordonnancement incluant les opérations de transport dans les flow-shops a fait l'objet d'une attention particulière ces dernières années et un état de l'art est proposé dans Crama et al., (2000). Ces articles traitent plus particulièrement de problèmes avec un unique transporteur et prennent en compte le transport à charge ainsi que le transport à vide. Kise (1999) a démontré que la minimisation du makespan dans un flow-shop à deux machines, avec temps de transport et un unique transporteur est déjà un problème NP-Difficile. D'autres résultats de complexité sont proposés dans Hurink et Knust (2001). King et al., (1993) propose un algorithme de Branch and Bound pour un problème de flow-shop avec un unique transporteur.

Langston (1987) propose des heuristiques pour la résolution de flow-shop flexible à deux étages et temps de transport inter-étages. Brucker et al, (2003), Nowicki (1999) ont introduit des graphes disjonctifs pour le flow-shop avec stocks de capacité limitée et ont proposé des méthodes de résolution. A notre connaissance, les seuls travaux concernant le job-shop avec un unique transporteur et des contraintes de capacité sur les stocks avec la politique de gestion PAPS sont ceux de Caumond et al., (2009) où les auteurs proposent un modèle linéaire mixte dont la résolution directe permet d'obtenir les solutions optimales d'instances ayant 5 jobs soit environ 111 opérations à ordonnancer.

Un problème similaire est le problème d'ordonnancement dans les systèmes flexibles de production avec plusieurs transporteurs identiques. Ce problème d'ordonnancement introduit pour la première fois en 1993 par Ulusoy et Bilge Bilge et Ulusoy (1993) a fait l'objet de plusieurs publications dans la littérature Bilge et Ulusoy (1995), Bilge et al., (1997), Abdelmaguid et al., (2004), Deroussi et al., (2008). Le job-shop avec transport et plusieurs robots différents est traité dans Lacomme et al., (2007), Lacomme et al., (2008) et Lacomme et al., (2010). La section

suivante présentera une modélisation mathématique du problème du job-shop avec plusieurs transporteurs.

3. Formalisation linéaire

Dans cette partie on donnera une formalisation linéaire du problème définit ci-dessus.

On suppose qu'on dispose, au début de l'ordonnancement, de toutes les ressources et en quantités suffisantes, que tous les jobs sont disponibles, que les buffers d'entrée/sortie des machines sont de capacités illimités et les temps de transport dépendent de la topologie de l'atelier et des robots, et satisfont l'inégalité triangulaire. Cette dernière stipule que le temps nécessaire pour aller d'une machine i vers une machine j directement doit être inférieur ou égale au temps nécessaire pour relier les deux machines (i, j) en faisant un détour par une autre machine k.

Cette situation peut se rencontrer dans la réalité et induit en erreur le processus d'évaluation du plus long chemin dans le graphe. Plusieurs travaux supposent que les temps du déplacement respecte cette inégalité ou prévoient des traitements si linégalité se présente en privilégiant de faire le détour au lieu de faire le déplacement directe comme dans Knust (1999).

Pour ce qui nous concerne on suppose que ces déplacements respectent l'inégalité triangulaire. Ceci peut être garanti en pré-calculant les plus courts chemins entre les machines.

Pour présenter notre modélisation linéaire nous nous servons de cinq types de variables de décision $(t_i, t'_i, c_i, c'_i, C_{\max})$ et de cinq variables binaires $(b_{i,j}, d_{i,j}, f_{i,k}, e_{i,j}, g^k_{i,j})$, les temps des déplacements à charge des robots sont donnés par les variables $t^k_{i,j}$, et les temps des déplacements à vide sont donnés par les variables $v^k_{i,j}$. Les variables continues sont liées aux dates de début des opérations machine t_i et les dates de début des opérations transport t'_i. Dans la suite nous donnons la définition de toutes les variables utilisées et la modélisation du problème.

3.1. Notations

J : ensemble de jobs; $J = \{J_1, J_2, ..., J_n\}$;

M : ensemble de machines ; $M = \{M_1 M_2, ... M_m\}$;

R : ensemble de robots ; $R = \{R_1, R_2, ... R_K\}$;

O : ensemble des opérations machine $i \in O$ tel que $\exists j \in J$, $\exists k \in [1, n_j]$, avec $i = O_{j,k}$ où $O_{j,k} \in O$ représente la $k^{ième}$ opération machine du job j ;

P : ensemble des opérations machine qui n'ont pas de prédécesseurs dans O. $i \in P$ signifie que $\exists j \in J$, tel que $i = O_{j1}$ est la première opération machine du job j.

U : ensemble d'opérations machines qui n'ont pas de successeurs dans O. $i \in U$ signifie que $\exists j \in J$, tel que $i = O_{jn_j}$ est la dernière opération du job j.

T : ensemble des opérations de transport $i \in T$ signifie que $\exists j \in J$, $\exists k \in [1, n_j - 1]$, tel que $i = \tau_{j,k}$ où $\tau_{j,k}$ représente la $k^{ième}$ opération de transport du job j cette opération (τ_{jk}) est entrelacée entre les opérations machines $O_{j,k}$ et $O_{j,(k+1)}$;

E : ensemble de toutes les opérations du problème $E = O \cup T$;

λ : est une bijection : $\lambda : O \leftrightarrow T$, liant chaque opération machine i à son successeur

transport immédiat dans la gamme du job $\forall i \in U / i = O_{j,k}$, $\lambda(i) = \tau_{j,k}$;

p_i : temps opératoire de l'opération machine $i \in O$;

μ_i : la machine où s'exécute l'opération machine $i \in O$

$\tau_{i,j}^k$: $j^{ième}$ opération de transport du job i affecté au robot k

$\tau_{i,j}$ $j^{ième}$ opération de transport du job i avant de lui affecter un robot

$t_{i,j}^k$ durée de déplacement à charge du robot k de la machine i vers la machine j, incluant le temps de déchargement.

$v_{i,j}^k$ durée de déplacement à vide du robot k de la machine i vers la machine j, incluant le temps de chargement.

$(i-1)$ le prédécesseur de l'opération i dans le même job

$(i+1)$ le successeur de l'opération i dans le même job

H Un nombre entier positif suffisamment grand

3.2. Variables:

c_i date de fin d'exécution de l'opération machine $i \in O$

$c_i^{'}$ date de fin d'exécution de l'opération transport $i \in T$

t_i date de début de l'opération machine $i \in O$

$t_i^{'}$ date de début de l'opération transport $i \in T$

$$b_{i,j} = \begin{cases} 1 & \text{si l'opération } i \text{ est exécutée avant l'opération machine } j \text{ sur la même machine } m \\ 0 & \text{sinon} \end{cases}$$

$$f_{i,k} = \begin{cases} 1 & \text{si l'opération de transport } i \text{ est affectée au robot k} \\ 0 & \text{sinon} \end{cases}$$

$$d_{i,j} = \begin{cases} 1 & \text{si l'opération de transport } j \text{ est exécutée avant l'opération de transport } i \\ 0 & \text{sinon} \end{cases}$$

$$e_{i,j} = \begin{cases} 1 & \text{si les deux opérations de transport } i \text{ et } j \text{ sont réalisées par le même robot} \\ 0 & \text{sinon} \end{cases}$$

$$g_{i,j}^k = \begin{cases} 1 & \text{si les deux opérations de transport } i \text{ et } j \text{ sont afféctées au même robot } k \\ 0 & \text{sinon} \end{cases}$$

Soit les trois opérations machines successives $O_{j(k-1)}$, O_{jk} et $O_{j(k+1)}$ et les deux opérations transport $\tau_{j,(k-1)}$, $\tau_{j,k}$, la Figure 3-1 donne une illustration des relations existantes entre les différentes variables

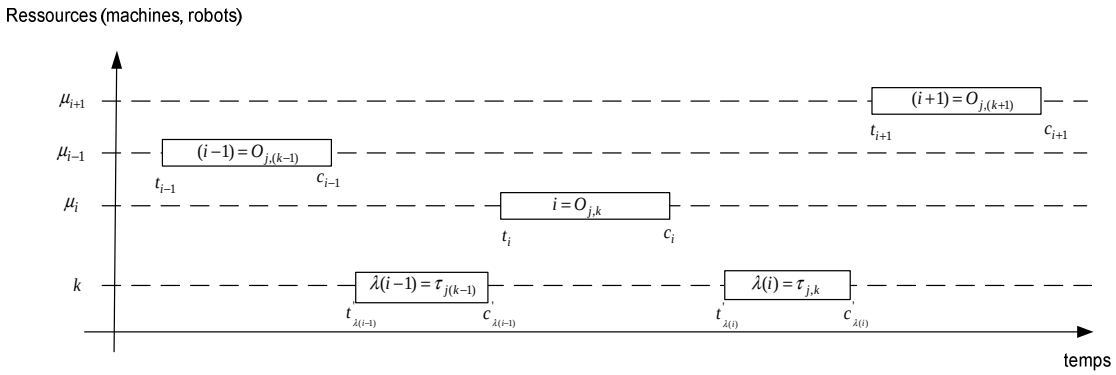


Figure 3-1 : Relations entre les opérations machines et les opérations transport du même job

3.3. Contraintes

On recense quatre ensembles de contraintes : les contraintes de précédence entre les opérations machine et les opérations transport (Contraintes (2)–(3)); les contraintes de disjonction machine (Contraintes (4.1)–(4.3)); les contraintes d'affectation des robots aux opérations de transport (Contraintes (5)); et les contraintes de disjonction transport entre les opérations transport réalisées par le même robot (Contraintes (6.1)–(6.6)).

Date de fin des opérations machines. Ces contraintes garantissent que la date de fin c_i de l'opération machine i soit supérieure ou égale à sa date de début t_i plus sa durée opératoire p_i.

$$c_i = t_i + p_i \qquad \forall i \in O \qquad (1)$$

Contraintes de précédence entre les opérations transport et les opérations machine. Ce groupe de contraintes assure que la date de début de l'opération machine ne peut se faire qu'une fois terminée l'opération transport la précédant immédiatement. La durée de cette dernière dépend du robot k qui lui est assigné.

$$t_i \geq t'_{\lambda,(i-1)} + \sum_{k=1}^{K} (f_{\lambda(i-1),k} \times t^k_{\mu_{i-1},\mu_i}) \qquad \forall i \in O - P \qquad (2)$$

Comme $\sum_{k=1}^{K} f_{i,k} = 1$, alors la contrainte (2) peut être réécrite en $t_i \geq t'_{i-1} + f_{\lambda(i-1),k} \times t^k_{\mu_{i-1},\mu_i}$ pour le robot k tel que $f_{i,k} = 1$.

Contraintes de précédence entre les opérations machine et les opérations transport. Ce groupe de contraintes assure qu'une opération transport ne commence qu'une fois l'opération machine précédente terminée.

$$t'_{\lambda(i)} \geq c_i \qquad \forall i \in O - U \qquad (3)$$

Les contraintes de disjonctions sur les machines. Cette contrainte impose que chaque machine ne traite qu'une opération à la fois. Etant donné deux opérations $i \in O$ et $j \in O$ traitées consécutivement sur une même machine $\mu_i = \mu_j = \mu$, les contraintes sont données par les

équations (4.1-4.3). Les contraintes (4.1) et (4.2) sont arbitrées par la variable binaire $b_{i,j} \in \{0,1\}$ qui rend active une seule contrainte à la fois correspondant à un seul ordre d'exécution. Quand $b_{i,j} = 0$ la contrainte (4.2) est toujours valide car H est suffisamment grand et la contrainte (4.1) devient active. Dans ce cas, la contrainte 4.1, peut être réécrite en $t_i \geq c_j$, ce qui signifie que l'opération machine i ne peut commencer son traitement que si la l'opération machine j est terminé le sien. Par contre si $b_{i,j} = 1$, c'est la contrainte (4.2) qui est active impliquant que l'opération machine j ne commence qu'une fois l'opération machine *i* terminée. La contrainte (4.3) garanti que soit l'opération machine $i \in O$ est lieu en premier soit l'opération machine $j \in O$ est lieu en premier.

$$t_i \geq c_j - H.b_{i,j} \qquad \forall (i,j) \in O \,/\, \mu_i = \mu_j \qquad (4.1)$$

$$t_j \geq c_i - H.(1 - b_{i,j}) \qquad \forall (i,j) \in O \,/\, \mu_i = \mu_j \qquad (4.2)$$

$$b_{i,j} + b_{j,i} = 1 \qquad \forall (i,j) \in O \,/\, \mu_i = \mu_j \qquad (4.3)$$

Contraintes d'affectation des robots aux opérations de transport. Cet ensemble de contraintes assure que pour chaque opération de transport ($\forall i \in T$) un robot lui est affecté.

$$\sum_{k=1}^{K} f_{i,k} = 1 \qquad \forall i \in T \qquad (5)$$

Contraintes de disjonction pour les opérations de transport. Ces contraintes expriment que deux opérations de transport ne peuvent avoir lieu simultanément sur le même robot. Les contraintes (6.1), (6.2) et (6.3) affecte 1 à $e_{i,j}$ si et seulement si les deux opérations transport i et j sont affectées au même robot k.

Dans le cas où deux opérations de transport sont affectées au même robot, nous avons $d_{i,j} = 1$ si l'opération de transport i est réalisée avant l'opération de transport j et $d_{j,i} = 1$ dans le cas contraire. La contrainte 6.6 exprime donc qu'une seule de ces deux situations se produit.

$$g_{i,j}^{k} \geq 1 - (1 - f_{i,k})H - (1 - f_{j,k})H \qquad \forall (i,j) \in T^2, \forall k \in K \qquad (6.1)$$

$$g_{i,j}^{k} \leq f_{i,k} \qquad \forall (i,j) \in T^2, \forall k \in K \qquad (6.2.1)$$

$$g_{i,j}^{k} \leq f_{j,k} \qquad \forall (i,j) \in T^2, \forall k \in K \qquad (6.2.2)$$

$$e_{i,j} = \sum_{k=1}^{K} g_{i,j}^{k} \qquad \forall (i,j) \in T^2 \qquad (6.3)$$

$$t_i' \geq t_j' + \sum_{k=1}^{K} f_{i,k}.\left(t_{\mu_j,\mu_{j+1}}^{k} + v_{\mu_{j+1},\mu_i}^{k}\right) + (e_{i,j} - 1)H + (d_{i,j} - 1).H \qquad \forall (i,j) \in T^2 \qquad (6.4)$$

$$t_j' \geq t_i' + \sum_{k=1}^{K} f_{ik}.\left(t_{\mu_i\mu_{i+1}}^{k} + v_{\mu_{i+1}\mu_j}^{k}\right) + (e_{ij} - 1)H - d_{ij}.H \qquad \forall (i,j) \in T^2 \qquad (6.5)$$

$$d_{i,j} + d_{j,i} = 1 \qquad \forall (i,j) \in T^2 \qquad (6.6)$$

Les contraintes (6.1), (6.2.1) et (6.2.2) assurent que si $g_{i,j}^k = 0$ les deux opérations i et j ne sont pas affectées au robot k. En effet, si $f_{i,k} = f_{j,k} = 1$ (les deux opérations sont affectées au robot k) les contraintes (6.1), (6.2.1) et (6.2.2) assure que $g_{i,j}^k = 1$ et se réécrivent :

$$g_{i,j}^k \geq 1 \qquad (6.1)$$

$$g_{i,j}^k \leq 1 \qquad (6.2.1)$$

$$g_{i,j}^k \leq 1 \qquad (6.2.2)$$

Si $f_{i,k} = 1$ et $f_{j,k} = 0$ la contrainte (6.1) est trivialement vérifiée tandis que les contraintes (6.2.1) et (6.2.2) se réécrivent $g_{i,j}^k \leq 1$ et $g_{i,j}^k \leq 0$ ce qui implique au final que $g_{i,j}^k = 0$. De manière similaire si $f_{i,k} = 0$ et $f_{j,k} = 1$, on obtient $g_{i,j}^k = 0$. Comme nous avons $\sum_{k=1}^{K} f_{i,k} = 1$ pour chaque opération, au plus une seule variable $g_{i,j}^k = 1$ et au pire aucune. Cela justifie la contrainte $e_{i,j} = \sum_{k=1}^{K} g_{i,j}^k$.

Si $e_{ij} = 0$, les contraintes (6.4) et (6.5) sont toujours valides puisque H est un entier suffisamment grand entier positif. Si $e_{ij} = 1$ (les opérations de transport i et j sont assignées au même robot), les contraintes (6.3) et (6.4) peuvent être réécrites comme suit:

$$t_i' \geq t_j' + \sum_{k=1}^{K} f_{ik} . \left(t_{\mu_j \mu_{j+1}}^k + v_{\mu_{j+1} \mu_i}^k \right) + (d_{ij} - 1).H \qquad \forall (i,j) \in T^2$$

$$t_j' \geq t_i' + \sum_{k=1}^{K} f_{ik} . \left(t_{\mu_i \mu_{i+1}}^k + v_{\mu_{i+1} \mu_j}^k \right) - d_{ij}.H \qquad \forall (i,j) \in T^2$$

Dans ce cas, ces deux opérations doivent être ordonnées, soit l'opération $i \in T$ est traitée avant l'opération $j \in T$ si $d_{ij} = 1$, ou dans l'ordre inverse lorsque $d_{ij} = 0$. La contrainte (6.4) devient active lorsque $d_{ij} = 1$ et assure que l'opération j ne peut commencer qu'une fois le transport à charge de l'opération i terminée, ceci est symbolisé par l'addition de la durée de transport à charge et à vide du robot k $t_j \geq \left(t_i + t_{\mu_i \mu_{i+1}}^k + v_{\mu_{i+1} \mu_j}^k \right)$. En effet le robot k effectue d'abord le transport à charge $t_{\mu_i \mu_{i+1}}^k$ puis se déplace à vide de la machine d'arrivée (μ_{i+1}) de l'opération i vers la machine de départ (μ_j) de l'opération transport j représenté par la variable $v_{\mu_{i+1} \mu_j}^k$. L'ordre des opérations $(i,j) \in T^2$ est inversé si $d_{ij} = 0$ ce qui activera la contrainte (6.5).

3.4. Fonction objectif

Le critère à minimiser est la date de fin de la dernière opération machine "Makespan" donnée par la variable $C_{\max} = \max(c_i), \forall i \in U$. Les contraintes (7) sont ajoutées pour garantir les valeurs prises par la variable $C_{\max}$.

$Min\ C_{\max}$

$$C_{\max} \geq c_i \qquad \forall i \in U\ , \qquad (7)$$

Toutes les variables sont supposées être positives ou nulles. Cette formalisation génère un grand nombre de variables binaires qui rendent difficile la résolution de ce PL. Cependant, les bornes inférieures obtenues en limitant le temps de calcul sont très utiles pour juger de la qualité des méthodes approchées proposées pour la résolution de ce problème dans la suite du chapitre.

3.5. Analyse de la formalisation linéaire

Soit le problème suivant : n jobs, m machines. Supposons que tous les jobs passent sur toutes les machines. On a dons $n \times m$ opérations machines et $(n \times m) - m$ opérations transports. Le Figure 3-1 nous donne une idée générale sur le nombre de contraintes et de variables générées par cette formalisation.

Tableau 3-1 : Nombres de contraintes de la formalisation linéaire

N° de contrainte	**Contraintes**	**Plages de variation des indices**	**Nombre de contraintes**
(1)	$c_i = d_i + p_i$	$\forall i \in O$	$n \cdot m$
(2)	$t_i \geq t'_{\lambda(i-1)} + \sum_{k=1}^{K} (f_{\lambda(i-1),k} \times t^k_{\mu_{i-1},\mu_i})$	$\forall i \in O - P$	$n \cdot m - n$
(3)	$t'_{\lambda(i)} \geq c_i$	$\forall i \in U$	$n \cdot m - n$
(4.1)	$t_i \geq c_j - H.b_{i,j}$	$\forall (i,j) \in O$ / $\mu_i = \mu_j$	$m \cdot C_n^2 = \frac{m \cdot n!}{(n-2)!*2!} = (1/2)(m \cdot n^2 - m \cdot n)$
(4.2)	$t_j \geq c_i - H.(1 - b_{i,j})$	$\forall (i,j) \in O$ / $\mu_i = \mu_j$	$m \cdot C_n^2 = \frac{m \cdot n!}{(n-2)!*2!} = (1/2)(m \cdot n^2 - m \cdot n)$
(4.3)	$b_{i,j} + b_{j,i} = 1$	$\forall (i,j) \in O$ / $\mu_i = \mu_j$	$m \cdot C_n^2 = \frac{m \cdot n!}{(n-2)!*2!} = (1/2)(m \cdot n^2 - m \cdot n)$
(5)	$\sum_{k=1}^{K} f_{i,k} = 1$	$\forall i \in T$	$n \cdot m - n$
(6.1)	$g^k_{i,j} \geq 1 - (1 - f_{i,k}).H - (1 - f_{j,k}).H$	$\forall (i,j) \in T$, $\forall k \in K$	$k \cdot ((n \cdot m - n)^2 - (n \cdot m - n))$

(6.2.1)	$g_{i,j}^{k} \le f_{i,k}$	$\forall (i,j) \in T$, $\forall k \in K$	$k \cdot \left((n \cdot m - n)^2 - (n \cdot m - n) \right)$
(6.2.1)	$g_{i,j}^{k} \le f_{j,k}$	$\forall (i,j) \in T$, $\forall k \in K$	$k \cdot \left((n \cdot m - n)^2 - (n \cdot m - n) \right)$
(6.3)	$e_{i,j} = \sum_{k=1}^{K} g_{i,j}^{k}$	$\forall (i,j) \in T$	$(n \cdot m - n)^2 - (n \cdot m - n)$
(6.4)	$t_i^{'} \ge t_j^{'} + \sum_{k=1}^{K} f_{ik} \left(t_{\mu_j \mu_{j+1}}^{k} + v_{\mu_{j+1} \mu_i}^{k} \right) + (e_{ij} - 1)H + (d_{ij} - 1)H$	$\forall (i,j) \in T$	$(n \cdot m - n)^2 - (n \cdot m - n)$
(6.5)	$t_j^{'} \ge t_i^{'} + \sum_{k=1}^{K} f_{ik} . \left(t_{\mu_i \mu_{i+1}}^{k} + v_{\mu_{i+1} \mu_j}^{k} \right) + (e_{ij} - 1)H - d_{ij} . H$	$\forall (i,j) \in T$	$(n \cdot m - n)^2 - (n \cdot m - n)$
(6.6)	$d_{ij} + d_{ji} = 1$	$\forall (i,j) \in T$	$(n \cdot m - n)^2 - (n \cdot m - n)$
(7)	$C_{\max} \ge t_i$	$\forall i \in U$	n
Total $(3k+4)(n \cdot m)^2 - (3k+3/2)nm - (13/2+6k)n^2 m + (3k+4)n^2 + (3k+2)n$			

Remarques

Pour les groupes de contraintes (6.1), (6.2.1), (6.2.1), (6.3) si on exclu les contraintes à l'intérieur du même job on trouve la formule suivante : $(n(n-1))((n-1)(m-1)) \times k$.

Pour les groupes de contraintes (6.4), (6.5), (6.6), si on exclu les contraintes à l'intérieur du même job on trouve la formule suivante : $(n(n-1))((n-1)(m-1))$.

Une observation similaire peut être faite pour les variables $d_{i,j}$ et $e_{i,j}$. Il est important de noter qu'on ne génère pas ces variables si *(i,j)* appartiennent aux mêmes jobs. Dans ce cas la précédence est gérée par les contraintes conjonctives. Le nombre de variables générées est donné par le tableau suivant.

Tableau 3-2 : Nombre de variables générées par notre formalisation

Variables	Plages de variation des indices	Nombre de contraintes
c_i , t_i	$\forall i \in O$	$n \times m * 2$
$c_i^{'} , t_i^{'}$	$\forall i \in O - U$	$n(m-1) * 2$
$b_{i,j}$	$\forall (i,j) \in O \ / \ \mu_i = \mu_j$	$C_n^2 = \frac{n!}{(n-2)!*2!} = \frac{n^2 - n}{2}$
$f_{i,k}$	$\forall i \in T$ et $\forall k \in K$	$n(m-1) \times k$
$d_{i,j}$	$\forall (i,j) \in T$	$((n \times m) - n)((n \times m) - n - 1)$
$e_{i,j}$	$\forall (i,j) \in T$	$((n \times m) - n)((n \times m) - n - 1)$
$g_{i,j}^k$	$\forall i \in T$	$((n \times m) - n)((n \times m) - n - 1) \times k$
Total		$2(n \times m) + (2+k)(n(m-1)) + \frac{1}{2}(n^2 - n) +$ $((n \times m) - n)((n \times m) - n - 1)(2 + 2k)$

Exemple sur une instance de la littérature

Pour donner une illustration du nombre de contraintes et de variables que génère notre modélisation on se sert de deux instances de la littérature. Ces instances sont tirées du benchmark d'Hurink et Knust qui est décrit plus loin. La première instance est **P1-T1-D1-d1** de taille **(6x6).** Elle possède 6 jobs qui sont traités par 6 machines où chaque job est composé de 6 opérations machines et passe par toutes les machines. La deuxième instance est **P02-T1-D1d1** de taille **(10x10)** dont chaque job comporte 10 opérations machines et passe par toutes les machines. Le choix de ces instances n'est pas anodin, car elles constituent un cas extrême pour les instances de 36 ou 100 opérations machines.

Le résultat du calcul du nombre de contraintes et le nombre variables que génère le **PL** sont fournis dans les tableaux 3-3, 3-4 et 3-5.

Tableau 3-3 : Calcul de nombre de contraintes générées pour l'instance **P1-T1-D1-d1**

	P1-T1-D1-d1 taille (6x6)				**P02-T1-D1d1 taille (10x10)**			
Numéro de contrainte	**1 robot**	**2 robots**	**3 robots**	**4 robots**	**1 robot**	**2 robots**	**3 robots**	**4 robots**
(1)	36	36	36	36	100	100	100	100
(2)	30	30	30	30	90	90	90	90
(3)	30	30	30	30	90	90	90	90
(4.1)	105	105	105	105	450	450	450	450
(4.2)	105	105	105	105	450	450	450	450
(4.3)	105	105	105	105	450	450	450	450
(5)	30	30	30	30	90	90	90	90
(6.1)	870	1740	2610	3480	8010	16020	24030	32040
(6.2.1)	870	1740	2610	3480	8010	16020	24030	32040
(6.2.1)	870	1740	2610	3480	8010	16020	24030	32040
(6.3)	870	870	870	870	8010	8010	8010	8010
(6.4)	870	870	870	870	8010	8010	8010	8010
(6.5)	870	870	870	870	8010	8010	8010	8010
(6.6)	870	870	870	870	8010	8010	8010	8010
(7)	6	6	6	6	10	10	10	10
TOTAL	**6537**	**9147**	**11757**	**14367**	**57800**	**81830**	**105860**	**129890**

Tableau 3-4 : Nombre de variables générées pour l'instance **P1-T1-D1-d1**

Variables	**Nombre de variables P1-T1-D1-d1**				**Nombre de variables T1-P02.dat.D1_d1**			
	1 robot	**2 robots**	**3 robots**	**4 robots**	**1 robot**	**2 robots**	**3 robots**	**4 robots**
c_i , t_i	72	72	72	72	200	200	200	200
$c_i^{'} , t_i^{'}$	60	60	60	60	180	180	180	180
$b_{i,j}$	15	15	15	15	45	45	45	45
$f_{i,k}$	30	60	90	120	90	180	270	360
$d_{i,j}$	870	870	870	870	8010	8010	8010	8010
$e_{i,j}$	870	870	870	870	8010	8010	8010	8010
$g_{i,j}^{k}$	870	1740	2610	3480	8010	16020	24030	32040
Total	**2787**	**3687**	**4587**	**5487**	**24545**	**32645**	**40745**	**48845**

La résolution optimale du modèle linéaire pour l'instance **P1-T1-D1-d1** (tableau 3-5) est obtenue en 4 heures.

Tableau 3-5 : Résultats obtenu après de l'exécution de l'instance **P1-T1-D1-d1** avec un robot

Solutions Optimale	Temps d'exécution	Nombre de contraintes	Nombre de variable
87	4 heures	**6492**	**2787**

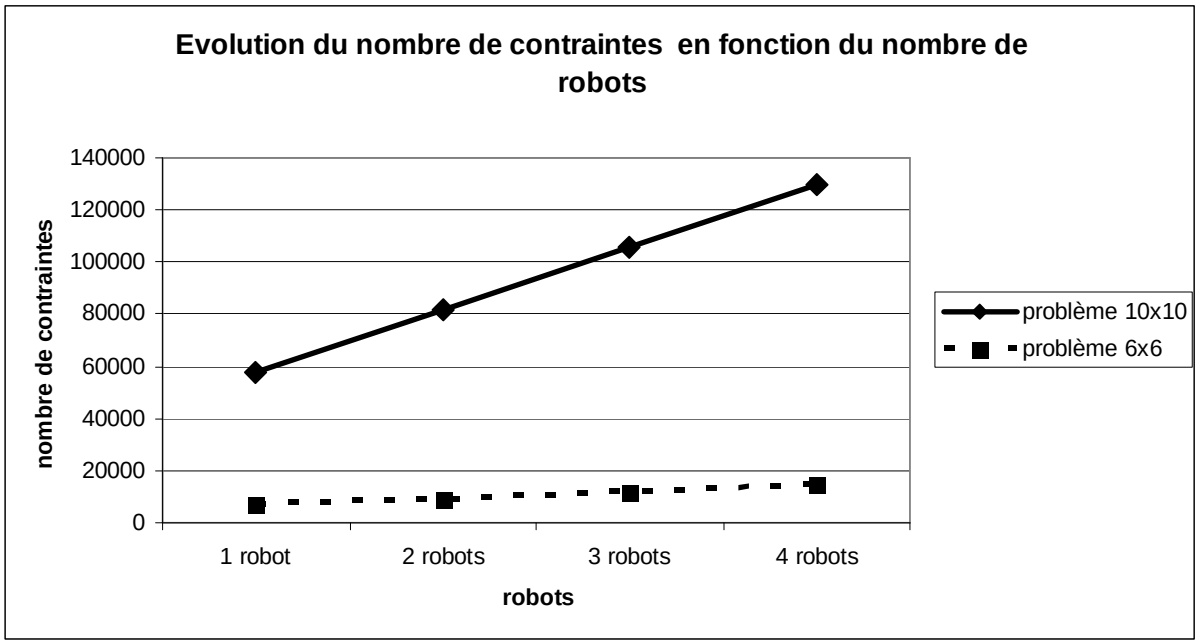


Figure 3-2 : Evolution du nombre de contraintes en fonction du nombre de robots pour l'instance P1-T1-D1-d1 et l'instance T1-P02.dat.D1_d1

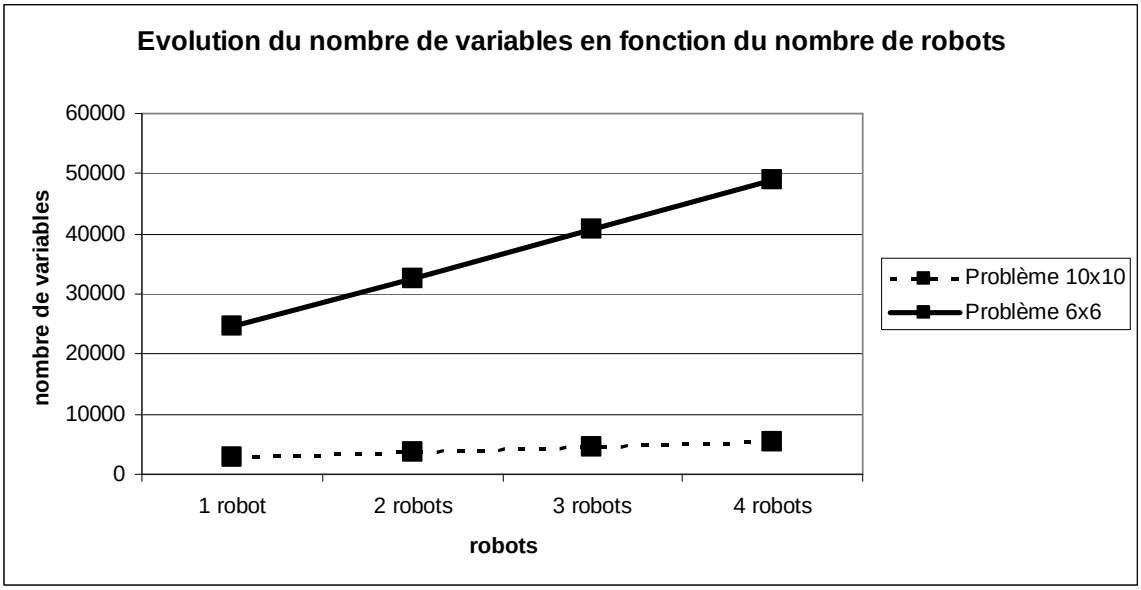


Figure 3-3 : Evolution du nombre de variables en fonction du nombre de robots pour l'instance P1-T1-D1-d1 et l'instance T1-P02.dat.D1_d1

A titre d'exemple nous donnons la solution optimale de l'instance **P1-T1-D1-d1** sur la Figure 3-4.

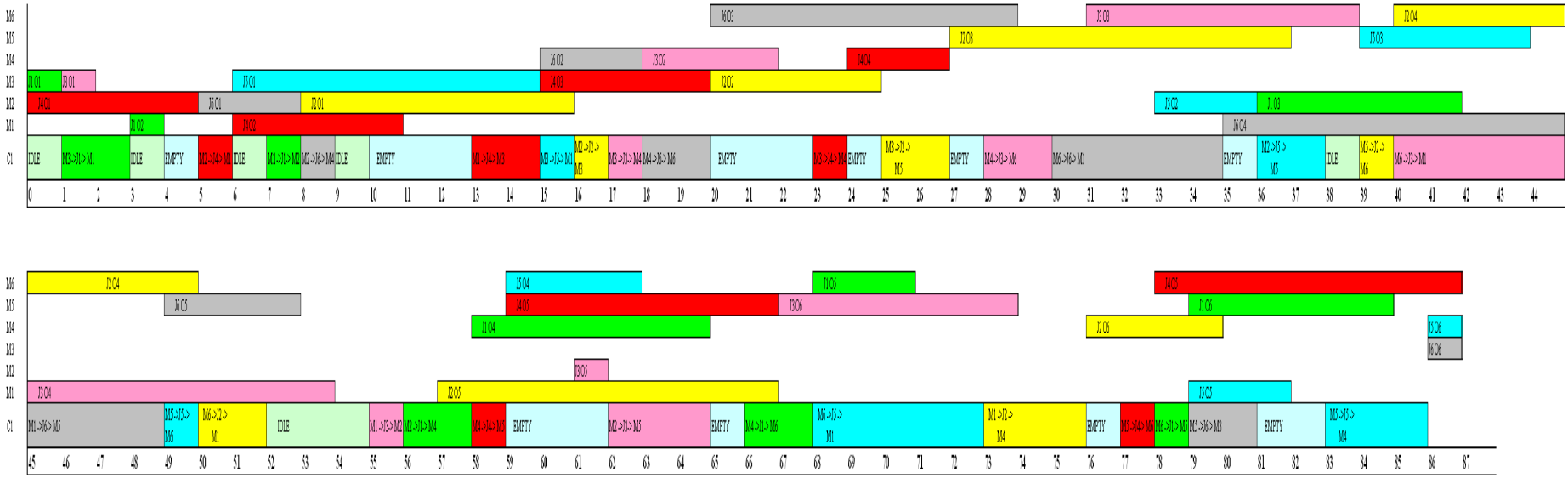


Figure 3-4 : La solution optimale de l'instance **P1-T1-D1-d1**

4. Modélisation du job-shop avec transport

Comme nous venons de voir, cette formalisation génère beaucoup de contraintes et beaucoup de variables binaires, ce qui rend difficile sa résolution dans des temps raisonnables. Pour résoudre ce problème les méthodes exactes ne constituent pas, du moins pour l'instant, le meilleur outil, c'est pourquoi que nous avons orienté nos effort vers les méthodes approchées. Elles s'appuient sur une modélisation sous la forme de graphe disjonctif du problème.

4.1. Modélisation sous forme d'un graphe conjonctif-disjonctif

Notre proposition de modélisation du *JS* avec transport avec plusieurs robots consiste en l'extension du graphe disjonctif proposé dans Roy et Sussmann (1964), Knust (1999), Hurink et Knust (2002)et Hurink et Knust (2005). Cette extension est proposée dans Lacomme et al., (2007) et Lacomme et al., (2010).

Le graphe disjonctif modélisant le job-shop classique présenté dans le chapitre précédant et étendu dans Knust (1999) par l'ajout d'opérations de transport, représentées par des carrés, entre les opérations machines d'un même job. Dans notre cas on utilise un carré vide pour représenter l'opération de transport. Ce carré contiendra le numéro du robot qui effectuera l'opération de transport une fois que le processus d'affectation des robots aux opérations de transport ait été terminé.

Dans cette modélisation le problème du job-shop avec plusieurs robots est représenté par un graphe $G = (V_M \cup V_t, C \cup D_m \cup D_R)$ où :

- V_M est l'ensemble de sommets représentant les opérations machines plus deux nœuds fictifs $\{0,*\}$ tel que $\{0\}$ est le prédécesseur de toutes les opérations du problème, sa date de début est donc (0) et $\{*\}$ est le successeur de toutes les dernières opérations de tous les job et sa date de début sera le makespan ;
- V_t est l'ensemble de sommets représentant les opérations de transport ;
- $(V_M \cup V_t)$ est l'ensemble de sommets du graphe ;
- C représente l'ensemble des arêtes conjonctives à l'intérieur du même job. Les arrêtes sont de type : $O_{i,k} \to t^{R_r}_{\mu_{i,k},\mu_{i,k+1}} \to O_{i,k+1}$ + deux arrêtes reliant les opérations de début et de fin des jobs aux nœuds fictifs $\{0\} et \{*\}$ i.e. : $\forall j \in J$ on ajoute les deux arêtes $0 \to O_{j,1}$ et $O_{j,n_j} \to *$;
- D_M est l'ensemble contenant toutes les arrêtes représentant les disjonctions machine ;
- D_r est l'ensemble contenant toutes les arrêtes représentant les disjonctions robot (transport).

S'aidant de l'exemple suivant (J3), on donnera une illustration du graphe disjonctif conjonctif associé. Soit $\boldsymbol{P}$ un problème de job-shop avec 4 robots : les temps opératoires des opérations machines sont donnés par le Tableau 4-1.

Tableau 4-1 : Temps opératoires

	Op 1	Op 2	Op 3
Job 1	(M3, 1)	(M1, 3)	(M2, 7)
Job 2	(M3, 5)	(M4, 4)	(M1, 1)
Job 3	(M2, 5)	(M1, 5)	(M3, 3)

Tableau 4-2 : Temps de déplacement à charge et à vide des quatre robots

Durée de transport à charge des robots 1 à 4				
	M1	M2	M3	M4
M1	-	2	2	2
M2	2	-	2	2
M3	2	2	-	2
M4	2	2	2	-

Durée de transport à vide des robots 1 à 4				
	M1	M2	M3	M4
M1	-	1	1	1
M2	1	-	1	1
M3	1	1	-	1
M4	1	1	1	-

La Figure 4-1 donne une illustration d'un graphe disjonctif-conjonctif représentant le problème de job-shop avec transport. Comme on peut le remarquer les opérations de transport ne sont affectées à aucun robot.

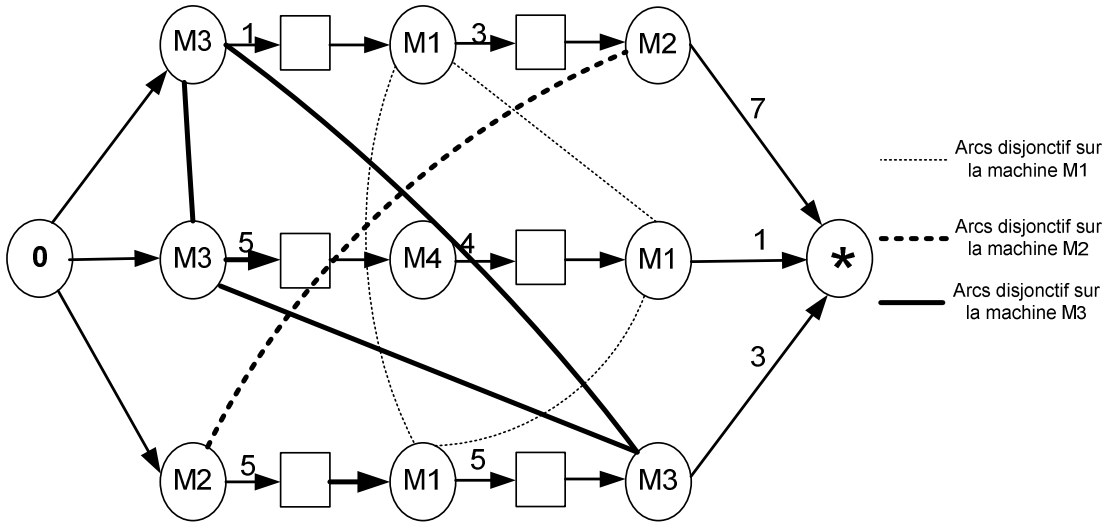


Figure 4-1 : Modélisation du problème *JS3* avec transport sous la forme d'un graphe disjonctif.

Supposons que l'on dispose d'une affection des robots aux opérations de transport sous la forme :

- Robot 1 – opération de transport 1 du job 1
- Robot 2 – opération de transport 2 du job 1
- Robot 1 – opération de transport 1 du job 2
- Robot 3 – opération de transport 2 du job 2
- Robot 4 – opération de transport 1 du job 3

- Robot 1 – opération de transport 2 du job 3

Cela permet d'obtenir un graphe disjonctif non orienté dans lequel on fait apparaître que les opérations de transport affectées au même robot sont en disjonction (cf. Figure 4-2).

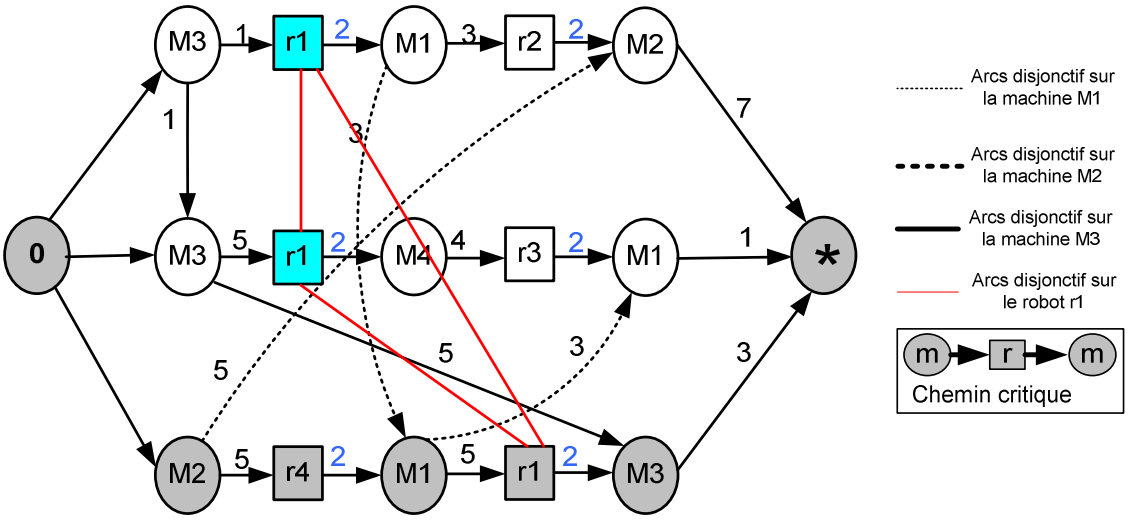


Figure 4-2 : Le graphe disjonctif non orienté du problème *JS3*

Une solution du problème de JS avec transport consiste à choisir une solution au problème de disjonction pour chaque robot. Par exemple, on peut choisir l'ordre des opérations de transport pour le robot 1 de la forme : opération de transport 1 du job 1, opération de transport 1 du job 2 et l'opération de transport 2 du job 3. Le graphe disjonctif est alors complètement orienté et représente une solution du problème dont on peut calculer le makespan par une simple application d'un algorithme de plus long chemin (Figure 4-3).

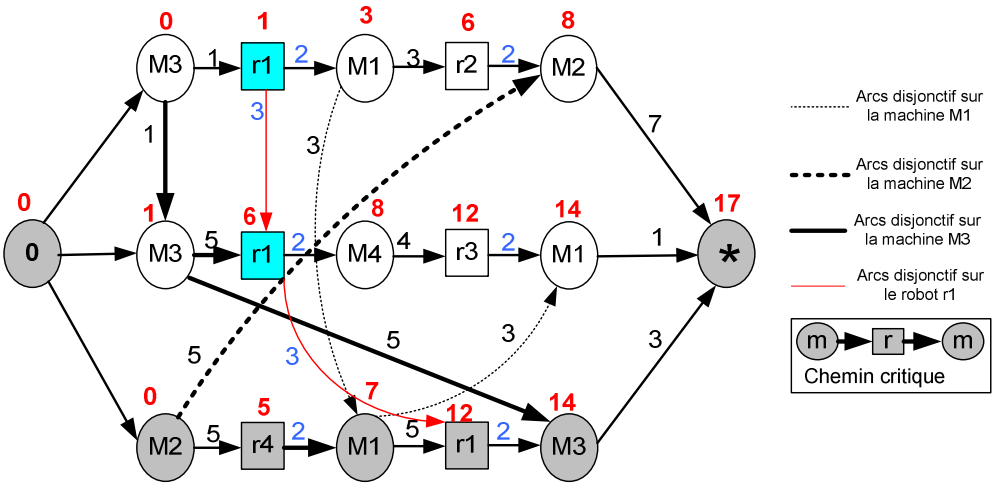


Figure 4-3 : Graphe conjonctif résultat de l'orientation du graphe disjonctif du problème *JS3*

Ce graphe constitue la première étape pour une résolution utilisant des méthodes approchées, il offre un moyen visuel représentant les différentes contraintes du problème. Basés sur le graphe ci-dessus, nous proposons d'utiliser le principe de vecteur à valeurs dupliquées proposé par Bierwirth (1995). Il s'agit d'une suite ordonnée de numéro de jobs. La $j^{ème}$ occurrence

du nombre *i* dans le vecteur par répétition désigne la $j^{ème}$ opération du job *i*. L'avantage de cette représentation est de ne représenter que des ordres valides d'opérations (i.e. des ordres compatibles avec les contraintes de précédence). Pour le job-shop ces vecteurs par répétition permettent d'obtenir uniquement des graphes acycliques.

4.2. Representation R1 du problème du job-shop avec transport

Plusieurs représentations existent pour les problèmes d'ordonnancement. Ces dernières constituent des codages adoptés pour représenter des solutions d'un problème d'ordonnancement. On a choisi de représenter une solution du problème du job-shop avec transport avec plusieurs robots par l'intermédiaire de plusieurs vecteurs. Dans la première représentation on se sert de 3 vecteurs, le premier pour représenter un ordre d'exécution pour les opérations machines, le deuxième et le troisième vecteur pour représenter un ordonnancement des opérations transport. Ces vecteurs sont appelés des sélections. On a une sélection machine composée d'un vecteur SM et une sélection transport composée de deux vecteur $SR = DT \cup AR$ où AR représente les robots affectés aux opérations de transport et DR représente l'ordre choisi pour la réalisation de ses opérations. La deuxième représentation décrite plus loin utilise deux vecteurs, le premier pour représenter un ordre de toutes les opérations du problème et le second pour indiquer l'affectation des robots aux opérations transport.

Dans cette représentation nous utilisons trois vecteurs par répétition notés DM, DR et AR, à l'image du principe de vecteur à valeurs dupliquées utilisé par Bierwith pour le ***JS.*** Ces vecteurs sont :

- DM constitue une sélection machine. Il indique un ordre de passage des jobs sur les différentes machines. C'est un vecteur par répétition, contenant autant d'éléments que d'opérations machines sur l'ensemble du problème. Chaque job j est représenté par un numéro apparaissant n_j fois, où n_j est le nombre d'opérations machine du job j.
- DR qui est une sélection transport. Il représente un ordre d'exécution des opérations transport. Chaque élément de DR est un numéro de job j qui apparaît $(n_j - 1)$ fois, où $(n_j - 1)$ est le nombre d'opération transport du job j.
- AR représente l'affectation des robots aux opérations de transport. Ce vecteur, de taille égale à celle du vecteur DR, contient un numéro de robot affecté à l'opération transport figurant à la même position dans DR.

Les deux vecteurs DR et AR sont utilisés conjointement pour définir précisément les opérations de transport affectées à chaque robot et l'ordre dans lequel elles vont être exécutées. Ainsi, les trois vecteurs DM, DR et AR permettent de décrire un ordre d'opérations, cet ordre est utilisé pour orienter le graphe disjonctif. Pour l'exemple *JS3* donné par le Tableau 4-1 et le Tableau 4-2, la Figure 4-3 et la

Figure 4-4 donnent une illustration de cette représentation.

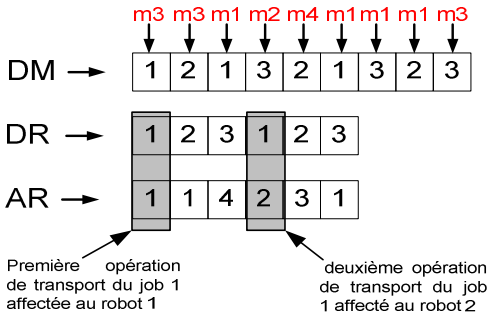


Figure 4-4 : Représentation par vecteur *DM* , *DR* et *AR* du graphe de la Figure 4-3

Le principal inconvénient de cette représentation est l'existence des cycles. En effet la Figure 4-5 et la Figure 4-6 montrent un cycle absorbant constitué par la deuxième opération machine du job 1 s'exécutant sur la machine 1 ; la troisième opération machine du job 2 s'exécutant sur la machine 1 ; et les deuxièmes opérations de transport des jobs 1 et 2 qui sont réalisées par le robot 2. Ce cycle $(o_{1,2} \to t_{1,2} \to t_{2,2} \to o_{2,3} \to o_{1,2})$ rend impossible l'évaluation du graphe ce qui est symbolisé par le l'étiquette $+\infty$ sur les nœuds appartenant au cycle. Cette situation apparaît suite à une mauvaise coordination lors de la construction du vecteur *DM* d'une part et des vecteurs *DR* et *AR* d'autre part. Il est toutefois possible de définir des heuristiques, permettant d'obtenir un graphe conjonctif-disjonctif sans cycle absorbant. Ce point est abordé dans le paragraphe 4.3.

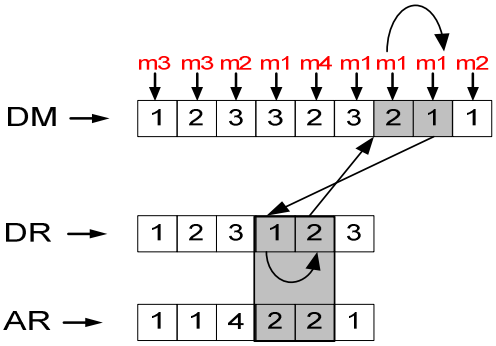


Figure 4-5 : Représentation générant un cycle absorbant

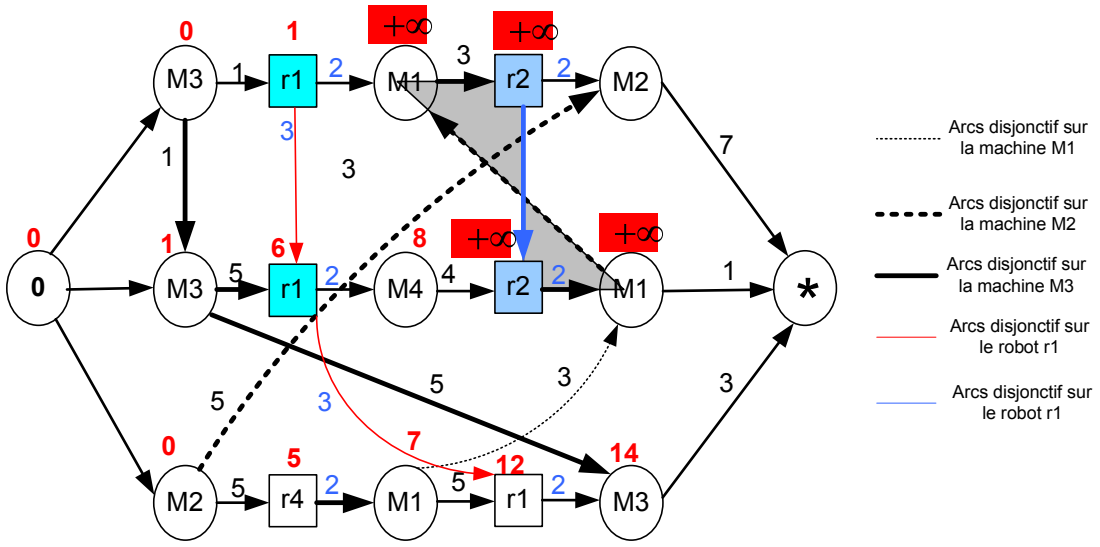


Figure 4-6 : Graphe associé à la représentation de la Figure 4-5

4.3. Heuristiques de construction de solution

Pour une résolution approchée du problème, un effort important a été réalisé pour le modéliser d'abord par le graphe disjonctif puis par des vecteurs avec répétitions. Nous proposons deux heuristiques dédiées au problème du job-shop avec transport comportant plusieurs robots. Ces heuristiques sont basées sur l'algorithme de Giffler et Thompson (1960). Le but de celles présentées dans cette partie est de permettre la construction de solutions sans cycles. S'aidant de l'exemple 2-1, on présente deux heuristiques de construction de solutions partielles ou complètes. La première appelée ***H1***, part de l'idée de générer les vecteurs *DR* et *AR* à partir d'un vecteur *DM* déjà donné. Le vecteur *DM* peut être fixé ou construit aléatoirement. La deuxième se charge de la construction des trois vecteurs à la fois et permet ainsi de générer une seule solution complète, elle est appelée ***H2***. Ces heuristiques constituent les briques de base de l'algorithme mémétique présenté dans la section suivante.

4.3.1. Heuristique H1

Dans ce paragraphe nous proposons une heuristique de construction basée sur des règles de choix. Pour un vecteur *DM* fixé, le challenge est de trouver les vecteurs *DR* et *AR* qui ne génèrent pas de cycle et fournissent une bonne solution. Pour y arriver, des règles de choix sont introduites lors de la construction des deux vecteurs *DR* et *AR* .

A chaque itération de l'heuristique ***H1***, deux opérations sont sélectionnées, à savoir une opération machine et une opération transport. La première est lue dans le vecteur *DM* alors que la deuxième opération est l'opération transport du même job succédant à l'opération lue dans *DM* , et ceci en lui affectant un robot selon une règle bien déterminée.

Les règles de choix du robot sont de la forme suivante :

- Choisir le robot dont la date de disponibilité est la plus proche. Cette date est en effet la date d'arrivée du robot vide à la machine de départ du déplacement à charge.
- La date minimale de fin de transport, qui inclue la date de disponibilité plus la durée du déplacement à vide vers la machine de départ du déplacement plus la durée de transport à charge.

- Choisir le robot le plus rapide, i.e. dont la vitesse est la plus élevée.
- Choisir le robot le plus économique si toutefois une fonction économique est introduite.
- Choisir le robot le moins utilisé pour équilibrer la charge des robots.

Pour la sélection du robot qui effectuera l'opération de transport en cours, nous avons opté pour le choix du robot qui fournit la date minimale de fin au plus tôt de l'opération de transport, ce qui implique d'explorer tous les robots lors de chaque affectation.

Le fonctionnement générale de l'heuristique ***H1*** est donné par le pseudo code suivant :

Algorithme 4-1 : Pseudo code de l'heuristique ***H1***

```
Nom de procédure generer_DR_AR(DM,DR,AR)
Donnée
  DM : sélection machine
Sortie
  DR : sélection transport
  AR : affectation robot
Début
  U := O                                  // ensemble d'opérations non encore ordonnées
  S := ∅                                  // ensemble d'opérations déjà ordonnées
  Tant que U ≠ ∅ faire
       E := Elire(U,S)                    // ensemble d'opérations éligibles
       D := Extension de E                // exploration des robots possibles
       i := Règle(D,F)                    // Choisir une opération i de l'ensemble D en
                                                  fonction d'une règle F
       S := Update(S,i)                   // ajout de l'opération i à S
       U := U − {i}                       // soustraire l'opération i du U
  Fin Tant que
Fin
```

Dans l'algorithme ci-dessus, on complète un ordonnancement partiel itération après itération. D'abord, les ensembles U (ensemble des opérations non ordonnancées) et S (ordonnancement partiel) sont initialisés de manière à représenter l'ordonnancement vide initial. L'ordonnancement est alors construit de manière itérative par la boucle principale. Chaque itération consiste à ordonnancer une nouvelle opération choisie par une règle de priorité. Les procédures *Elire*, *Extension*, *Règle* et *Update* sont appelées, mais leur contenu n'est pas explicitement défini par le pseudo-algorithme car leur contenu dépend du problème.

L'itération commence par construire l'ensemble E des opérations éligibles. Une opération est éligible pour l'ordonnancement partiel S si c'est une opération non ordonnancée ($S \subset E$) et qu'il est possible de commencer cette opération sans violer aucune contrainte. L'ensemble des opérations éligibles S est donc calculé à partir des ensembles U et S. La procédure *Elire* a donc deux paramètres U et S, elle retourne l'ensemble des opérations éligibles. Il est important de noter que les opérations de transport dans E sont des opérations non affectées aux robots.

La seconde étape consiste à construire avec la procédure *Extension* l'ensemble D qui contient toutes les opérations machines de E et toutes les opérations de transport de E affectées à chaque robot. Toutes les affectations robots sont explorées. Les dates de début au plus tôt des opérations machines et transport sont évaluées.

Ensuite, la procédure *Règle* retourne une opération i choisie parmi les opérations de D. L'opération est choisie en fonction de la règle de priorité, cette opération sera ajoutée dans l'ordonnancement.

L'insertion de l'opération *i* dans l'ordonnancement partiel est réalisée par la procédure Update. Cette procédure ajoute l'opération dans l'ordonnancement et met éventuellement à jour les dates de début des autres opérations de l'ordonnancement. La procédure Update renvoie un ordonnancement partiel qui devient l'ordonnancement partiel courant S.

Finalement, on retire l'opération *i* de l'ensemble des opérations à ordonnancer U .

S'appuyant sur l'exemple *JS3* on donne une illustration du fonctionnement de l'heuristique ***H1***, étape par étape et à chaque fois on montre les changements apportés sur le graphe disjonctif associé.

Soit la Figure 4-7 dans laquelle le vecteur *DM* est fixé et *DR* et *AR* sont vides et le graphe disjonctif associé est à droite des vecteurs. La lecture du vecteur *DM* permet d'orienter les arcs disjonctifs machines et les arcs disjonctifs transports sont donc à construire au fur et à mesure de l'avancement de l'heuristique ***H1.***

Considérons l'exemple de la figure Figure 4-7 sur lequel nous appliquons l'heuristique de construction avec comme :

- règle de choix de l'opération transport l'opération dont la date de fin au plus tôt est la plus petite ;
- règle de choix du robot, le robot disponible au plut tôt.

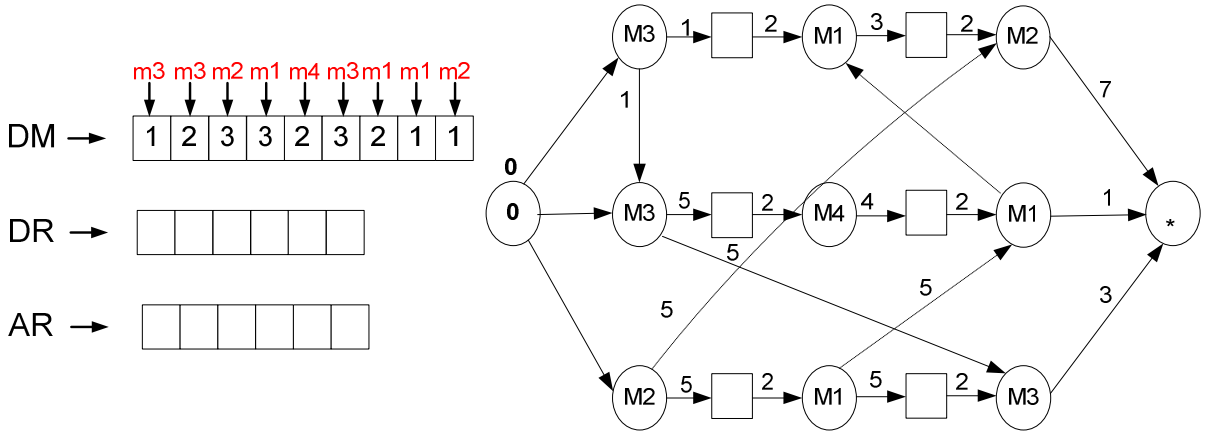


Figure 4-7 : Le graphe associé au vecteur *DM* (*DR* et *AR* vides)

L'ensemble des opérations éligibles est $E = (O_{11}, O_{13})$, i.e. E contient les premières opérations machine de chaque job. La date de fin de l'opération O_{11} est 1 et la date de fin de O_{13} est 5. Ainsi $s = O_{11}$ (Figure 4-8).

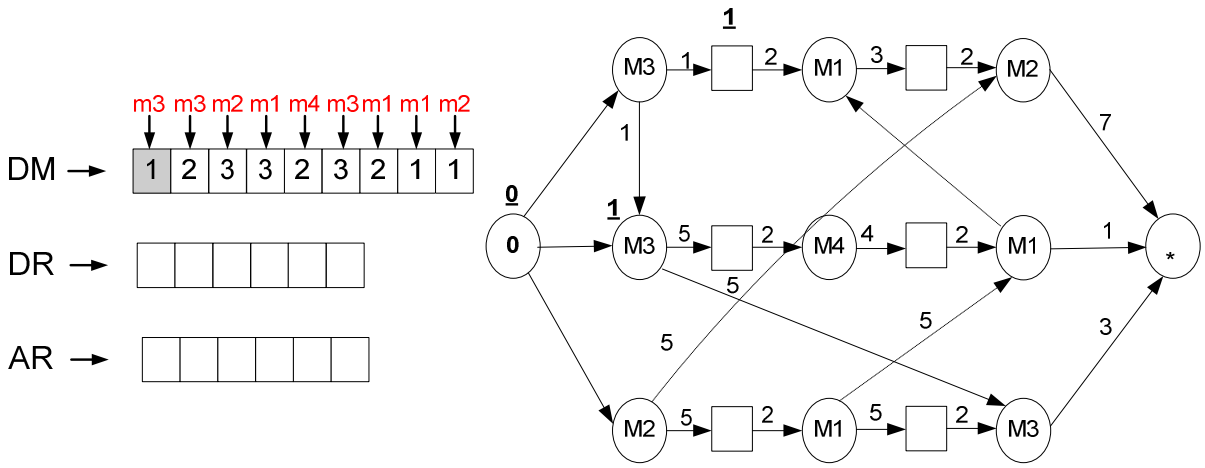


Figure 4-8 : Etape 1 de la lecture dans *DM*

L'ensemble des opérations éligibles est $E=(O_{12},O_{13},\tau_{11})$, i.e. E contient les premières opérations machine du job 2 et du job 3 ainsi que la première opération transport du job 1. A la date 1, les quatre robots sont disponibles et on envisage l'affection de l'opération de transport aux quatre robots : $D=\left(O_{12},O_{13},\tau_{11}^1,\tau_{11}^2,\tau_{11}^3,\tau_{11}^4\right)$. Les quatre opérations $\tau_{11}^1,\tau_{11}^2,\tau_{11}^3,\tau_{11}^4$ ont la même date de fin au plus tôt c'est-à-dire 3 (les quatre robots sont identiques) alors que O_{12} a une date de fin au plus tôt égale à 6 et O_{13} a une date de fin au plus tôt égale à 5. La ligne i := Regle(D,F) sélectionne l'opération ayant la date de fin au plus tôt la plus petite par exemple τ_{11}^1 (Figure 4-9).

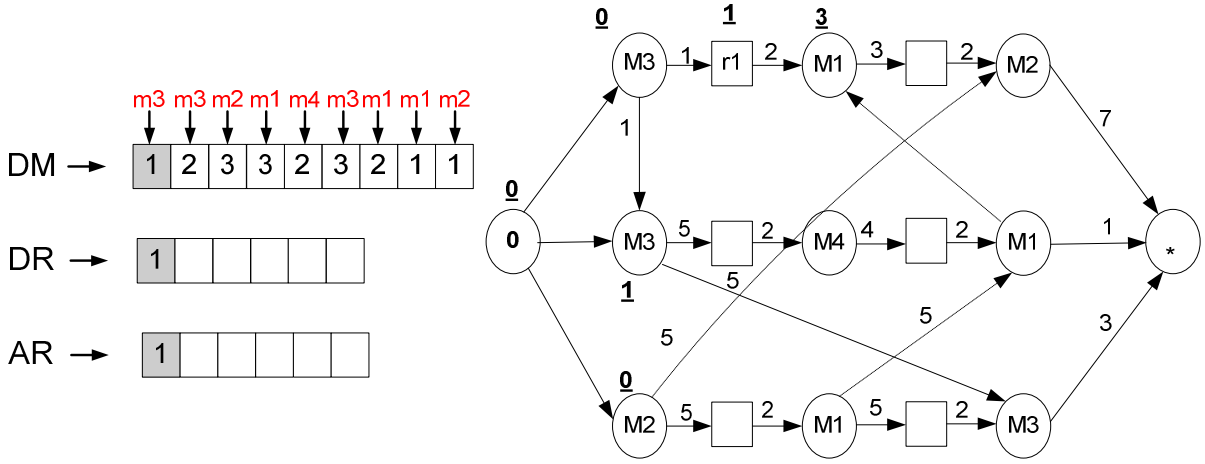


Figure 4-9 : Etape 2 de la lecture dans *DM*

L'ensemble des opérations éligibles est $E=(O_{12},O_{13})$, i.e. E contient les premières opérations machine du job 2 et du job 3. Nous avons $D=(O_{12},O_{13})$: O_{12} a une date de fin au plus tôt égale à 6 et O_{13} a une date de fin au plus tôt égale à 5. La ligne i := Regle(D,F) sélectionne l'opération ayant la date de fin au plus tôt la plus petite donc ic O_{13} (Figure 4-10).

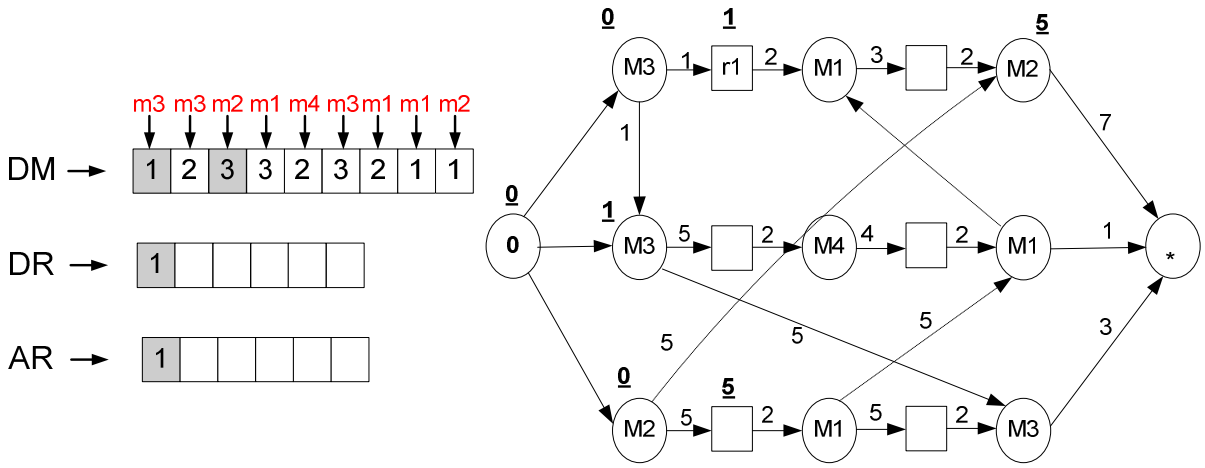


Figure 4-10 : Etape 3 de la lecture dans *DM*

L'ensemble des opérations éligibles est $E=(O_{12},\tau_{31})$, i.e. E contient la première opération machine du job 2 et la première opération de transport du job 3. Nous avons $D=\left(O_{12},\tau_{31}^1,\tau_{31}^2,\tau_{31}^3,\tau_{31}^4\right)$: O_{12} a une date de fin au plus tôt égale à 6 et $\tau_{31}^1,\tau_{31}^2,\tau_{31}^3,\tau_{31}^4$ à une date de fin au plus tôt égale à 7. La ligne i := Regle(E,F) sélectionne l'opération ayant la date de fin au plus tôt la plus petite qui est ici O_{12} (Figure 4-11).

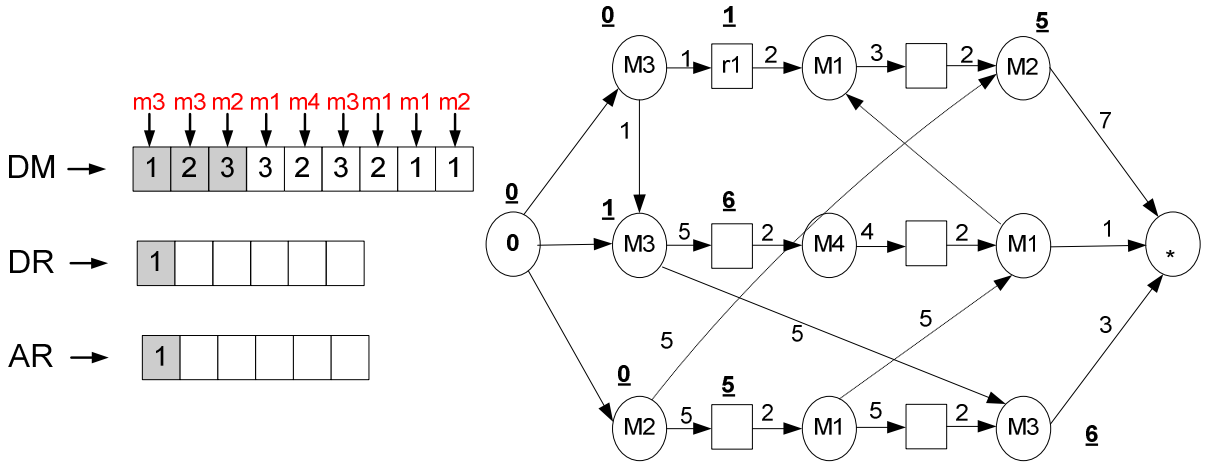


Figure 4-11 : Etape 4 de la lecture dans *DM*

L'ensemble des opérations éligibles est $E=(\tau_{21},\tau_{31})$, i.e. la première opération de transport du job 2 et la première opération transport du job 3. Nous avons $D=\left(\tau_{21}^1,\tau_{21}^2,\tau_{21}^3,\tau_{21}^4,\tau_{31}^1,\tau_{31}^2,\tau_{31}^3,\tau_{31}^4\right)$ avec $\tau_{31}^1,\tau_{31}^2,\tau_{31}^3,\tau_{31}^4$ qui ont une date de fin au plus tôt égale à 7 et $\tau_{21}^1,\tau_{21}^2,\tau_{21}^3,\tau_{21}^4$ qui ont date de fin au plus tôt égale à 8. La ligne i := Regle(E,F) sélectionne l'opération ayant la date de fin au plus tôt la plus petite ici τ_{31}^1 (Figure 4-12).

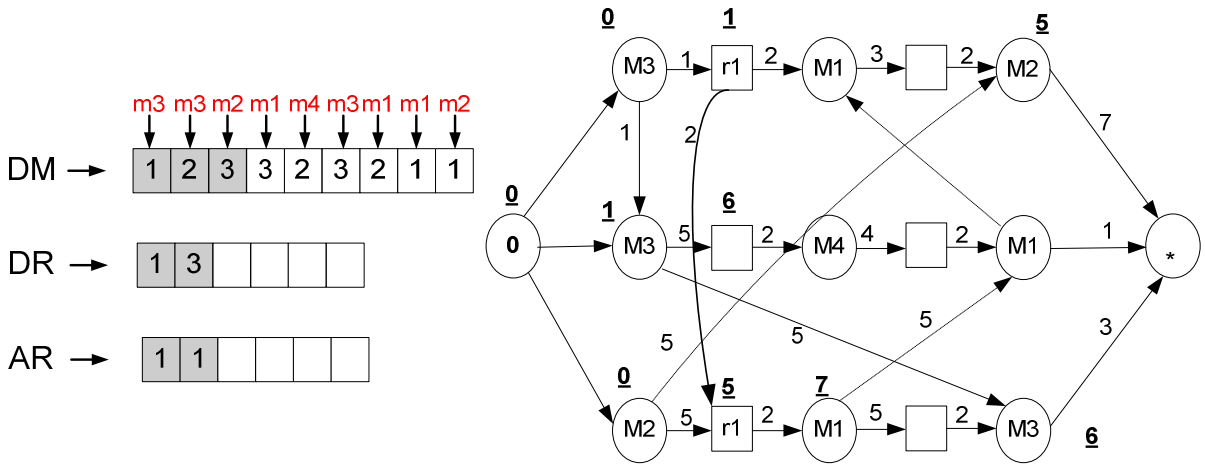


Figure 4-12 : Etape 5 de la lecture dans *DM*

L'ensemble des opérations éligibles est $E=(\tau_{21},O_{32})$, i.e. la première opération transport du job 2 et la deuxième opération de transport du job 3. Nous avons $D=(\tau_{21}^1,\tau_{21}^2,\tau_{21}^3,\tau_{21}^4,0_{32})$ avec $\tau_{21}^2,\tau_{21}^3,\tau_{21}^4$ qui ont une date de fin au plus tôt égale à 8, τ_{21}^1 a une date de fin au plus tôt égale à 10 (la date de fin du déplacement à charge du robot r1 est 7 auquel on doit ajouter 1 unité de temps pour le déplacement à vide de M1 vers M3 auquel on doit ajouter 2 unités de temps pour le déplacement à charge de M3 vers M4) et O_{23} qui a une date de fin au plus tôt égale à 13. La ligne i:= Regle(E') sélectionne l'opération ayant la date de fin au plus tôt la plus petite ici τ_{31}^2 (Figure 4-13).

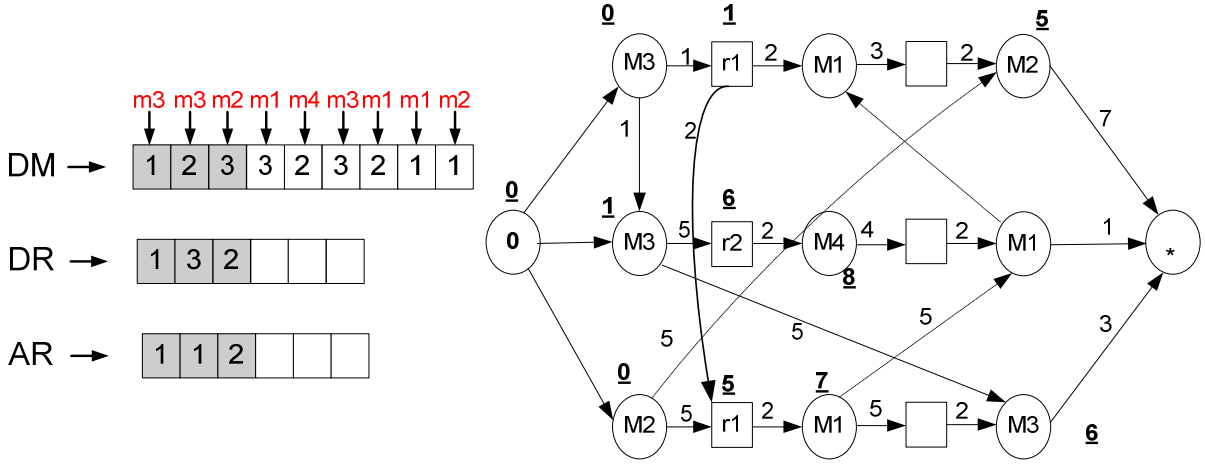


Figure 4-13 : Etape 6 de la lecture dans *DM*

A la fin de l'exécution de l'heuristique on obtient une solution initiale du problème *JS3*. Cette heuristique fonctionne uniquement dans le cas où le vecteur *DM* est fixé.

4.3.2. Heuristique H2

L'heuristique ***H2*** est une heuristique de construction, elle fournit une solution complète à un problème de job-shop avec plusieurs robots. Le principe de construction de l'ordonnancement est similaire à l'heuristique ***H1***, on complète un ordonnancement partiel itération après itération. Contrairement à l'heuristique ***H1*** qui part d'un ordre des opérations

machines fixé, ***H2*** ordonnance deux opérations à la fois, une opération machine et l'opération de transport correspondante qui la succède. A chaque itération de l'heuristique se fait un choix d'une opération machine de l'ensemble U (ensemble des opérations non ordonnancées) ne possédant pas de prédécesseurs dans U. Comme chaque opération de transport possède toujours un prédécesseur machine, le choix d'une opération dans U concerne uniquement les opérations machines. Une fois que le choix de l'opération machine ait été effectué, l'opération de transport suivante est immédiatement traitée en lui affectant un robot selon une règle de choix bien déterminée, ensuite on soustrait les deux opérations de l'ensemble U pour les mettre dans l'ensemble S contenant les opérations ordonnées. L'algorithme 4-2 donne le pseudo code du fonctionnement général de cette heuristique.

Chaque itération consiste à ordonnancer une nouvelle opération choisie par une règle de priorité. Les procédures Tail, Elire, Extension, Update, Règle et Update sont appelées.

La sélection va se faire comme suit : une opération machine qui n'a pas de prédécesseur dans U est choisie à chaque itération. Si plusieurs choix existent, on fait intervenir le calcul de la queue Tail qui trie les jobs dans un ordre donné.

Le tri peut se faire soit arbitrairement par un ensemble de paramètres soit par une règle, par exemple, si plusieurs choix existent on choisit l'opération ayant la queue la plus grande, sachant qu'on appelle queue d'un job la somme des processing times des opérations non encore ordonnancées.

.

Algorithme 4-2 : Génération des vecteurs DM DR et AR par l'heuristique H2

```
Nom de procédure gener_DM_DR_AR
    Données : donnée du problème (gammes, temps opératoires et temps de transport)
    Sortie
      DM : sélection machine
      DR : sélection transport
      AR : assignement robot
  Début
    U ≔ O                                  // ensemble de toutes les opérations non ordonnées
    S ≔ ∅                                  // ensemble de toutes les opérations ordonnées
    Tant que U ≠ ∅ faire
        Pour I ∈ J faire
          Calculer Tail[i] pour job i      // calculer la priorité du job i
        Fin pour

        Pour i:=1 à n faire
          j := Elire(U, J)                 // élire un job j
          S := Update(S, {o_{j,s_j}})      // mettre à jour l'ensemble S avec l'opération {o_{j,s_j}}
          E := {τ_{j,s_j}}                 // traiter l'opération transport suivante {τ_{j,s_j}}
          D := Extension de E              // exploration des robots possibles
          p := Règle(F, k)                 // choisir le robot k selon la règle F
          S := Update(S, {τ^k_{j,s_j}})    // mettre à jour l'ensemble S avec l'opération {τ^k_{j,s_j}}
          U := U- {o_{j,s_j}}              // mettre à jour l'ensemble U en soustrayant
                                                 l'opération {o_{j,s_j}}
        Fin pour
      Fin Tant que
    Fin
```

S'aidant de l'exemple *JS3* on donnera une illustration du fonctionnement de l'heuristique

H2. Soit la figure suivante dans laquelle les vecteurs *DM* , *DR* et *AR* sont vides :

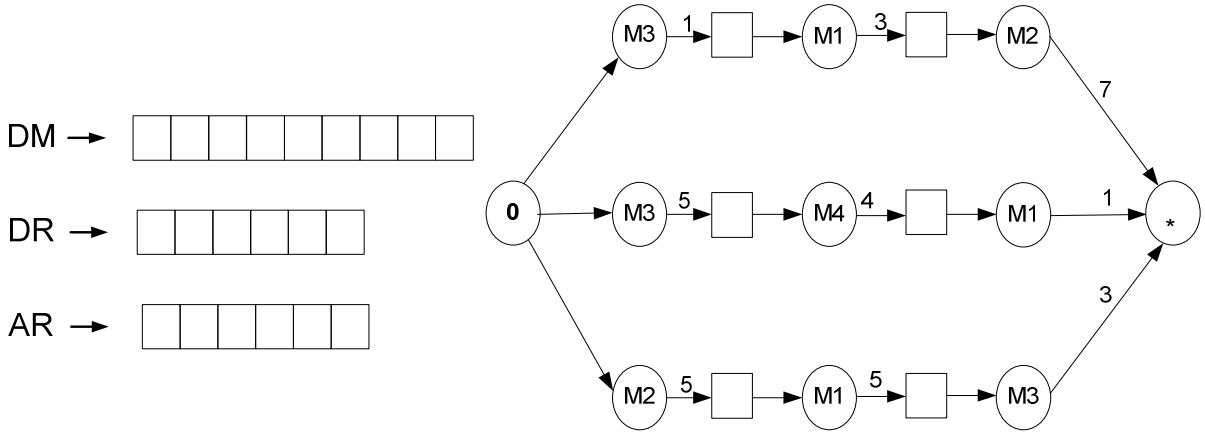


Figure 4-14 : Graphe de départ, les 3 vecteurs DM, DR et AR sont vides

Au départ l'ensemble $U = \{o_{1,1}, o_{1,2}, o_{1,3}, o_{2,1}, o_{2,2}, o_{2,3}, o_{3,1}, o_{3,2}, o_{3,3}\}$ et l'ensemble $S = \Phi$,

Les figures qui suivent vont montrer l'orientation du graphe disjonctif-conjonctif pour le problème de l'exemple *JS3*.

Les seules opérations ordonnançables sont l'opération 1 du job 1, l'opération 1 du job 2 et l'opération 1 du job 3. Nous avons *Tail(1)=11*, *Tail(2)=10* et *Tail(3)=13*. La règle consiste à choisir l'opération appartenant au job ayant donné la queue maximale i.e. l'opération 1 du job 3 (Figure 4-15).

Simultanément on ordonnance l'opération de transport que l'on affecte au robot ayant la date de disponibilité la plus petite (par exemple le robot 1).

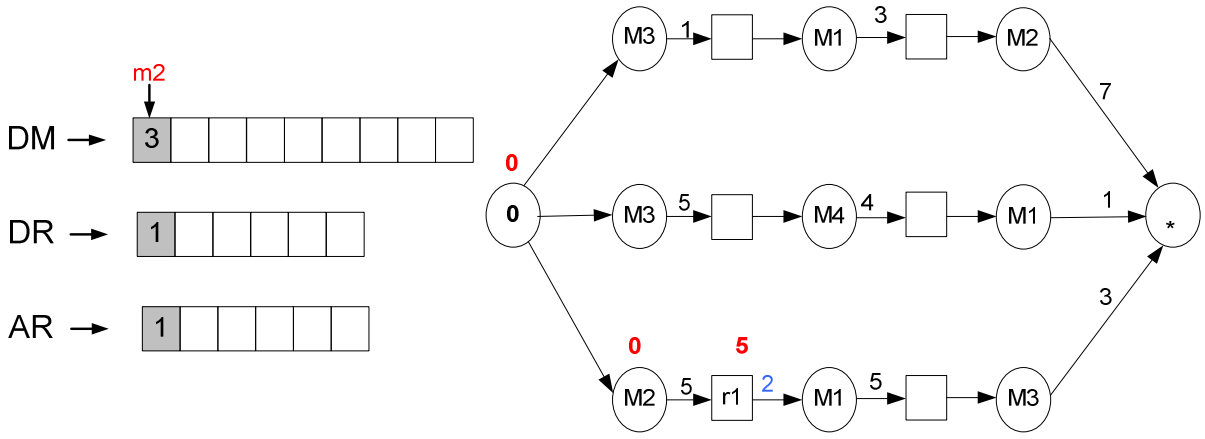


Figure 4-15 : L'affectation de l'opération du job 1

Les opérations ordonnançables sont l'opération 1 du job 1, l'opération 1 du job 2 et l'opération 2 du job 3. Nous avons *Tail(1)=11, Tail(2)=10 et Tail(3)=8*. La règle consiste à choisir l'opération appartenant au job ayant donné la queue maximale i.e. l'opération du job 1. Simultanément on ordonnance l'opération de transport que l'on affecte au robot ayant la date de disponibilité la plus petite (par exemple le robot 2 Figure 4-16).

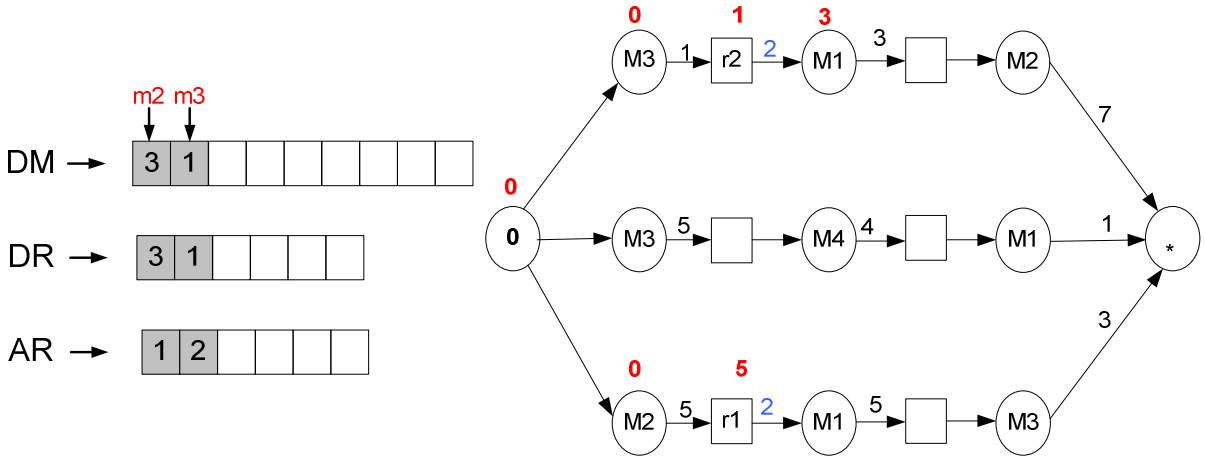


Figure 4-16 : Etape 2 de l'heuristique *H2* exécutée sur l'exemple *JS3*

Les opérations ordonnançables sont l'opération 2 du job 1, l'opération 1 du job 2 et l'opération 2 du job 3. Nous avons *Tail(1)=10*, *Tail(2)=10* et *Tail(3)=8*. La règle consiste à choisir l'opération appartenant au job ayant donné la queue maximale. Par exemple l'opération du job 2. Simultanément on ordonnance l'opération de transport que l'on affecte au robot ayant la date de disponibilité la plus petite (par exemple le robot 3).

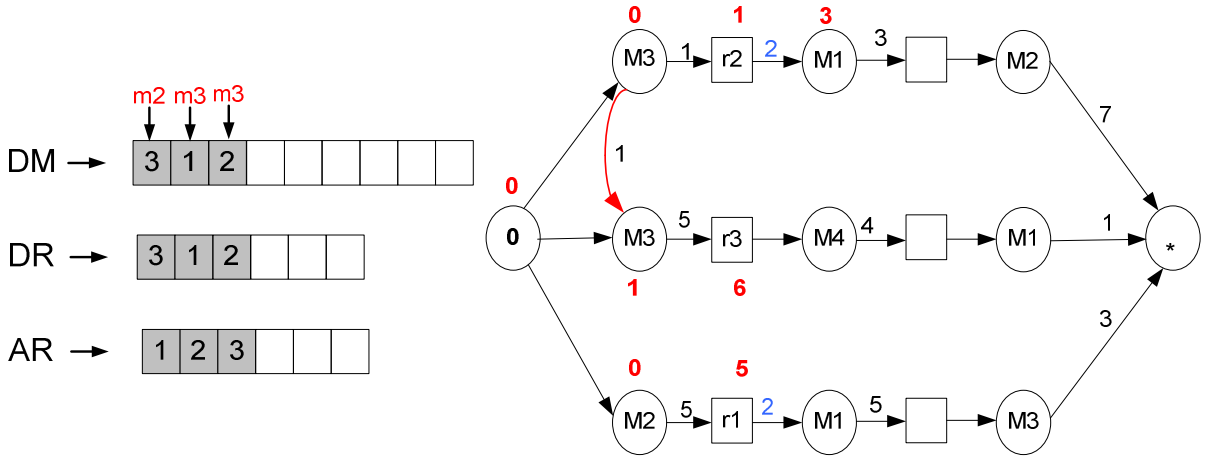


Figure 4-17 : L'affectation de l'opération de transport du job 2 au robot 3.

A la fin de l'exécution de l'heuristique on obtient une solution initiale du problème *JS3*.

4.4. Evaluation des performances des heuristiques H1 et H2

Pour étudier la performance des heuristiques ***H1*** et ***H2***, nous avons élaboré un plan d'expériences. Ainsi, afin d'évaluer la qualité des résultats fournis par les heuristiques nous les avons comparées à ceux donnés par la résolution du programme linéaire.

Rappelons que l'heuristique ***H1*** construit l'ordre des opérations transport pour un ordre des opérations machine fixé (le vecteur *DM* est donné). Nous avons utilisé les solutions obtenues par la résolution du programme linéaire pour déterminer le vecteur *DM* en tant que paramètre d'entrée de ***H1***. L'heuristique génère par la suite les vecteurs *DR* et *AR* pour chaque vecteur DM

fourni. Les résultats obtenus sont donc comparés aux solutions optimales issues du ***PL***.

Le récapitulatif de ces résultats est donné par les tableaux (Tableau 4-3 et

Tableau 4-4), il concerne un benchmark de taille moyenne, composé de 3 problèmes de job-shop comprenant de 16 à 30 opérations machines. La colonne $\mathbf{C_{max}}$***** contient les résultats optimaux obtenus par le ***PL***. La colonne C_{max}***H1*** contient les résultats obtenus par l'exécution de l'heuristique ***H1***, sur les vecteurs *DM* donnés dans la colonne ***DM*,*** la colonne ***Dev%*** représente la déviation en pourcentage du C_{max} fourni par l'heuristique ***H1*** par rapport à la solution optimale fourni par le ***PL***. La dernière ligne du tableau contient la moyenne des déviations calculée pour chaque problème.

Le Tableau 4-3 montre les résultats obtenus. La déviation moyenne est de **12,53%** et peut être jugé plutôt satisfaisant pour une heuristique de construction.

Tableau 4-3 : Test réalisé sur l'heuristique ***H1***

Instances	***DM****	C_{max}*****	C_{max} ***H1***	***Dev%***
LLM1G1 - 1 Robot	1 1 2 4 2 3 1 3 4 2 1 2 4 3 3 4	40	41	2,50%
LLM1G1 - 2 Robots	1 3 4 1 3 2 2 3 1 4 3 4 1 2 4 2	39	44	12,82%
LLM1G1 - 3 Robots	1 3 4 1 3 2 2 3 1 4 3 4 1 2 4 2	39	44	12,82%
LLM1G1 - 4 Robots	1 3 4 1 3 2 2 3 1 4 3 4 1 2 4 2	39	44	12,82%
LLM2G1 - 1 Robot	1 3 4 2 2 3 1 4 4 1 2 3 3 1 4 2	78	107	37,18%
LLM2G1 - 2 Robots	1 3 2 4 2 3 1 4 4 1 2 3 3 1 2 4	50	58	16,00%
LLM2G1 - 3 Robots	1 3 4 2 2 3 1 4 4 1 3 3 2 1 4 2	48	49	2,08%
LLM2G1 - 4 Robots	1 3 4 2 2 3 1 4 4 1 3 3 2 1 4 2	48	48	0,00%
LLM3G1 - 1 Robot 1	1 3 2 4 1 2 3 4 4 1 1 4 3 2 2 3	59	87	47,46%
LLM3G1 - 2 Robots	1 3 4 2 2 1 4 4 1 3 3 2 3 1 4 2	45	48	6,67%
LLM3G1 - 3 Robots	1 3 4 2 1 4 2 3 1 3 2 4 3 1 4 2	44	44	0,00%
LLM3G1 - 4 Robots	1 3 4 2 1 4 2 3 4 4 1 3 2 3 1 2	44	44	0,00%
Dev moyenne				**12,53%**

Pour tester l'heuristique ***H2*** nous avons utilisé la même démarche que celle utilisée pour l'évaluation des performances de l'heuristique ***H1***. Les expérimentations numériques ont été réalisées en utilisant un nombre différents de robots pour les mêmes jeux de données. Nous avons fait comparaison entre les résultats de l'heuristique et les résultats du ***PL***. La dernière ligne des colonnes Dev% donne les déviations moyennes. Ces déviations moyennes sont entre 8.74 et 12.11%, ce qui peut être considéré comme une performance très satisfaisante pour une heuristique de construction de solutions complètes.

Tableau 4-4 : Test réalisé sur l'heuristique **H2**

	1 robot			2 robots			3 robots			4 robots		
Instances	**C_{max}***	**C_{max} H2**	**Dev%**	**C_{max}***	**C_{max} H2**	**Dev%**	**C_{max}***	**C_{max} H2**	**Dev%**	**C_{max}***	**C_{max} H2**	**Dev%**
LLM1-G1	40	43	7,50%	39	41	5,13%	39	41	5,13%	39	41	5,13%
LLM2-G1	78	106	35,90%	50	59	18,00%	48	55	14,58%	48	55	14,58%
LLM3-G1	59	84	42,37%	45	52	15,56%	44	51	15,91%	44	51	15,91%
Dev moyenne			**28,59%**			**12,89%**			**11,87%**			**11,87%**

Dans cette partie nous avons montré comment construire une solution complète à un problème de job-shop avec plusieurs robots. Le point traité dans la partie suivante est la possibilité d'améliorer une solution donnée, s'appuyant sur des propriétés de voisinage. Nous proposons une recherche locale efficace qui permet d'améliorer une solution.

4.5. Recherche Locale

Cette partie est consacrée à la recherche locale, qui constitue une autre brique de base des méta-heuristiques que nous utilisons par la suite pour résoudre le problème du ***JS*** avec plusieurs robots.

4.5.1. Présentation générale

Cette recherche locale s'appuie sur la notion de blocs introduite par Grabowski et al., (1996), Nowicki et Smutnicki (1996), Hurink et Knust (2002). Ces blocs sont une succession d'opérations de même type réalisées par une même ressource (machine ou robot). Ces blocs se trouvent le long du chemin critique, la recherche de ce chemin est donc nécessaire pour leur l'identification.

En plus des blocs machines existant pour le ***JS*** classique, on trouve des blocs transport constitués par les opérations transports réalisées par le même robot. On peut donc considérer que le chemin critique se compose :

- de « *bloc machine* », un bloc machine étant une succession d'opérations machine ayant lieu sur la même machine et appartenant à des jobs différents ;
- de « *bloc transport* », un bloc transport étant une succession d'opérations transport affectées au même robot et appartenant à des jobs différents ;
- de « *bloc conjonctif* », un bloc conjonctif étant une succession d'opérations machines et d'opération transport d'un même job.

Ces trois situations sont représentées sur le graphe de la Figure 4-18.

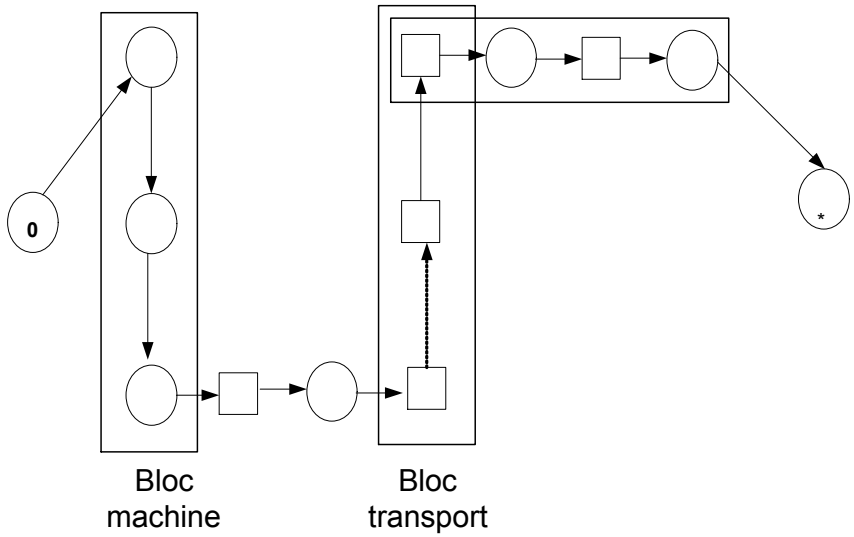


Figure 4-18 : Graphe illustrant les blocs transports et machines

Toute solution voisine comprenant les mêmes arcs que ceux du chemin critique ne peut être que de cout supérieur ou égal au cout de la solution courante. Les modifications à faire sur le graphe doivent concerner des opérations situées à l'extrémité d'un block machine ou d'un bloc transport. Pour un bloc transport, on peut aussi modifier l'affectation d'un robot à une opération transport.

Pour casser le chemin critique on procède par inversion des arcs qui sont aux extrémités des blocs, dans notre cas, on inverse les arcs à l'extrémité la plus à gauche, car on analyse le chemin critique à l'envers partant du nœud $\{*\}$ vers $\{0\}$. Chaque fois qu'on rencontre un bloc on inverse l'arc et on réévalue la nouvelle configuration du graphe. Ces opérations sont effectuées sur les vecteurs DM, DR et AR. Dans ce qui suit, on donne une illustration pour chaque type d'opérations en s'appuyant sur la solution fournie par le graphe disjonctif et sa représentation sous forme de vecteurs illustrée par la même Figure 4-19.

Nous avons choisi ce graphe parce que le chemin critique qu'il contient comporte les deux types de blocs, On remarque sur le chemin critique (en gris) deux blocs formés d'opérations machine et opérations transport : *(i)* le premier de droite à gauche est composés des opérations s'exécutant sur la machine $M1$, *(ii)* le deuxième bloc transport est composé des opérations transports réalisé par le robot $r1$. La Figure 4-19 présente une solution de départ qu'on va améliorer au moyen de la recherche locale.

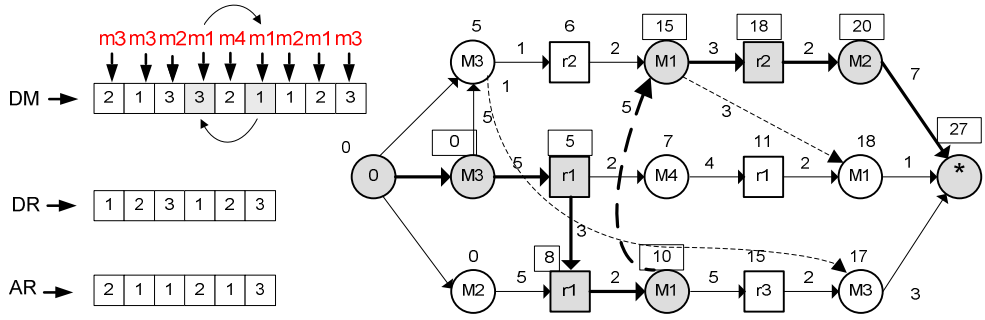


Figure 4-19 : Graphe d'illustration de l'inversion d'un arc d'un bloc machine

Dans l'analyse de la solution donnée par le graphe de la Figure 4-19, on tente de briser le bloc machine formé sur la machine M1 par les opérations $o_{3,2}$ et $o_{1,2}$. Pour casser ce bloc il suffit d'inverser l'arc $(o_{3,2} \rightarrow o_{1,2})$ (le résultat de cette opération est donné par la Figure 4-21. Cette opération nécessite d'inverser l'ordre relatif de ces deux opérations dans le ve cteur DM

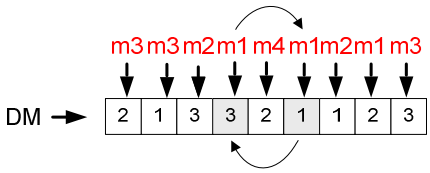


Figure 4-20 : Inversion de l'arc $(o_{3,2} \rightarrow o_{1,2})$ dans le vecteur *DM*

On signale la disparition des arcs $(o_{3,2} \rightarrow o_{1,2})$ et $(o_{1,2} \rightarrow o_{2,3})$, et l'apparition de plusieurs nouveaux arcs $(o_{1,2} \rightarrow o_{3,2})$, $(o_{3,2} \rightarrow o_{2,3})$, $(o_{3,1} \rightarrow o_{1,3})$ et $(\tau_{3,1} \rightarrow \tau_{2,2})$ réalisées par la simple permutation des cases concernées dans le vecteur *DM* (voir Figure 4-21). Après cette opération, la nouvelle solution obtenue est meilleur que la solution précédente puisqu'elle vaut 21 alors que l'ancienne valait 27.

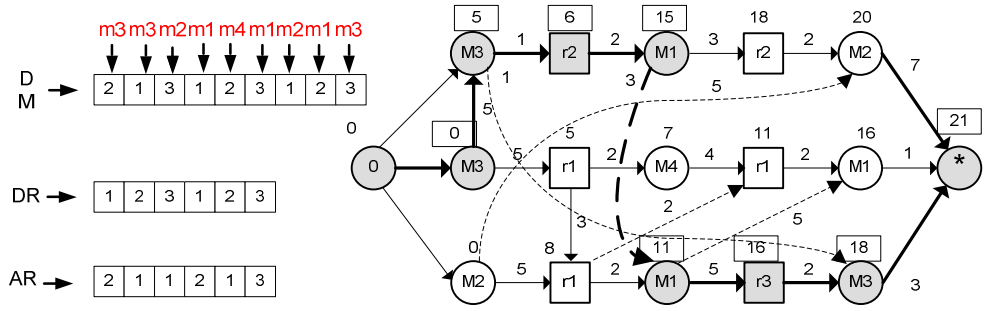


Figure 4-21 : Résultat de l'inversion d'un arc d'un bloc machine

On obtient un nouveau graphe dans lequel figure un arc de l'opération machine $(o_{1,2} \rightarrow o_{3,2})$. Ce graphe représente une solution de coût 21 qui est strictement inférieur au coût de 27. Ce graphe est donc le graphe courant sur lequel la recherche locale va continuer. Ce graphe possède un nouveau chemin critique.

On parcourt le nouveau chemin critique en partant du sommet final * et on recherche un nouveau bloc machine ou un bloc transport. Dans notre cas, le premier, toujours en allant de la droite vers la gauche, est celui de la machine M1.

On est conduit dans un premier temps à inverser l'ordre des opérations $o_{1,2}$ et $o_{3,2}$ ce qui donnerait le graphe initial de coût 27. Cette modification donnant une solution de cout supérieur, le parcours du chemin critique continue jusqu'au bloc machine de la machine M3.

Le processus de la recherche locale continue avec l'inversion de l'arc dans le bloc machine de M3 formé par les opérations $o_{1,1}$ et $o_{2,1}$. On envisage alors d'inverser l'ordre relatif de ces deux opérations comme le montre la figure ci-dessous.

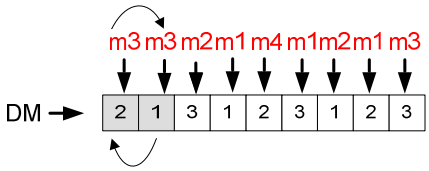


Figure 4-22 : Inversion de l'arc $(o_{1,1} \rightarrow o_{2,1})$ dans le vecteur *DM*

La nouvelle solution est donnée par la Figure 4-23. La valeur du chemin le plus long (makespan) est de 21. La solution n'a pas été améliorée le processus itératif de recherche locale s'arrête.

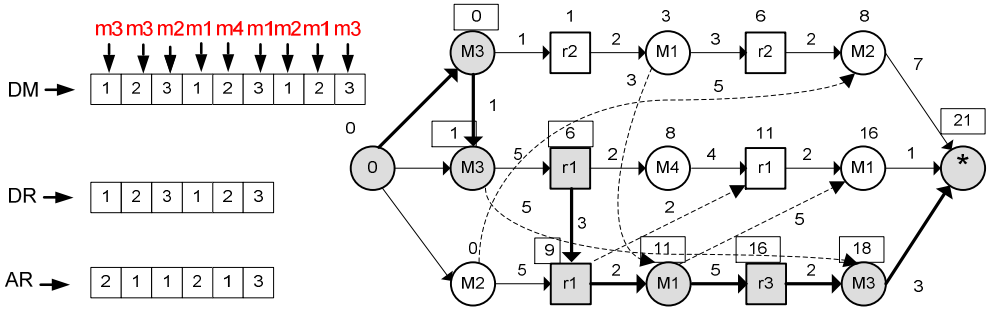


Figure 4-23 : Résultat de l'inversion de l'arc $(o_{1,1} \rightarrow o_{2,1})$

On peut remarquer que pour un bloc transport, on peut inverser un arc entre deux opérations ou bien modifier l'affection d'une opération à un robot.

Ces mécanismes sont implantés dans la procédure Local_Search dont le pseudo code est donné par l'Algorithme 4-3.

Pour simplifier la lecture de ce pseudo code, nous utilisons l'opération i à la place de $f(i)$ et l'opération j à la place de $f(\text{Pred}[i]=j)=f(j)$. Une solution S est représentée par les 4 vecteurs $DM, DR,$ AR et PRED augmentée d'une variable $C_{\max}$ qui sert à stocker le coût de cette solution. Evaluer (S.*DM, S.DR, S.AR, S.PRED, S.Cmax*), est une fonction d'évaluation d'une solution S donnée par les 3 vecteurs $DM, DR,$ AR et qui retourne le cout de cette évaluation dans la variable $C_{\max}$ et stocke le chemin critique dans le vecteur *PRED*.

Algorithme 4-3 : Recherche locale R1

```
Nom de la procédure : Local_Search
        Données
        S: solution donnée par (DM, DR, AR, PRED, Cmax)
        Sorties
        S: solution donnée par (DM, DR, AR, PRED, Cmax)
    Début
    i:=N+1                              // commencer l'analyse à partir du nœud {*}
    iter:=1                             // initialiser le compteur d'amélioration à 1
    C^best := S.Cmax                    // Initialiser le meilleur coût au coût de la solution
                                           d'entrée
      Tant que (i≠0) and (iter<=nm) faire
         j:=Pere(i)                     // j le prédécesseur immédiat de l'opération i dans le
                                           chemin critique
         Si (job[i] ≠ job[j]) alors
            Si (i et j sont deux opérations machines) alors
             Save_S :=S
             Permuter les opérations i et j dans DM
             Evaluer(S.DM, S.DR, S.AR, S.PRED, S.Cmax)

             Si ( S.Cmax < C^best ) alors
                 C^best := S.Cmax ;     // amélioration
                 i:= PRED (N+1)         // retour au nœud {*} du nouveau chemin critique
                 iter :=iter+1
             Sinon
                 S:=Save_S              // annuler le changement
                 i:=PRED[i]             // continuer l'analyse d'ancien chemin critique
             Fin Si
         Sinon
            Si (i et j sont deux opérations transport) alors
             Save_S :=S
             Permuter les opérations i et j dans DR
             Evaluer (S.DM, S.DR, S.AR, S.PRED, S.Cmax)

             Si ( S.Cmax  < C^best ) alors
                 C^best := S.Cmax ;     // amélioration
                 i:= PRED[N+1]          // retour au nœud {*} du nouveau chemin critique
                 iter :=iter+1          // incrémenter le nombre d'améliorations
             Sinon
                 S:=Save_S              // annuler le changement
                 i:= PRED[i]            // continuer l'analyse d'ancien chemin critique
             Fin Si
             Si (i est une opération de transport) et (j est une opération machine) alors
                 Save_S :=S
                 Changer l'affectation robot dans AR à l'opération i
                 Evaluer (S.DM, S.DR, S.AR, S.PRED, S.Cmax)
                 Si ( S.Cmax < C^best ) alors
                      C^best := S.Cmax ;  // amélioration
                      i:= PRED[N+1]       // retour au nœud {*} du nouveau chemin critique
                      iter :=iter+1       // incrémenter le nombre d'améliorations
                 Sinon
                      S:=Save_S                 // annuler le changement
                      i:=PRED[i]                // continuer l'analyse d'ancien chemin critique
                 Fin Si
             Fin Si
      Fin Tant que

      Retourner S
    Fin
```

4.5.2. Expérimentations numériques

Afin d'évaluer les performances de la recherche locale proposée, des expérimentations numériques ont été réalisées sur un problème de job-shop avec plusieurs robots identiques. Le problème est décrit dans les tableaux Tableau 4-5, Tableau 4-6, Tableau 4-7.

Le plan d'expérience consiste à générer des solutions canoniques, c'est-à-dire à ordonnancer toutes les opérations du job 1, puis toutes les opérations du job 2, ensuite les opérations du job 3 et à la fin les opérations du job 4. Les opérations de transport sont affectées à un seul robot. Les colonnes « $C^{*}_{\max}$ » et « $C^{1}_{\max}$ » du Tableau 4-8 donnent respectivement la solution optimale trouvée par le ***PL*** et celle de l'ordonnancement canonique. Sur toutes ces solutions nous exécutons la recherche locale avec 100 et 1000 itérations. Les résultats sont donnés respectivement dans les colonnes « $C^{2}_{\max}$ » et « $C^{3}_{\max}$ ».

Les structures des solutions sont données par les vecteurs *DM*, *DR* et *AR*, avant l'exécution de la recherche locale (i.e. pour les solutions canoniques) et après l'exécution de la recherche locale avec 100 itérations.

Pour les tests qui impliquent plusieurs robots, nous remarquons que la recherche locale affecte les opérations transports aux autres robots alors qu'elles étaient affectées au seul robot 1 au départ. Nous remarquons aussi que la solution finale est meilleure dans tous les cas après l'exécution de la recherche locale.

Tableau 4-5 : Gammes opératoires et processing time

Job 1	M3 temps opératoire : 1	M1 temps opératoire : 3	M2 temps opératoire : 6	M4 temps opératoire : 7
Job 2	M2 temps opératoire : 8	M3 temps opératoire : 5	M1 temps opératoire : 10	M4 temps opératoire : 4
Job 3	M3 temps opératoire : 5	M4 temps opératoire : 4	M1 temps opératoire : 9	M2 temps opératoire : 1
Job 4	M2 temps opératoire : 5	M1 temps opératoire : 5	M3 temps opératoire : 5	M4 temps opératoire : 3

Tableau 4-6 : Durées des déplacements à vide

	M1	**M2**	**M3**	**M4**
M1	0	1	1	1
M2	1	0	1	1
M3	1	1	0	1
M4	1	1	1	0

Tableau 4-7 : Durées des déplacements à charge

	M1	M2	M3	M4
M1	/	1	2	3
M2	1	/	1	2
M3	2	/	/	1
M4	2	/	/	/

Tableau 4-8 : Résultats de test des performances de la recherche locale

Nb de robots	$C^*_{\max}$	**Représentation en vecteurs DM DR AR**	$C^1_{\max}$	**Meilleur solution obtenu après 100 itérations**	$C^2_{\max}$	$C^3_{\max}$
1	40	DM=1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4 DR = 1,1,1,2,2,2,3,3,3,4,4,4 AR = 1,1,1,1,1,1,1,1,1,1,1,1	85	DM=1,1,2,1,1,2,2,3,3,2,3,4,3,4,4,4 DR = 1,1,2,1,2,3,3,2,4,3,4,4 AR = 1,1,1,1,1,1,1,1, 1,1,1,1,1	54	54
2	39	DM=1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4 DR = 1,1,1,2,2,2,3,3,3,4,4,4 AR = 2,2,2,2,2,2,2,2,2,2,2,2	141	DM=1,1,2,4,3,3,2,2,2,1,1,4,3,3,4,4 DR = 1,1,3,2,2,4,1,3,3,2,4,4 AR = 2,2,2,1,1,1,2,1,2,1,1,1	47	45
3	39	DM=1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4 DR = 1,1,1,2,2,2,3,3,3,4,4,4 AR = 3,3,3,3,3,3,3,3,3,3,3,3	183	DM=1,1,2,1,3,2,2,3,1,4,4,4,3,3,2,4 DR = 1,1,1„2,2,3,4,3,3,2,4,4 AR =1,2,3,1,1,1,12,2,1,1,1	45	45
4	39	DM=1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4 DR = 1,1,1,2,2,2,3,3,3,4,4,4 AR = 4,4,4,4,4,4,4,4,4,4,4,4	225	DM=1,1,2,3,1,2,2,3,2,1,4,3,4,3,3,4,4, DR = 1,1,2,1,2,3,2,3,3,4,4,4 AR =4,4,1,4,1,1,1,1,2,1,1,1	60	45

Ces résultats montrent que cette recherche est efficace d'une part et d'autre part que le nombre d'itérations est un paramètre important qu'il faudra régler. Sachant que cette recherche locale est utilisée dans une méta-heuristique, cette valeur sera choisie en fonction d'autres paramètres de la méta-heuristique.

4.6. Représentation R2 du problème du job-shop avec transport

4.6.1. Représentation avec 2 vecteurs

Nous avons décomposé le problème du job-shop avec transport en deux sous-problèmes : le problème disjonctif des opérations machine et celui des opérations transport. De manière évidente il existe des vecteurs donnant des graphes cycliques ne correspondant pas à des solutions. Toutefois, les vecteurs sont de taille raisonnable en termes de nombre d'opérations.

Nous introduisons dans ce paragraphe une nouvelle modélisation qui repose sur un vecteur contenant simultanément les opérations machines et les opérations transport. Le vecteur de séquence est appelé MTS pour une sélection machine et transport, on rappelle que la précédence des opérations est de gauche à droite, et que chaque job doit apparaitre autant de fois qu'il possède d'opérations machine et transport.

Lorsqu'on parcoure le vecteur MTS de gauche à droite, la première occurrence du job j concerne l'opération machine, tandis que l'occurrence suivante concerne l'opération de transport successeur de l'opération machine précédente et ainsi de suite. On note que pour les dernières opérations tous les jobs n'ont pas de successeur transport.

A un job j possèdant n_{j_O} opérations machines, correspondent $n_{j_T} = n_{j_O} - 1$ opérations transport. Ainsi, le nombre d'occurrence du job j dans le vecteur MTS est $n_{j_T} + n_{j_O} = 2n_{j_O} - 1$.

S'appuyant sur le problème *JS3* et la résolution optimale donnée par la Figure 4-24, on donne ci-après une illustration de cette nouvelle modélisation.

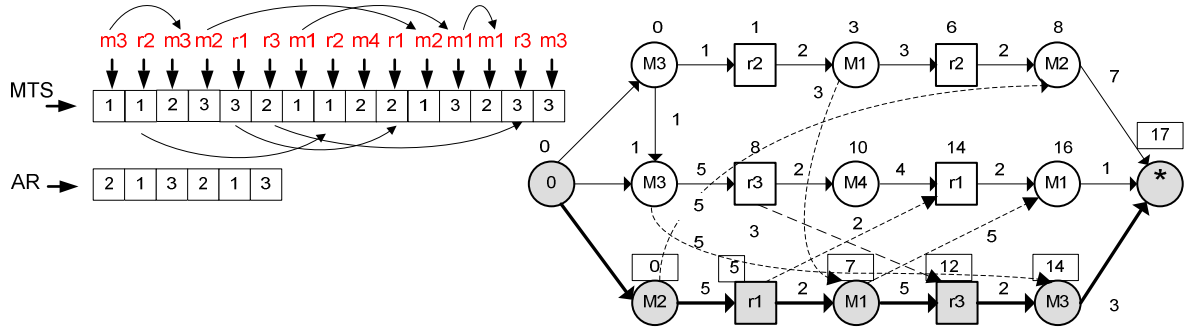


Figure 4-24 : Illustration de la modélisation à deux vecteurs

Les flèches sur le vecteur *MTS* montrent les arcs de précédences entre les différentes opérations du graphe. Avec ces deux vecteurs on peut facilement représenter le problème du job-shop avec plusieurs robots.

Contrairement à la modélisation utilisant 3 vecteurs, la modélisation à deux vecteurs ne produits pas de cycle dans le graphe, ce qui nous amène à travailler uniquement dans l'espace de recherche des solutions faisables. Ainsi nous pouvons conserver la séquence des opérations machines et transports dans un seul vecteur. Dans la suite de ce chapitre la représentation utilisant 3 vecteurs est appelée **R1** et la représentation utilisant 2 vecteurs est appelée **R2**. La preuve de l'inexistence des cycles pour cette modélisation est donnée ci-dessous.

4.6.2. Preuve de l'inexistence des cycles dans une représentation à deux vecteurs

Cette modélisation ne crée pas de cycle, et ceci lui donne un grand avantage par rapport à la première représentation, puisque à chaque fois qu'on génère une séquence, cette dernière est toujours faisable. Pour prouver la non génération de cycles, on utilise la preuve par l'absurde.

Supposons qu'il existe une séquence SEQ qui contient des cycles et soit $pos[i]$ la fonction qui donne la position de l'opération i dans la séquence SEQ. SEQ contient toutes les opérations des différents jobs avec la précédence de gauche à droite.

Pour qu'un cycle existe, il faut qu'à un certain endroit de la séquence on retrouve une opération i prédécesseur d'une autre opération j et que $pos[i] > pos[j]$.

La précédence de gauche à droite implique que les indices des opérations à gauche sont inférieurs à ceux des opérations à droite. Donc, si une opération i est prédécesseur d'une autre opération j, on doit avoir forcement $pos[j] > pos[i]$, ce qui contredit la condition d'existence de cycles.

En d'autres termes on peut imaginer qu'il peut y avoir un cycle formé de plus de deux opérations i et j, supposant qu'il existe une opération k appartenant à ce cycle, alors on a les résultats suivant :

- $pos[j] > pos[i]$ et $pos[i] > pos[j]$
- $pos[i] > pos[k]$ et $pos[k] > pos[i]$
- $pos[j] > pos[k]$ et $pos[k] > pos[j]$

De ces résultats on arrive à la conclusion qu'un tel cycle n'existe pas. Ce résultat montre

que le seul cycle qui peut exister dans la séquence SEQ est le cycle mono-sommet autrement dit il n'y aura pas de cycle.

4.7. Recherche Locale sur la représentation R2

Pour cette deuxième représentation, la recherche locale, repose sur le même type d'opérations, sauf qu'au lieu de travailler sur 3 vecteurs, cette dernière utilise uniquement deux vecteurs, le pseudo code de cette recherche locale est donné ci-dessous (Algorithme 4-4). Il est important de noter que dans cet algorithme, le robot est est changé à une opération transport, même si celle-ci n'appartient pas à un bloc transport. Nous considerons dans ce cas que le bloc transport est formé d'une seule opération.

Algorithme 4-4 : Recherche locale R2

```
Nom de la procédure : Local_Search_R2
    Input data
        Données
        S: solution donnée par (MTS, AR, PRED, Cmax)
        Sorties
        S: solution donnée par (MTS, AR, PRED, Cmax)
    Début
    i:=N+1                          // commencer l'analyse à partir du nœud {*}
    iter:=1                         // initialiser le compteur d'amélioration à 1
    C^best := S.Cmax                // Initialiser le meilleur coût au coût de la solution
                                       d'entrée
      Tant que (i≠0) and (iter<=nm) faire
         j:=Pere(i)                 // j le prédecesseur immédiat de l'opération i dans le
                                       chemin critique
         si (job[i] ≠ job[j]) alors
            Si (i et j sont deux opérations machines) alors
             Save_S :=S
              Permuter les opérations i et j dans MTS
              Evaluer(S.MTS, S.AR, S.PRED, S.Cmax)

             Si ( S.Cmax < C^best ) alors
                 C^best := S.Cmax ;     // amélioration
                 i:=PRED[N+1] // retour au nœud {*} du nouveau chemin critique
                 iter :=iter+1 ;
             Sinon
                 S:=Save_S              // annuler le changement
                 i:=PRED[i]             // continuer l'analyse d'ancien chemin critique
             Fin Si
         Sinon
            Si (i et j sont deux opérations transport) alors
             Save_S :=S
             Permuter les opérations i et j dans MTS
             Evaluer(S.MTS, S.AR, S.PRED, S.Cmax)

             Si ( S.Cmax < C^best ) alors
                 C^best := S.Cmax ;     // amélioration
                 i:= PRED[N+1]          // retour au nœud {*} du nouveau chemin critique
                 iter :=iter+1          // incrémenter le nombre d'améliorations
             Sinon
                 S:=Save_S              // annuler le changement
                 i:= PRED[i]            // continuer l'analyse d'ancien chemin critique
                 changer_robot=vrai // activer le changement de robot dans le vecteur AR

             Fin Si
        Fin Si
            Si ((i est une opération de machine) et (j est une opération transport)) ou
                 (changer-robot=vrai) alors

                 Save_S :=S
                 Si (i et j sont deux opérations transport) alors
                             Si random >0.5 alors
                             Changer l'affectation robot dans AR à l'opération i
```

```
                    Sinon
                    Changer l'affectation robot dans AR à l'opération j
                    Fin Si
            Sinon
                Changer l'affectation robot dans AR à l'opération i
            Fin Si
            Evaluer(S.MTS, S.AR, S.PRED, S.Cmax)
        Si ( S.Cmax < C^best ) alors
            C^best := S.Cmax ;        // amélioration
            i:= PRED[N+1]             // retour au noeud {*} du nouveau chemin critique
            iter :=iter+1             // incrémenter le nombre d'améliorations

        Sinon
            S:=Save_S                 // annuler le changement
            i:= PRED[i]               // continuer l'analyse d'ancien chemin critique

        Fin Si
        changer_robot=faux
    Fin Tant que
  Retourner S
Fin
```

Pour résoudre le problème du ***JS*** avec plusieurs robots, nous avons choisi les méthodes approchées, puisque les méthodes exactes sont très gourmandes en ressources.

Notre choix s'est porté en premier lieu sur l'algorithme génétique (mémétique) présenté dans le paragraphe suivant.

5. Algorithme génétique

L'utilisation des algorithmes génétiques a commencé au milieu des années soixante (1960), John Holland cité par Haupt et Haupt (2004) avait développé les principes de base des algorithmes génétiques, par la suite, cette méta-heuristique a été adaptée à plusieurs domaines et a reçu plusieurs extensions.

Cette méta-heuristique repose sur le principe de l'évolution naturelle, dans lequel, l'hérédité joue un très grand rôle dans l'évolution. Ainsi chaque population hérite des qualités de la génération précédente et transmet à son tour ses proppres propriétés à la génération suivante. Une population est composée d'individus, eux mêmes composés à son tour de chromosomes. De plus chaque chromosome est à son tour composé d'allèle et chaque allèle est constiyué de gènes ces derniers contenant les caractères héréditaires. L'évolution d'une population se fait par plusieurs mécanismes impliquant ses individus. Parmi ces mécanismes, on retrouve la sélection, le croisement et la mutation. Dans la suite de cette section consacrée aux algorithmes génétiques on donnera une description détaillée de chaque mécanisme. D'une manière générale un algorithme génétique fonctionne selon le schéma donné par le pseudo code suivant ci-dessous.

Algorithme 5-1 : Schéma globale d'un algorithme génétique

```
Nom de la procédure : Algorithme_Genétique
    pop : population
Début
    generer la population de départ pop

    Tant que nécessaire faire
       Select_deux_parent(P1,P2)
       Croiser_les_deux_parents(P1,P2)
       Obtention_de_fils
       Mutation_des_fils
       Injection des nouveaux fils dans la population
    Fin Tant que
Fin
```

Il faut générer une population initiale, puis à chaque itération le GA (pour ***G***enetic ***A***lgorithme) sélectionne deux individus *P1* et *P2* pour jouer le rôle de procréateurs. Par le mécanisme de croisement on obtient des fils qui sont mutés avec une probabilité *P*, puis on injecte les fils obtenus dans la population. Pour éviter l'agrandissement de la taille de la population, on choisi des individus à éliminer. Afin de mettre en œuvre cette méta-heuristique pour la résolution d'un problème donné, on doit associer à chaque individu un rôle à jouer ; ce dernier consiste en la représentation d'une solution, (totale ou partielle), ce qui nous amène à s'interroger sur la manière de représenter ces individus. Dans le paragraphe suivant nous donnons une liste des principales représentations existantes. Connaître ces dernières est essentiel pour bien choisir la représentation la plus adéquate au problème traité.

Cinq éléments sont nécessaires pour décrire un algorithme génétique selon Haupt et Haupt (2004) qui sont : le codage d'individu, la méthode de génération de la population initiale, la sélection, la mutation et les paramètres de dimensionnement ; ce dernier décrit la taille de la population, le nombre de génération et le critère d'arrêt. Les quatre premiers paramètres sont décris ci-après.

5.1. Le codage d'un individu

Plusieurs représentations existent, elles se basent sur l'utilisation d'un seul ou plusieurs vecteurs, où chaque élément du vecteur est censé représenter une information du problème. Ces représentations sont :

- le codage par valeurs : chaque élément du vecteur contient une valeur quelconque, cette dernière représente une information particulière (Figure 5-1) ;
- le codage binaire : on utilise un vecteur, et tous les éléments du vecteur contiennent zéro (0) ou (1). Ce codage ressemble à celui utilisé en informatique pour représenter les informations, il peut être vu comme une chaine de bits (Figure 5-2) ;
- le codage par permutation de valeurs entières : on utilise des entiers naturels au lieu des nombres binaires $\{0,1\}$ (Figure 5-3).

1,2	5,8	7	1,05	0	2	8	2	4	8,22	9	0	3	6	1

Figure 5-1 : Illustration d'un codage par valeurs

1	1	0	1	0	1	0	0	0	1	0	0	0	1	1

Figure 5-2 : Illustration d'un codage binaire

1	5	7	9	0	2	8	2	4	8	9	4	3	6	5

Figure 5-3 : Illustration d'un codage par valeur entières

Il est communément admis que les techniques de représentation indirectes sont les plus efficaces. Ces techniques ont données d'excellents résultats sur des problèmes d'ordonnancement (de *JS* par exemple) mais aussi sur des problèmes de tournées de véhicules (la méthode très

efficace de Christian Prins pour le VRP est basée sur une telle représentation El Fallahi et al., (2008)).

Pour notre notre problème, on utilise l'ensemble des vecteurs DM, DR et AT pour représenter un chromosome. La solution associée au chromosome est obtenue en créant le graphe associée à cette représentation.

De plus, nous avons choisi le codage par valeurs entières, par contre nous utilisons trois vecteurs *DM*, *DR* et *AR* pour coder un individu, dans la première représentation ***R1***, et deux vecteurs *MTS* et *AR* pour coder un individu dans la deuxième représentation ***R2***. Ces seuls vecteurs donnent une représentation partielle d'une solution car, on n'a aucune information sur la qualité de l'individu codé.

A ce codage on ajoute une variable C_{max} pour renseigner le coût de la solution modélisée et un vecteur supplémentaire appelé *PRED* pour stocker le chemin critique. Ce dernier et C_{max} sont obtenus par l'exécution d'une procédure d'évaluation *Evaluer(S.DM,S.DR,S.AR,S.PRED,S.*C_{max}*)* pour la première modélisation avec trois vecteurs et *Evaluer(S.MTS,S.AR,S.PRED,S.*C_{max}*)* pour la seconde modélisation (deux vecteurs).

La Figure 5-4 montre le codage choisi pour représenter un individu pour chaque modélisation. Le chemin critique est donné par le vecteur *PRED[i]*. Ainsi, lors de la recherche locale le chemin critique est parcouru au moyen du vecteur *PRED[i]*.

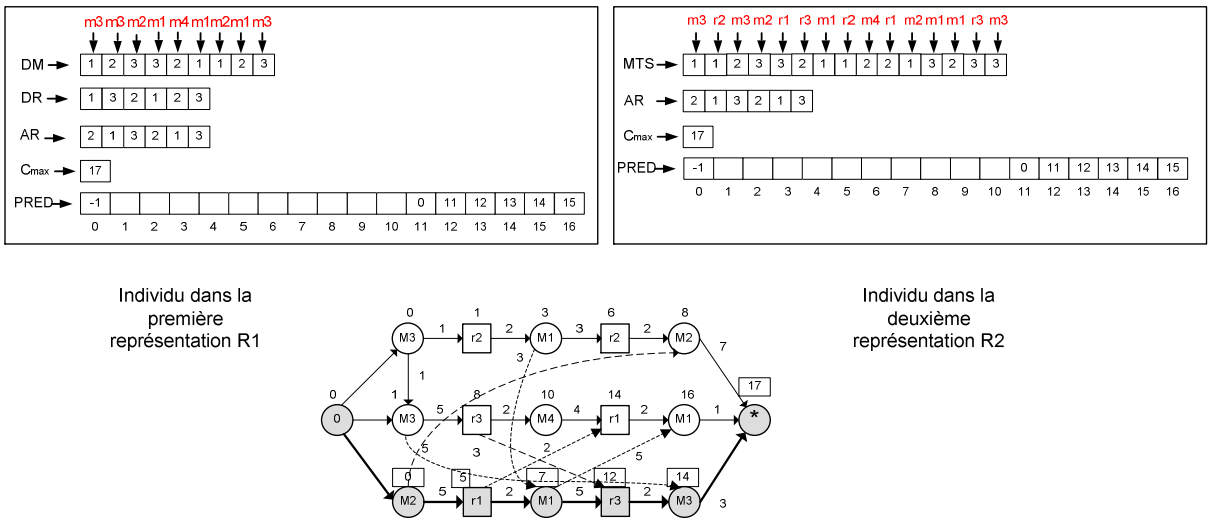


Figure 5-4 : Codage utilisé pour représenter les chromosomes (individus)

Les mécanismes de reproduction permettent d'obtenir de nouveaux individus à partir d'individus existants. Lorsque le mécanisme implique plusieurs individus on parle alors de croisement, tandis que s'il implique uniquement un seul individu il s'agit alors d'une mutation.

5.2. Croisement des chromosomes (crossover)

Pour décrire les croisements existant on utilise P1 et P2 pour les individus parents, et E1 et E2 pour les individus résultant de ce croisement (enfants). Plusieurs types de croisement existent :

- Le croisement à un seul point : dans ce croisement, un seul point est choisi sur les vecteurs P1 et P2, celui-ci divise les deux parents en 2 et on obtient alors quatre sous chaines (P1.C1 et P1.C2) et (P2.C1 et P2.C2), la concaténation de ces deux chaines nous donne deux individus enfants (E1=P1.C1•P2.C2) et (E2=P2.C1•P1.C2) comme le montre la Figure 5-5 ;
- Le croisement multipoints (Figure 5-6) : ce croisement multipoints implique plusieurs points, ainsi pour chaque individu. On peut obtenir n sous-chaines avec $n > 2$ ainsi chaque enfant peut être construit de plusieurs sous-chaines appartenant aux deux parents.
- Le croisement uniforme : Pour faire un croisement uniforme on se sert d'un vecteur appelé masque, qui est une suite binaire, on applique toujours ce masque sur les deux parents, par exemple on peut affecter 1 dans le vecteur masque pour les éléments à prendre du parent P1 et 0 pour les éléments à prendre de P2; il ressemble à un croisement multipoints avec constitution d'un seul enfant et les points sont toujours les mêmes.

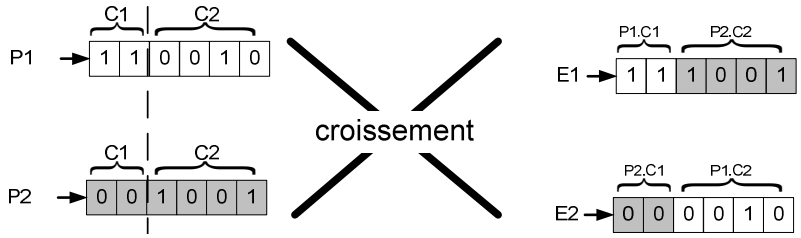


Figure 5-5 : Exemple de croisement à un seul point

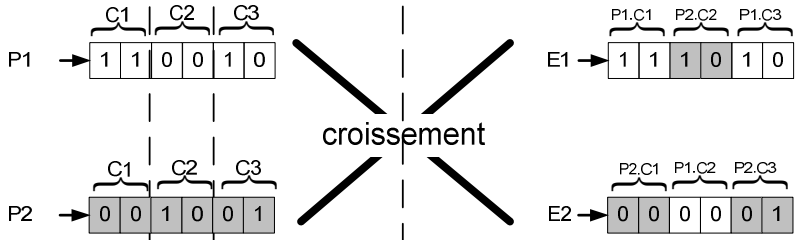


Figure 5-6 : Exemple de croisement multipoints (deux points)

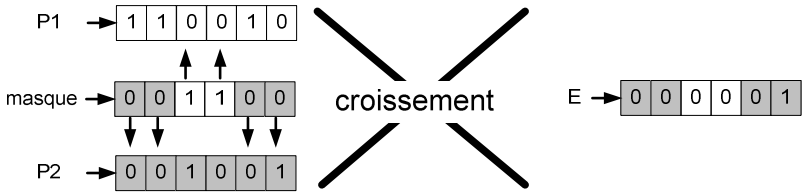


Figure 5-7 : Exemple de croisement uniforme

Nous utilisons l'opérateur de croisement appelé *GOX* inspiré de l'opérateur OX pour le TSP. Ce croisement est particulièrement bien adapté aux chromosomes à valeurs dupliquées. Un croisement de GOX à deux points de coupure Bierwirth (1995) est appliqué à deux parents et l'idée principale est de conserver l'ordre relatif des opérations des parents vers leurs fils. Ce croisement utilisé avec succès par Caumond et al., (2008) est un croisement bipoint (cas particulier du croisement multipoints). Les deux parents jouent deux rôles, le premier est

considéré comme donneur et le deuxième comme receveur. L'enfant est obtenu en sélectionnant de manière aléatoire une sous-chaine du donneur qu'on copie dans l'enfant et puis on complète les autres cases en allant de la gauche à droite du receveur tout en gardant l'intégrité des informations (si par exemple la sous-chaine du donneur comporte toute les opération du job i du donneur, on ne doit pas copier les opérations du job i du receveur, sinon l'enfant se retrouvera avec plus d'opération du job i que ce dernier n'en possède). Ce croisement est utilisé dans tous les algorithmes génétiques qui suivent. Sur la Figure 5-8 les cases grisées du donneur représentent la sous-chaîne choisie. Les cases grisées de l'enfant représentent la sous-chaîne réinsérée.

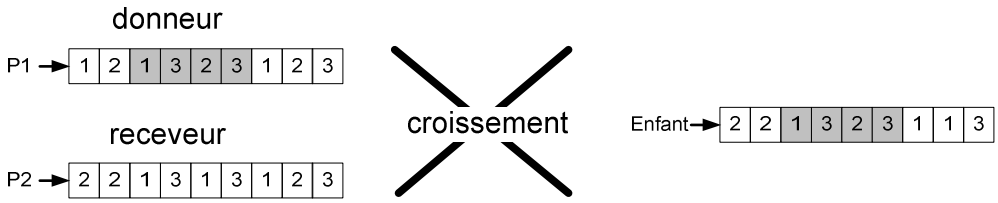


Figure 5-8 : Exemple de croisement GOX

Le rôle des parents n'étant pas symétrique, un mécanisme de permutation des rôles est ajouté au début de la procédure de croisement. L'existence de deux représentations nous amené à élaborer deux algorithmes génétiques qui ont le même schéma d'optimisation, la seule différence résidant dans la procédure de croisements. Pour réaliser un croisement sur la première représentation, le choix s'est porté sur le croisement du vecteur DM, et de générer ensuite les vecteurs DR et AR par l'heuristique ***H1***, l'exécution de la procédure *Evaluer_R1(S.DM,S.DR,S.AR,S.PRED,S.*C_{max}*,S.SIGN)* permettant de construire le vecteur $PRED$ et de renseigner la valeur de $C_{\max}$.

Dans la deuxième représentation, les croisements sont réalisés sur le vecteur MTS, ensuite le vecteur AR est obtenu par des règles heuristiques incorporées à la procédure *Evaluer_R2(S.MTS,S.AR,S.PRED,S.Cmax, S.SIGN)*. Les pseudos codes de ces deux algorithmes sont donnés ci-dessous.

Algorithme 5-2: Croisement sur la représentation R1

```
Nom de la procédure     R1_Crossover
Données
  P1, P2 : chromosomes
Sorties
  enfant : chromosome
Début
  avec probabilité de  1/2 permuter les rôles des parents P1 et P2
  k = random (1.. ∑_{i=1}^{n} n_i )
  l = random (1.. ∑_{i=1}^{n} n_i )
  Pour i:=k à l faire
       enfant.DM[i]:=P1.DM[i]
       USE[P1.DM[i]]:= USE[P1.DM[i]]+1
  Fin Pour
  j :=0
  Pour i:=1 à ∑_{i=1}^{n} n_i faire
      j :=j+1 ;
      Si (j=k) et (l+1<= ∑_{i=1}^{n} n_i ) alors
           j :=l+1 ;
      Fin Si
      Si (USE[P1.DM[i]< n_i ) et (j<= ∑_{i=1}^{n} n_i )  alors
        enfant.DM[j]:=P2.DM[i]
        USE[P2.DM[i]]:=USE[P2.DM[i]]+1
    Fin Si
  Fin Pour
  H1 (enfant.DM, enfant.DR, enfant.AR)
  Evaluer_R1(enfant.DM, enfant.DR, enfant.AR, enfant.PRED, enfant.C_max, enfant.SIGN)
Fin
```

Algorithme 5-3: Croisement sur la représentation R2

```
Nom de la procédure : R2_Crossover
Données P1, P2 : chromosomes
Sorties
  enfant : chromosome
Début
  avec probabilité de 1/2 permuter les rôles des parents P1 et P2
  Pour i:=k à l faire
       enfant.DM[i]:=P1.DM[i]
       USE[P1.DM[i]]:= USE[P1.DM[i]]+1
  Fin Pour
  j :=0
  Pour i:=1 à ∑_{i=1}^{n} n_i faire
      j :=j+1 ;
      Si (j=k) et (l+1<= ∑_{i=1}^{n} n_i ) alors
           j :=l+1 ;
      Fin Si
      Si (USE[P1.MTS[i]< n_i ) et (j<= ∑_{i=1}^{n} n_i )  alors
        enfant.MTS[j]:=P2.MTS[i]
        USE[P2.MTS[i]]:=USE[P2.MTS[i]]+1
    Fin Si
  Fin Pour
    Evaluer_R2(enfant.MTS, enfant.AR, enfant.PRED, enfant.Cmax, enfant.SIGN)
Fin
```

5.3. La mutation

La mutation, est l'un des mécanismes de diversification, qui permet de créer de la diversité dans la population pour éviter une convergence prématurée vers un même minimum local. Elle se caractérise par une modification à réaliser sur un individu. La mutation consiste alors à utiliser un système de voisinage, qui permet d'obtenir un nouvel individu à partir d'un individu existant. Le choix du système de voisinage dépend du problème étudié.

Contrairement aux algorithmes génétiques, la mutation est utilisée pour assurer la convergence pour les algorithmes mémétiques. Dans ce cas, la mutation est une recherche locale généralement basée sur un algorithme itératif. Cet algorithme consiste à parcourir le voisinage et à retenir le premier voisin améliorant (stratégie "première amélioration"). En fonction de la représentation (*R1* ou *R2*) nous utilisons une des deux recherches locales proposées (algorithmesAlgorithme 4-3 et Algorithme 4-4).

De plus, on choisit d'arrêter la recherche locale au bout de nm améliorations. Cette fonctionnalité est introduite pour deux raisons. D'abord, cette limitation permet de contrôler le temps passé à faire des mutations, car pour les problèmes de grande taille, le nombre d'itérations avant d'arriver à un optimum local peut être relativement élevé. Deuxièmement, quels que soient les réglages effectués, l'algorithme a tendance à converger prématurément. En limitant le nombre d'itérations dans la mutation, on introduit un levier pour contrôler cette convergence.

5.4. La sélection

La sélection est le mécanisme qui régule la vie d'un individu au sein de sa population, il lui permet de survivre, de se reproduire ou de mourir. La survie d'un individu dépend de sa qualité relative dans sa population.

Le but d'une méthode approchée est de fournir des solutions de très bonne qualité, ces solutions sont représentées par des individus (chromosomes). La survie d'un individu est donc fortement liée à la qualité de la solution qu'il représente, qu'elle-même est liée à la fonction objectif du problème étudié. On privilégie par conséquent les individus ayant un « score » d'adaptation le plus élevé, répondant le mieux à la fonction objectif, et on pénalise ceux en ayant un relativement faible.

La sélection repose sur une stratégie de choix. On choisit deux parents pour une opération de reproduction avec une probabilité (p1 et p2) pour chaque parent ; avec une autre on choisi l'individu à éliminé et de la même manière on choisi celui à muter etc.

Pour cela plusieurs stratégies existent dans la littérature, parmi lesquelles on trouve :

- La sélection par rang : dans cette stratégie, la population est triée selon le fitness des individus de ***1*** à ***N***, de sorte que le meilleur chromosome possède le rang ***N*** et le mauvais le rang ***1.*** La sélection d'un chromosome dépend alors du rang qu'il occupe.
- La sélection par roue de fortune (roulette wheel) : cette sélection consiste en l'utilisation d'une roue divisée en secteurs, chaque secteur représente la réponse à la fonction objectif. Plus le fitness de l'individu est adapté à la fonction objectif, plus la surface du secteur est grande et donc plus la probabilité d'être sélectionné est importante.
- La sélection par steady-state : a chaque évolution de l'algorithme génétique, les meilleurs chromosomes sont sélectionnés pour produire des chromosomes fils, qui sont injectés dans la population en retirant les chromosomes de mauvaise qualité.

- La sélection par tournoi : dans cette stratégie, la sélection, consiste en la réalisation de plusieurs tournois sur un ensemble d'individus choisis aléatoirement dans la population, afin de choisir l'individu qui possède le meilleur score d'adaptation. La méthode consiste à sélectionner une paire d'individus pour faire un tournoi, qui en determine un seul avec une probabilité dite de victoire. Cette sélection s'appelle tournoi binaire, on peut aussi faire un tournoi sur trois individus voir plus, et on sélectionne le meilleurs de tous les individus.
- La sélection d'élitisme : pour éviter la perte des meilleurs chromosomes à chaque changement de population, l'élitisme consiste à injecter dans la nouvelle population une proportion de meilleurs individus de la population précédente et la création du reste des individus selon une procédure usuelle de génération de chromosome.

Nous choix s'est porté sur l'utilisation de la sélection par tournoi binaire pour le choix de parents pour le croisement et la sélection d'élitisme pour la régénération de la population.

5.5. La détection des doubles par une fonction de hachage

La sélection, permet à la population de converger vers les meilleurs individus, ce qui a le principal inconvénient de converger vers le même type d'individus ce qui constitue un appauvrissement du matériel génétique. Cela nuit à la convergence de l'algorithme, car ce dernier se bloque prématurément dans un minimum local. Pour éviter cette situation, un mécanisme de détection de doublons est mis en œuvre, sachant que différents chromosomes peuvent amener à une même solution. Notre technique de détection de doublons consiste à calculer une signature basée sur l'ordonnancement plutôt que sur le chromosome. Cette signature est une valeur qui représente la somme des carrés des dates de début de toutes les opérations du problème modulo k, avec k un grand entier choisi d'une manière empirique. Ainsi, si on note par t_i, la date de début de l'opération $i \in E$, la signature d'un chromosome C est alors $sign = \sum_{i \in E} t_i^2 \bmod k$. Il faut noter que deux individus avec des signatures différentes peuvent amener à un même ordonnancement. Pour minimiser ce risque, il faut que k soit suffisamment grand. La détection consiste en l'utilisation d'un grand vecteur à une dimension appelé $DOUBLE$ de taille k où chaque indice de ce vecteur représente une signature. A chaque fois qu'on génère un individu avec la signature Sn on incrémente la case $DOUBLE[Sn]$, ainsi cette case comporte le nombre d'individus générés possédant la signature Sn. Pour savoir si un individu possédant une signature Sn est un doublon, il suffit de consulter la case $DOUBLE[Sn]$ ce qui se fait en $O(1)$. Pour être efficace, la taille du vecteur $DOUBLE$ doit être très grande, car si k est petit, plusieurs individus différents auront la même signature, ne pas les accepter nuira à l'accessibilité de l'espace de recherche. La taille actuelles des mémoires centrales des ordinateurs permettent d'envisager des valeurs très grande de k et permettent, ainsi, d'utiliser cette technique bien que gourmande en mémoire.

5.6. La génération de la population initiale

L'élément de base d'un algorithme génétique est la population. Cette dernière à partir d'individus initiaux converge vers une autre possédant des individus ayant une meilleure qualité par rapport à la fonction objectif, et ceci en héritant des qualités des individus des générations précédentes. Au départ il faut donc générer cette population.

Dans l'algorithme mémétique utilisant la première représentation ***R1***, on commence par

générer 1 chromosome avec l'heuristique ***H2*** puis on génère n chromosomes canoniques où n est le nombre de jobs présents dans le problème, le reste de la population est obtenu comme suit (Algorithme 5-4) :

- on génère le vecteur *DM* aléatoirement par une heuristique ***H3*** sachant que le vecteur *DM* contient autant d'éléments que d'opérations machines où chaque job j doit apparaître n_j fois où n_j représente le nombre d'opérations machines que possède le job j. A l'aide du vecteur *DM* on génère les autres *DR* et *AR* par l'heuristique ***H1***.
- L'exécution de la procédure *Evaluer_R1(S.DM,S.DR,S.AR,S.PRED,S.*C_{max}*,S.SIGN)* permet de compléter le chromosome.

Algorithme 5-4 : Génération de la population initiale dans la représentation R1

```
Nom de la procédure :Pop_intiale_R1

Paramètres de l'algorithme
    pop     : population initiale
    N_pop   : taille de la population
    n       : nombre de jobs présents dans le problème
    c       : chromosome
    DOUBLE  : vecteur utilser pour detecter les doublons
Debut
    H2(c)
    Evaluer_R1 (C.DM, C.DR, C.AR, C.PRED, C.Cmax, C.SIGN)
    Add(pop,c,DOUBLE,i)

    Pour i:=2 à n+1 faire
      Generer_DM_canonique(C.DM)
      H1(C)
      Evaluer_R1 (C.DM, C.DR, C.AR, C.PRED, C.Cmax, C.SIGN)
      Si Add(pop,c,DOUBLE,i) alors

    Fin pour
    i:=n+2
    Tant que i< N_pop faire
      H3(C.DM)
      H1(C)
      Evaluer_R1 (C.DM, C.DR, C.AR, C.PRED, C.Cmax, C.SIGN)
      Si Add(pop,c,DOUBLE,i) alors
      i :=i+1
    Fin Tant que

  Fin
```

Lors de la création de la population initiale, on met à jour au fur et à mesure le tableau *DOUBLE* au moyen de la procédure *add*(*C, pop, DOUBLE, i*) dont la structure est donnée par l'Algorithme 5-5. Ainsi, on est en mesure d'interdire les doubles dans la population.

Algorithme 5-5 : Fonction d'ajout des chromosomes dans la population

```
Nom de la fonction : add(C,pop,DOUBLE,i): boolean;

Paramètres de cette fonction
    pop    : population initiale
    c      : chromosome
    DOUBLE : vecteur utilser pour detecter les doublons
    Ind_ajout : place dans laquelle on inser le chromosome c
Début
    Si DOUBLE[C.SIGN] > 0 alors
      retourner faux // ajout non réalisé
    Sinon
      DOUBLE[pop[Ind_ajout ].SIGN] := DOUBLE[pop[Ind_ajout ].SIGN] -1
      DOUBLE[C.SIGN] := DOUBLE[C.SIGN]+1
      pop[Ind_ajout ] :=C
      retourner vrai // ajout réussi
    Fin Si

  Fin
```

Dans le cas de la représentation R2, la population initiale est générée comme suit (algorithme 4-9) : on génère les vecteurs *MTS* par une variante de l'heuristique ***H3*** et l'exécution de la procédure *Evaluer _R2(S.MTS,S.AR,S.PRED,S.Cmax, S.SIGN))* permet de compléter le chromosome.

Algorithme 5-6 : Génération de la population initiale dans la représentation R2

```
Nom de la Procédure : Pop_intiale_R2

Paramètres de cet algorithme
    pop    : population initiale
    N_pop  : taille de la population
    n      : nombre de jobs présents dans le problème
    c      : chromosome
    DOUBLE : vecteur utilser pour detecter les doublons
début
    i:=1
    Tant que i< N_pop faire
      H3(C.MTS)
         Evaluer_R2(C.MTS, C.AR, C.PRED, C.Cmax, C.SIGN)
      Si Add(pop,c,DOUBLE,i) alors
      i :=i+1
    Fin Tant que

  Fin
```

5.7. Méthode de remplacement et condition d'arrêt

Malgré les différents mécanismes introduits pour limiter la convergence prématurée, il peut y avoir un appovrissement de la population limitant le domaine de recherche, c'est pourquoi nous introduisons une procédure de redémarrage qui perturbe les chromosomes pour obtenir une population voisine.

Nous avons réalisé deux implémentations, la première utilise la représentation *R1*, cet

algorithme mémétique se nomme ***MA_R1***, la seconde s'appui sur la modélisation *R2* et cet algorithme mémétique se nomme ***MA_R2***, pour ces deux implémentations, le schéma d'optimisation reste le même.

5.8. Structure générale de l'algorithme mémétique

Les différentes parties de l'algorithme mémétique décrites ci-dessus sont utilisées conjointement dans notre algorithme mémétique. L'algorithme 4-11 présente l'algorithme de principe utilisé.

Algorithme 5-7 : Algorithme mémétique

```
nom de Procédure : Memetic_Algorithm
Paramètres de cet algorithme
    CRA : critère d'arrêt
    mni : nombre maximal d'iteration du MA
    np  : nombre maximal d'iteration improductive (non améliorantes) avant le restart
    pm  : probabilité d'éxécution de la recherche locale
    p   : pourcentage de la population à remplacer
    pop : population
    r   : nombre de robots
debut
    Pop_initiale                      // génération de la population initiale suivant la
                                         représentation R1 ou R2
    npi := 0 ;                        // itération courante
    ni  := 0 ;                        // nombre d'itération improductives
    Repeter
        SelectSolution (P1,P2)
        C := R1_Crossover(P1,P2)              // dans la modélisation R1
        C := R2_Crossover(P1,P2)              // dans la modélisation R2
        LocalSearch(C) avec une probabilité pm       //  recherche  locale  suivant  la
        représentation R1 ou R1
        Select(ind)                           //  séléctionnet  l'indice  du  chomosome  à
        remplacer
        Si add(Pop,C, DOUBLE,ind) alors       // remplacer le chromosome Pop[ind] par C
          trier(Pop)                          // trier la population
          Si (npi=np) alors
             Restart(Pop,p)                          // regénérer p nouveaux chromosomes
           Fin Si
        Fin Si
    Jusqu'à (CRA).                        // critère d'arrêt
Fin
```

Le schéma d'optimisation est basé sur un algorithme mémétique incrémental et hybride. À l'initialisation, l'algorithme consiste à construire la population initiale. Les compteurs ni (pour le nombre d'itérations sans amélioration) et npi (nombre total d'itérations) sont initialisés à zéro (ligne 2 et 3).

D'abord les deux parents *P1* et *P2* sont choisis par tournoi binaire (*SelectSolution(P1,P2)*) pour l'opération de reproduction (croisement dans notre cas). Selon la représentation on exécute le bloc correspondant pour compléter le chromosome enfant obtenu « C ». Avec une probabilité *Pm* l'individu « C » subit une mutation pour être amélioré. La future place de l'individu est donnée par *Select(ind)* qui renvoi l'indice du chromosome à remplacer par « C ». Que l'individu ait été muté ou non, on vérifie si c'est un double avec la procédure *add(Pop,C, DOUBLE,ind)*. Si c'est un double, l'individu est abandonné, sinon on l'insère dans la population.

On signale aussi qu'on compte le nombre d'itérations improductives dans le sens où le meilleur chromosome de la population reste inchangé en incrémentant la variable *npi*. Si cette dernière atteint le nombre *np*, une opération de régénération de la population par le mécanisme

d'élitisme est exécutée et ceci est réalisé par la procédure *Restart(Pop,p)* où p est le pourcentage de chromosomes à régénérer. Cet algorithme fonctionne d'une manière itérative jusqu'à la satisfaction d'un critère d'arrêt qui peut être un nombre maximum d'itérations à réaliser, ou un temps maximum d'exécution ou l'atteinte d'une valeur pour le meilleur individu de la population etc.

Nos tests ont montré qu'une exécution de l'algorithme à k robots obtient avec les mêmes paramètres et avec le même critère d'arrêt des résultats moins bons que les résultats d'une exécution avec k' robots où $k' \prec k$. Cette situation bien qu'anormale est due au caractère stochastique de la méthode. Pour palier ce problème, on a implémenté un algorithme incrémental, qui est exécuté, si le nombre de robots est supérieur à 1.

Cet algorithme fonctionne comme suit : au départ, une population avec un robot est générée, l'algorithme est exécuté pendant un certain temps. Puis le meilleur individu est choisi ; ensuite une nouvelle population contenant cet individu est générée en incrémentant le nombre de robots. Ce processus est réitéré jusqu'à l'atteinte du nombre maximal de robots contenu dans l'énoncé du problème.

Il faut renseigner le critère d'arrêt pour chaque nombre de robots qui correspond à une itération de l'algorithme incrémental, le pseudo code suivant donne le schéma de fonctionnement de cet algorithme.

Algorithme 5-8 : Résolution incrémentale des instances avec plusieurs robots

```
Nom de la procédure :   Multi_Robots
Données
    Mni         : nombre maximale d'itération du GA
    nb_robot    : nombre de robots présents dans le problème
    P           : pourcentage de la population à remplacer à restart
    Np          : nombre maximale d'itérations improductives avant le restart
    pm          : probabilité de la recherche locale
    Pop         : population
Début
      InitializePopulation (Pop)   // initialiser la population avec 1 robot
    Pour r:=1 à max_robot Faire
              MA_R1(mni,np, pm, p, pop, r)          // dans la modélisation R1
              MA_R2(mni,np, pm, p, pop, r)          // dans la modélisation R1
              initializePopulation (Pop,r)          // ne garder que le meilleur
                                                       individu de l'éxécution
                                                       précedente,   et   générer   les
                                                       autres individus avec r robots
    Fin Pour
Fin
```

Pour tester la qualité de notre méthode, il nous faut des benchmarks à l'image de ceux qu'existent pour le job-shop ou pour les autres problèmes. A notre connaissance, dans la littérature uniquement deux benchmarks existent, le premier est celui de Bilge et Ulusoy (1995), qui concerne le domaine des AGV (Automated Guided Vehicules) où, on utilise deux transporteurs identiques et une configuration spéciale de l'atelier. En effet, tous les jobs sont à chercher depuis une machine d'entrée L (pour Load station) et chaque job ayant terminé son exécution doit être acheminé à la station de déchargement U (pour Unload station).

Le deuxième Benchmark est celui de Hurink et Knust (2002), qui concerne l'utilisation d'un seul robot pour des problèmes de job-shop. Ayant constaté le manque de benchmark traitant le cas de plusieurs robots nous avons étendu ce dernier pour combler ce vide.

6. Application numérique

Toutes les procédures sont implémentées sous Delphi 7.0 sous Windows XP Media Center. L'exécution est réalisée sur un ordinateur disposant d'un processeur Pentium E6300 de 1.8 Ghz, et de 2 GB de mémoire vive. Pour mesurer la performance des algorithmes ***MA_R1*** et ***MA_R2*** nous avons envisagé 5 réplications avec des racines différentes pour le générateur aléatoire.

Les notations suivantes sont utilisées dans les tableaux des résultats :

N	nombre total d'opération à ordonnancer ;
LB[1]	la borne inférieur introduite par Hurink et Knust (2005) ;
LB[2]	la borne inférieur trouvé par l'exécution de notre programme linéaire MILP dans un environnement Cplex avec 3600 secondes comme critère d'arrêt ;
BFS	meilleure solution trouvée (pour best found solution) par notre Algorithme dépendant du temps d'arrêt ;
BFS^{10min}	meillieure solution trouvée par notre Algorithme en 10 minutes de calcul ;
BFS^{30min}	meillieure solution trouvée par notre Algorithme en 30 minutes de calcul ;
BFS^{lg}	meillieure solution trouvée par notre Algorithme en 1 heure de calcul ;
BFS^{mixte}	meillieure solution trouvée par notre Algorithme en 1 heure de calcul, avec l'utilisation de deux jeux de paramètres ;
UB^{one}	meilleure solution trouvée en 10 minutes de calcul par « one-stage approach » proposée par Hurink et Knust (2005) ;
UB^{two}	meilleure solution trouvée en 10 minutes de calcul par « two -stage approach » proposée par Hurink et Knust (2005) ;
UB^{sh}	meilleure solution trouvée en 10 minutes de calcul par « combined approach » proposée par Hurink et Knust (2005) ;
UB^{lg}	meilleure solution trouvée en 1 heure de calcul par « combined approach » proposée par Hurink et Knust (2005) ;
BSH	« best single past heuristic » proposée par Bilge et Ulusoy (1995) ;
ALG1	algorithme 1 proposé par Bilge et Ulusoy (1995) ;
ALG2	algorithme 2 proposé par Bilge et Ulusoy (1995) ;
R1 MM	règle 1 minimum makespan proposé par Bilge et Ulusoy (1995) ;
STW	« sliding time window » premièrement proposé par Bilge et Ulusoy (1995) ;
UGA	l'algorithme génétique proposé par Bilge et al., (1997) ;
GA	l'algorithme génétique proposé par **Abdelmaguid et al., (2004)** ;
ILS	« Itérative Local Search » proposée par Deroussi et al., (2008) ;
SA	« Simulated Annealing » proposé par Deroussi et al., (2008) ;
SALS	« Simulated Annealing » Local Search proposé par Deroussi et al., (2008) ;
AM 1-2	amélioration (gain) réalisée en passant de 1 robot à 2 robots ;
AM 2-3	amélioration (gain) réalisée en passant de 2 robots à 3 robots ;
AM 1-3	amélioration (gain) réalisée en passant de 1 robot à 3 robots ;
AM	amélioration moyenne ;
Stop_cr	critère d'arrêt ;
Dev_%	déviation en pourcentage de la meilleure solution trouvée par une méthode par rapport à la borne inférieur du problème $\frac{BFS - LB}{LB}$;

Nous utilisons les réglages suivants pour les différents paramètres :

- Les paramètres suivants sont utilisés pour l'algorithme MA_R1

  nm: 60 // *nombre maximal d'itération de la recherche locale;*

  nc: 115 // *nombre de chromosomes dans la population;*

  np: 15000 // *nombre d'itération non améliorante avant de faire un restart;*

  pm: 32 // *probabilité de la recherche locale;*

  pr: 32 // *pourcentage de chromosomes à remplacer pendant le restart;*

- L'algorithme MA_R2 utilise deux jeux de paramètres :

  Le premier jeu de paramètres est utilisé pour toutes les instances :

  nm: 150 // *nombre maximal d'itération de la recherche locale;*

  nc: 90 // *nombre de chromosomes dans la population;*

  np: 2200 // *nombre d'itération non améliorante avant de faire un restart;*

  pm: 85 // *probabilité de la recherche local;*

  pr: 85 // *pourcentage de chromosomes à remplacer pendant le restart;*

  Le deuxième jeu concerne uniquement les instances avec un seul robot :

  nm: 10 // *nombre maximal d'itération de la recherche locale;*

  nc: 95 // *nombre de chromosomes dans la population;*

  np: 1000 // *nombre d'itération non améliorante avant de faire un restart;*

  pm: 85 // *probabilité de la recherche locale;*

  pr: 75 // *pourcentage de chromosomes à remplacer pendant le restart;*

6.1. Les instances avec un seul robot

Les instances du job-shop avec transport et un seul robot sont proposées par Hurink et Knust (2005) et sont des instances carrées de job-shop composées :

- d'instances de taille moyenne *P01* (6x6) contenant 66 opérations : 36 opérations machines 6 jobs et 6 machines et 30 opérations transport ;
- d'instances de grande taille *P02* (10x10) contenant 190 opérations : 100 opérations machines 10 jobs et 10 machines et 90 opérations transport.

Pour ces instance les auteurs fournissent les déplacements du robot à charge et à vide, ces derniers dépendent des machines pour la majorité des instances, mais pour certaines ils dépendent à la fois des machines et des jobs transportés entre ces machines.

Dans cette partie on donne les résultats obtenus par les algorithmes proposés avec un seul robot sur les instances de Hurink et Knust. Ces derniers proposent dans Hurink et Knust (2005) plusieurs approches dédiées au job-shop avec un seul robot, dont trois recherches tabou basées sur la modélisation du problème proposée dans Knust (1999). La première méthode est appelée "one-stage approach" qui transforme le problème en un problème de job-shop avec des séquences-dépendent setup times. La seconde méthode appelée "two-stage approach" fonctionne

en deux étapes, pendant la première étape on choisit un ordre de passage des jobs sur les machines, ce qui revient à fixer une sélection machine, se basant sur cette sélection machine la deuxième étape cherche le meilleur ordre pour faire les opérations transport,

Une combinaison des deux approche est mise en œuvre dans "combined approach" afin d'intensifier la recherche. Hurink et Knust ont testé ces trois approches, et pour chacune ils donnent la meilleure solution trouvée après plusieurs exécutions de la méthode utilisant différents paramètres.

Hurink et Knust donnent des résultats pour 10 minutes, 30 minutes et 1 heure de calcul.

(BFS^{10min}), (BFS^{30min}) et (BFS^{long}) concernent les résultats trouvés par nos algorithmes ***MA_R2*** respectivement en 10 minutes, 30 minutes, et une heure de calcul. La colonne BFS^{mixte} donne la meilleure solution obtenue en utilisant les deux jeux de paramètres en une heure de calcul. (BFS^{MA_R1}) concerne les résultats trouvés par l'algorithme mémétique ***MA_R1*** utilisant la représentation **R1** obtenu en une heure de calcul. Comme le montrent le Tableau 6-1 notre algorithme mémétique, basé sur la deuxième modélisation obtient de meilleurs résultat que toutes les autres méthodes pour le cas à un robot. En outre d'avoir la meilleur déviation par rapport à la borne inférieur, il a permis de trouver plusieurs solutions optimales. Compte tenu que ***MA_R2*** fournit les meilleurs résultats pour les tests à 1 robot sur ce benchmark, il a été retenu pour les tests à deux puis à trois robots sur ce même benchmark.

Tableau 6-1 : Résultats des expérimentations numériques sur les insrances de Hurink et Knust (2005)

Résultats et déviation par rapport à la borne in trouvés par les approches d'Hurink et Knust et nos deux algorithmes MA_R1 et MA_R2

		Résultats d'Hurink et Knust, (2005)									***MA_R2***								***MA_R1***	
	N	***LB[1]***	***UB^{one}***	***Dev1 %***	***UB^{two}***	***Dev2%***	***UB^{sh}***	***Dev3 %***	***UB^{lg}***	***Dev4%***	***BFS^{10min}***	***Dev5%***	***BFS^{30min}***	***Dev6%***	***BFS^{lg}***	***Dev7 %***	***BFS^{mixte}***	***Dev8 %***	***BFS^{MA_R1}***	***Dev9 %***
T1-P01.dat.D1_d1	66	82	**87**	6.10	88	7.32	88	7.32			**87***	6.10	**87***	6.10	**87***	6.10	**87***	6.10	92	12.20
T1-P01.dat.D1_t1	66	77	**81**	5.19	83	7.79	83	7.79			**81**	5.19	**81**	5.19	**81**	5.19	**81**	5.19	83	7.79
T1-P01.dat.D2_d1	66	147	**148**	0.68	155	5.44	153	4.08			**148**	0.68	**148**	0.68	**148**	0.68	**148**	0.68	156	6.12
T1-P01.dat.D3_d1	66	213	217	1.88	219	2.82	216	1.41			**213***	0.00	**213***	0.00	**213***	0.00	**213***	0.00	220	3.29
T1-P01.dat.tkl.1	66	136	137	0.74	138	1.47	141	3.68			**136***	0.00	**136***	0.00	**136***	0.00	**136***	0.00	143	5.15
T1-P01.dat.T2_t1	66	71	**74**	4.23	76	7.04	74	4.23			**74***	4.23	**74***	4.23	**74***	4.23	**74***	4.23	78	9.86
T1-P01.dat.T3_t0	66	92	**92**	0	94	2.17	93	1.09			**92***	0.00	**92***	0.00	**92***	0.00	**92***	0.00	**92***	0.00
Déviation moyenne sur les instances P01				***2.69***		***4.87***		***4.23***		***NA***		***2.31***		***2.31***		***2.31***		***2.31***		**6,34**
T1-P02.dat.D1_d1	190	880	1044	18.64	1035	17.61	1013	15.11	**990**	12.50	1012	15.00	1002	13.86	1000	13.64	995	13.07	1048	19.09
T1-P02.dat.D1_t0	190	880	1042	18.41	1055	19.89	989	12.39	989	12.39	1017	15.57	1015	15.34	1009	14.66	**967**	9.89	1039	18.07
T1-P02.dat.D1_t1	190	880	1016	15.45	1021	16.02	995	13.07	989	12.39	**983**	11.70	**983**	11.70	**983**	11.70	**983**	11.70	1004	14.09
T1-P02.dat.D2_d1	190	892	1070	19.96	1064	19.28	1004	12.56	**993**	11.32	1045	17.15	1023	14.69	1023	14.69	1023	14.69	1082	21.30
T1-P02.dat.D3_d1	190	906	1070	18.1	1084	19.65	1078	18.98	1072	18.32	1100	21.41	1074	18.54	1074	18.54	**1050**	15.89	1123	23.95
T1-P02.dat.D5_t2	190	1167	1325	13.54	1390	19.11	1383	18.51	1371	17.48	1361	16.62	1327	13.71	**1317**	12.85	**1317**	12.85	1362	16.71
T1-P02.dat.T1_t1	190	874	1006	15.1	1053	20.48	1022	16.93	1018	16.48	978	11.90	978	11.90	978	11.90	**969**	10.87	1004	14.87
T1-P02.dat.T2_t1	190	880	1015	15.34	1058	20.23	1053	19.66	1030	17.05	993	12.84	993	12.84	993	12.84	**982**	11.59	1005	14.20
T1-P02.dat.T5_t2	190	898	1102	22.72	1102	22.72	1090	21.38	1020	13.59	1022	13.81	1022	13.81	1015	13.03	**1004**	11.80	1057	17.71
T1-P02.dat.tkl.1	190	888	1086	22.3	1090	22.75	1061	19.48	1018	14.64	1009	13.63	1009	13.63	1009	13.63	**1009**	13.63	1049	18.13
T1-P02.dat.tkl.2	190	896	1028	14.73	1073	19.75	1058	18.08	1014	13.17	1002	11.83	**992**	10.71	**992**	10.71	**992**	10.71	1061	18.42
T1-P02.dat. mult0.5_D1_d1	190	482	**555**	15.15	578	19.92	562	16.6	558	15.77	581	20.54	580	20.33	580	20.33	556	15.35	596	23.65
T1-P02.dat. mult0.5_D1_t1	190	482	544	12.86	559	15.98	551	14.32	**542**	12.45	546	13.28	**542**	12.45	**542**	12.45	**542**	12.45	582	20.75
T1-P02.dat. mult0.5_D2_d1	190	497	**633**	27.36	680	36.82	674	35.61	666	34.00	673	35.41	654	31.59	654	31.59	654	31.59	687	38.23
T1-P02.dat. mult0.5_D2_t0	190	497	**578**	16.3	603	21.33	595	19.72	595	19.72	584	17.51	584	17.51	584	17.51	584	17.51	616	23.94
T1-P02.dat. mult0.5_D2_t1	190	497	**613**	23.34	627	26.16	621	24.95	620	24.75	620	24.75	620	24.75	620	24.75	619	24.55	641	28.97
Déviation moyenne sur les instances P02				***18.08***		***21.11***		***18.58***		***16.63***		***17.06***		***16.09***		***15.93***		***14.88***		**20.76**
DEVIATION MOYENNE TOTAL				***13.40***		***16.16***		***14.22***		***NA***		***12.57***		***11.89***		***11.78***		***11.06***		***16.36***

L'étoile montre les solutions optimales trouvées par la modélisation linéaire, et les cases grises indiquent les nouvelles meilleures solutions pour les instances concernées

Analyse des résultats

Les résultats des tests utilisant un seul robot sont synthétisés dans le Tableau 6-2. Ces résultats montrent que la modélisation *R2* est meilleure que toutes les autres méthodes en déviation moyenne. Pour les instances de taille moyenne (*P01*) l'algorithme mémétique donne une déviation de ***2.31%*** par rapport à la borne inférieur *LB*. Cet algorithme a le mérite d'introduire de nouvelles solutions optimales T1-P01.dat.D3_d1 et T1-P01.dat.tkl.1 et également de donner les meilleures solutions de toutes les méthodes de la littérature. L'implémentation de la modélisation linéaire prouve que 5 solutions trouvées par notre algorithme (MA_R2) sont optimales.

Pour les grandes instances (*P02*), (BFS^{10min}) donne une déviation moyenne par rapport à la borne inférieure *LB* de *17.06%* en 10 minutes de calcul, et elle donne des résultats 9/16 fois meilleurs par rapport aux résultats fournis par la « one-stage approach » (UB^{one}) et 14/16 fois meilleurs comparés aux résultats de la « two-stage approach » (UB^{two}). Même remarque pour (BFS^{30min}) qui donne une déviation de *16.09%* pour 30 minutes de temps de calcul. Pour ces instances, la combined approach (UB^{lg}) est la meilleure. L'algorithme MA_R2 avec une heure de calcul, a le mérite de donner une meilleure déviation moyenne et d'améliorer 10 sur 16 fois les meilleurs résultats connus pour toutes approches confondues.

Le Tableau 6-2 synthétise les déviations moyennes des différentes approches par rapport aux meilleures solutions connues.

Tableau 6-2 : Déviation moyenne des résultats de toutes les méthodes par rapport aux meilleurs solutions connus pour chaque instances

	Résultats d'Hurink et Knust Hurink et Knust (2005)				**MA_R2**			
Instances	One stage approach UB^{one}	Two stage approach UB^{two}	Combined approach 10 min UB^{sh}	Combined approach 60min UB^{lg}	BFS^{10min} 10 min	BFS^{30min} 30 min	BFS^{long} 60 min	BFS^{mixte} 60 min
P01	0,37	2,50	1,88	NA	0,0	0,0	0,0	0,0
P02	3,46	6,07	3,83	2,10	2,47	1,63	1,49	0,57
Total	2,52	4,98	3,24	NA	1,72	1,13	1,04	0,39

6.2. Les instances avec plusieurs robots

6.2.1. Les instances étendues de Hurink et Knust (2005)

À notre connaissance il n'existe pas de résultats, ni d'instances pour le job-shop avec transport avec plusieurs robots et des vitesses différentes. Pour réaliser nos tests, on a étendu les instances de Hurink et Knust (2005) dont les temps de déplacement dépendent uniquement des machines pour faire 3 groupes d'instances :

- le premier groupe utilise la même vitesse du robot que celui utilisé par Hurink et Knust ;
- le deuxième groupe utilise des robots ayant des relations telles que, le deuxième robot est deux fois plus lent que le premier et le troisième 4 fois plus lent que le premier et ainsi de suite.
- le troisième groupe concerne des robots différents dont les temps de déplacement sont générés aléatoirement.

Les instances de Hurink et Knust's ainsi que l'extension qui ont été réalisées sont disponibles en téléchargement à l'adresse : http://www.isima.fr/~larabi.

Pour tester la performance de notre algorithme on a choisi de faire le test sur les instances du groupe 1 comportant des robots identiques, pour ainsi calculer le gain réalisé en passant de un à deux robots puis à trois robots et ainsi de suite. Les résultats de notre algorithme sont comparés aux bornes inférieures (*LB*) données par l'implémentation de notre ***PL*** obtenu après une heure de calcul.

Le Tableau 6-3 montre que les résultats de **MA_R2** sont très bons, sur les instances du (*P01*) la déviation par rapport à la borne inférieure est de *11.66%* par contre pour les instances (*P02*) la déviation est de *37.59%*. Cette déviation peut s'expliquer par la mauvaise qualité des bornes inférieures, car ces dernières sont obtenues en une heure de calcul de notre ***PL***. En effet, le nombre d'opérations est de 190 pour chaque instance appartenant au groupe (*P02*), ce qui rend difficile le calcul de bonnes bornes inférieures en une heure de calcul. La déviation moyenne totale est de *29.69 %*.

Tableau 6-3 : Résultats sur les instances de job-shop avec deux robots identiques

instances	PL		MA_R2	
	N	***LB***	***BFS^{long}***	***Dev%***
T1-P01.dat.D1_d1	66	60	63	5.00
T1-P01.dat.D1_t1	66	62	62	0.00
T1-P01.dat.D2_d1	66	71	84	18.31
T1-P01.dat.D3_d1	66	84	112	33.33
T1-P01.dat.tkl.1	66	68	77	13.24
T1-P01.dat.T2_t1	66	58	62	6.90
T1-P01.dat.T3_t0	66	62	65	4.84
Déviation moyenne sur les instances P01				***11.66***
T1-P02.dat.D1_d1	190	700	1009	44.14
T1-P02.dat.D1_t0	190	700	982	40.29
T1-P02.dat.D1_t1	190	700	985	40.71
T1-P02.dat.D2_d1	190	719	991	37.83
T1-P02.dat.D3_d1	190	757	1025	35.40
T1-P02.dat.D5_t2	190	809	1029	27.19
T1-P02.dat.T1_t1	190	689	960	39.33
T1-P02.dat.T2_t1	190	697	985	41.32
T1-P02.dat.T5_t2	190	722	1015	40.58
T1-P02.dat.tkl.1	190	708	1012	42.94
T1-P02.dat.tkl.2	190	718	990	37.88
T1-P02.dat.mult0.5_D1_d1	190	400	537	34.25
T1-P02.dat.mult0.5_D1_t1	190	406	543	33.74
T1-P02.dat.mult0.5_D2_d1	190	430	583	35.58
T1-P02.dat.mult0.5_D2_t0	190	419	559	33.41
T1-P02.dat.mult0.5_D2_t1	190	419	573	36.75
Déviation moyenne sur les instances P02				***37.59***
Déviation totale				***29.69***

Le Tableau 6-4 montre les résultats obtenus par ***MA_R2*** sur les instances d'Hurink et Knust en utilisant 3 robots identiques. Comme pour l'exécution à deux robots, pour les petites instances (*P01*), la déviation par rapport à la borne inférieure est de *3.8%*. Comme mentionné ci-dessus, la qualité des bornes inférieures n'est pas de bonne qualité. Pour les instances (*P02*) la déviation de notre algorithme est de *38.25%* et la déviation moyenne sur l'ensemble des instances est de *27.76%*.

Tableau 6-4 : Résultats sur les instances de job-shop avec trois robots identiques

		PL	**MA_R2**	
Instances	***N***	***LB***	***BFS^{long}***	***Dev%***
T1-P01.dat.D1_d1	66	60	62	3.33
T1-P01.dat.D1_t1	66	62	62*	0.00
T1-P01.dat.D2_d1	66	71	73	2.82
T1-P01.dat.D3_d1	66	83	88	6.02
T1-P01.dat.tkl.1	66	68	71	4.41
T1-P01.dat.T2_t1	66	58	61	5.17
T1-P01.dat.T3_t0	190	62	65	4.84
Déviation moyenne sur les instances P01				***3.80***
T1-P02.dat.D1_d1	190	686	987	43.88
T1-P02.dat.D1_t0	190	686	976	42.27
T1-P02.dat.D1_t1	190	692	983	42.05
T1-P02.dat.D2_d1	190	714	991	38.80
T1-P02.dat.D3_d1	190	738	1001	35.64
T1-P02.dat.D5_t2	190	785	1019	29.81
T1-P02.dat.T1_t1	190	669	960	43.50
T1-P02.dat.T2_t1	190	676	972	43.79
T1-P02.dat.T5_t2	190	701	987	40.80
T1-P02.dat.tkl.1	190	691	986	42.69
T1-P02.dat.tkl.2	190	702	986	40.46
T1-P02.dat.mult0.5_D1_d1	190	385	531	37.92
T1-P02.dat.mult0.5_D1_t1	190	391	524	34.02
T1-P02.dat.mult0.5_D2_d1	190	414	548	32.37
T1-P02.dat.mult0.5_D2_t0	190	414	545	31.64
T1-P02.dat.mult0.5_D2_t1	190	414	548	32.37
Déviation moyenne sur les instances P02				***38.25***
Déviation totale				***27.76***

L'étoile montre les solutions optimales trouvées par la modélisation linéaire

La difficulté de trouver de bonnes bornes inférieures pour les grandes instances, nous a poussé à choisir de calculer le gain engendré par l'ajout d'un nouveau robot au problème. Le Tableau 6-5 nous donne une vue récapitulative sur les conséquences d'ajout d'un robot. Les gains respectifs du passage de 1 à 2 robots, de 1 à 3 et de 3 à 3 robots sont : 13.89%, 6.26% et 3.32%.

Ces informations sont essentielles dans l'étape de dimensionnement de l'atelier. Elles peuvent nous aider à prendre une décision d'ajout d'un robot supplémentaire et ceci en faisant un

équilibrage entre le prix du robot et le gain engendré par cet achat.

Tableau 6-5 : Gain engendré par l'ajout d'un robot supplémentaire

	N	***BFS*** un robot	***BFS^{long}*** deux robots	***IMP 1-2***	***BFS^{long}*** trois robots	***IMP 1-3***	***IMP 2-3***
T1-P01.dat.D1_d1	66	87	63	-27.59	62	-28.74	-1.59
T1-P01.dat.D1_t1	66	81	62	-23.46	62	-23.46	0.00
T1-P01.dat.D2_d1	66	148	84	-43.24	73	-50.68	-13.10
T1-P01.dat.D3_d1	66	213	112	-47.42	88	-58.69	-21.43
T1-P01.dat.tkl.1	66	136	77	-43.38	71	-47.79	-7.79
T1-P01.dat.T2_t1	66	74	62	-16.22	61	-17.57	-1.61
T1-P01.dat.T3_t0	66	92	65	-29.35	65	-29.35	0.00
T1-P02.dat.D1_d1	190	1020	1009	-1.08	987	-3.24	-2.18
T1-P02.dat.D1_t0	190	991	982	-0.91	976	-1.51	-0.61
T1-P02.dat.D1_t1	190	1009	985	-2.38	983	-2.58	-0.20
T1-P02.dat.D2_d1	190	1056	991	-6.16	991	-6.16	0.00
T1-P02.dat.D3_d1	190	1089	1025	-5.88	1001	-8.08	-2.34
T1-P02.dat.D5_t2	190	1357	1029	-24.17	1019	-24.91	-0.97
T1-P02.dat.T1_t1	190	998	960	-3.81	960	-3.81	0.00
T1-P02.dat.T2_t1	190	1003	985	-1.79	972	-3.09	-1.32
T1-P02.dat.T5_t2	190	1041	1015	-2.50	987	-5.19	-2.76
T1-P02.dat.tkl.1	190	1022	1012	-0.98	986	-3.52	-2.57
T1-P02.dat.tkl.2	190	1004	990	-1.39	986	-1.79	-0.40
T1-P02.dat. mult0.5_D1_d1	190	586	537	-8.36	531	-9.39	-1.12
T1-P02.dat. mult0.5_D1_t1	190	557	543	-2.51	524	-5.92	-3.50
T1-P02.dat. mult0.5_D2_d1	190	676	583	-13.76	548	-18.93	-6.00
T1-P02.dat. mult0.5_D2_t0	190	593	559	-5.73	545	-8.09	-2.50
T1-P02.dat. mult0.5_D2_t1	190	619	573	-7.43	548	-11.47	-4.36
Amélioration moyenne				***-13.89***		***-16.26***	***-3.32***

6.2.2. Les instances de Bilge et Ulusoy (1995)

Le benchmark de Bilge et Ulusoy (1995) consiste en la définition de dix problèmes de job-shop, nommés EX1 à EX10 et l'utilisation de deux chariots identiques. Les temps de déplacement de ces derniers sont donnés par des matrices selon 4 topologies de disposition des machines dans l'atelier. Ainsi la combinaison de ces dix jobs et les quatre topologies fournit un benchmark de 40 instances. En plus de cela les auteurs définissent deux critères pour lesquels plusieurs résultats existent dans la littérature.

- Le critère **C1** : considère le dernier transport pour chaque job vers la station de déchargement (***U***) ;
- Le critère **C2** : ne considère pas le dernier transport pour chaque job vers la station de déchargement (***U***).

Ces deux critères multiplient par 2 le nombre d'instances. On signale que celles-ci sont alors nommées par $EXnm$, où, n représente le numéro de job et m celui de la topologie utilisée pour les déplacements des chariots, ainsi $EX103$ représente le problème 10 où les temps de déplacement des deux chariots sont donnés par la topologie 3.

Bilge et Ulusoy ont enrichi ces benchmarks par l'ajout d'un troisième chiffre qui peut être 0 ou 1. Si ce troisième chiffre vaut :

- **0** : les temps opératoire de toutes les opérations machines sont multipliés par **2**.
- **1** : les temps opératoires de toutes opérations machines sont multipliés par **3**.

On note que si ce troisième chiffre est employé alors, le temps de déplacement sont divisés par deux quelque soit la topologie utilisée. On note également, que l'emploi de ce troisième chiffre n'est utilisé que pour quelques instances, à savoir dont des résultats sont disponibles dans la littérature.

Les benchmarks de Bilge et Ulusoy ont une structure particulière, dans celle-ci chaque job commence par une opération de transport, alors que dans notre cas chaque job commence et termine par une opération machine. Cette situation nous amène à faire un choix entre :

- modifier notre modélisation pour qu'elle puisse prendre en compte ces instances ;
- adapter ces instances en ajoutant des opérations machine fictives de durée nulle au début et à la fin de chaque job.

Nous avons opté pour le deuxième choix. Cette adaptation augmente le nombre d'opérations dans le problème de $2*n$ où n est le nombre de jobs du problème. Il est important de noter que les tests ont montré que le fait d'ajouter des opérations de durées nulles n'affecte pas la complexité du problème.

Dans la littérature, pour les deux critères, plusieurs résultats existent donnés par différents auteurs. Pour chaque critère nous donnons les résultats trouvés par notre algorithme ***MA_R2***.

Nous tenons à rappeler que :

- C1: correspond à une minimisation du Makespan en prenant en compte la dernière opération de transport de chaque job vers la machine de déchargement U (unload station), ce critère est notamment étudié dans Deroussi et al., (2008)
- C2: correspond à une minimisation du Makespan en ne prenant pas en compte la dernière opération de transport de chaque job vers la machine de déchargement U (unload station); pour lequel plusieurs résultats existent, Bilge et Ulusoy (1993). Bilge et Ulusoy (1995)Bilge et al., (1997) Abdelmaguid et al., (2004).et Deroussi et al., (2008).

Le critère *C1* génère plus de complexité que le critère *C2,* puisque ce dernier ne considère pas les dernières opérations de transport. De plus, dans le benchmark de Bilge et Ulusoy, on trouve des instances de Flow-Shop telle que le problème EX8.

Notre approche est plus générale que celle proposée par Bilge et Ulusoy, elle n'impose aucune limite pour le nombre de robots et ces derniers peuvent être différents, et leurs déplacements à vide ou à charge peuvent l'être aussi.

Le critère *C1* est étudié uniquement par Deroussi et al., (2008). Dans cet article les auteurs proposent des résultats obtenus par trois approches différentes qui sont : la recherche locale itérée (***I***terative ***L***ocale ***S***earche (***ILS***)), le recuit simulé (***S***imulated ***A***nnealing (***SA***)) et le recuit simulé avec une recherche local (***S***imulated ***A***nnealing ***L***ocal ***S***earch (***SALS***)). Ces résultats sont comparés à ceux trouvés par ***MA_R2***. La déviation est calculée par rapport aux bornes inférieures trouvées par notre modélisation linéaire après quinze minutes de calcul.

Tableau 6-6 : Résultats sur les instances de Bilge et Ulusoy avec deux chariots critère C1

Instances	**PL**	**Framework**		Deroussi et al., (2008)					
	LB	***BFS^{10min}***	***Dev1***	***ILS***	***Dev1***	***SA***	***Dev1***	***SALS***	***Dev1***
EX11	87	114	31.03	114	31.03	114	31.03	114	31.03
EX21	80	116	45.00	118	47.50	118	47.50	116	45.00
EX31	94	121	28.72	121	28.72	121	28.72	121	28.72
EX41	87	138	58.62	138	58.62	138	58.62	138	58.62
EX51	74	110	48.65	110	48.65	110	48.65	110	48.65
EX61	97	129	32.99	130	34.02	129	32.99	129	32.99
EX71	78	**133**	70.51	134	71.79	137	75.64	135	73.08
EX81	167	167*	0.00	167	0.00	167	0.00	167	0.00
EX91	106	**127**	19.81	129	21.70	129	21.70	129	21.70
EX101	121	153	26.45	156	28.93	153	26.45	153	26.45
EX12	84	90	7.14	90	7.14	90	7.14	90	7.14
EX22	81	82	1.23	82	1.23	82	1.23	82	1.23
EX32	84	89	5.95	89	5.95	89	5.95	89	5.95
EX42	77	100	29.87	102	32.47	100	29.87	100	29.87
EX52	72	81	12.50	81	12.50	83	15.28	81	12.50
EX62	100	102	2.00	102	2.00	102	2.00	102	2.00
EX72	73	86	17.81	90	23.29	86	17.81	86	17.81
EX82	126	155	23.02	155	23.02	155	23.02	155	23.02
EX92	102	106	3.92	106	3.92	106	3.92	106	3.92
EX102	115	139	20.87	140	21.74	139	20.87	139	20.87
EX13	80	98	22.50	98	22.50	98	22.50	98	22.50
EX23	76	89	17.11	90	18.42	90	18.42	89	17.11
EX33	84	96	14.29	96	14.29	96	14.29	96	14.29
EX43	77	102	32.47	106	37.66	104	35.06	102	32.47
EX53	70	89	27.14	89	27.14	89	27.14	89	27.14
EX63	100	105	5.00	106	6.00	105	5.00	105	5.00
EX73	71	93	30.99	98	38.03	93	30.99	93	30.99
EX83	155	155*	0.00	155	0.00	155	0.00	155	0.00
EX93	106	107	0.94	107	0.94	107	0.94	107	0.94
EX103	110	139	26.36	141	28.18	139	26.36	139	26.36
EX14	97	140	44.33	140	44.33	140	44.33	140	44.33
EX24	90	134	48.89	138	53.33	136	51.11	134	48.89
EX34	102	148	45.10	149	46.08	149	46.08	148	45.10
EX44	96	163	69.79	164	70.83	163	69.79	163	69.79
EX54	82	134	63.41	134	63.41	136	65.85	134	63.41
EX64	99	151	52.53	151	52.53	151	52.53	153	54.55
EX74	80	162	102.50	167	108.75	163	103.75	161	101.25
EX84	128	178	39.06	178	39.06	178	39.06	178	39.06
EX94	108	149	37.96	149	37.96	150	38.89	149	37.96
EX104	127	183	44.09	187	47.24	184	44.88	183	44.09
Deviation totale moyenne			***30.26***		***31.52***		***30.88***		***30.39***

L'étoile montre les solutions optimales trouvées par la modélisation linéaire

Les résultats du Tableau 6-6 montrent que ***MA_R2*** donne une déviation totale de 30.39%. Celle-ci est meilleure que celles fournies par toutes les approches concurrentes. De plus notre algorithme donne des résultats identique ou meilleurs (18/40 pour(***ILS***)), (12/40 pour (***SA***)) et (3/40 pour (***SALS***)). Il fournit également deux nouvelles meilleures solutions pour les instances EX71 et EX91.

Le critère *C2* est le plus utilisé dans la littérature. Ils existent onze méthodes pour lesquelles des résultats sont disponibles. Le Tableau 6-7 donne une comparaison des résultats trouvés par notre algorithme avec les résultats de ces onze méthodes. Pour chaque instance on donne la borne inférieure qui servira au calcul de la déviation, cette borne étant trouvée par notre ***PL*** en 15 minutes de calcul.

Tableau 6-7 : Résultats du test sur les instances de Bilge et Ulusoy avec deux chariots et l'utilisation du critère C2

	PL	Bilge et Ulosoy, (1995)										Bilge et al., (1997)		Abdelmaguid et al.2004		Deroussi et al. (2008)						**MA_R2**	
	LB[2]	BSPH	Dev1	ALG1	Dev1	ALG2	Dev1	R1MM	Dev1	STW	DEV1	UGA	Dev1	GA	Dev1	ILS	Dev1	SA	Dev1	SALS	Dev1	*BFS^{10min}*	Dev1
EX11	**96**	108	12.50	101	5.21	96	0.00	96	0.00	96	0.00	96	0.00	96	0.00	96	0.00	96	0.00	96	0.00	96*	0.00
EX21	**79**	108	36.71	114	44.30	106	34.18	105	32.91	105	32.91	104	31.65	102	29.11	100	26.58	100	26.58	100	26.58	100	26.58
EX31	**90**	125	38.89	105	16.67	107	18.89	105	16.67	105	16.67	105	16.67	99	10.00	99	10.00	99	10.00	99	10.00	99	10.00
EX41	**78**	143	83.33	125	60.26	118	51.28	118	51.28	118	51.28	116	48.72	112	43.59	112	43.59	112	43.59	112	43.59	112	43.59
EX51	**87**	111	27.59	89	2.30	90	3.45	90	3.45	87	0.00	87	0.00	87	0.00	87	0.00	87	0.00	87	0.00	87*	0.00
EX61	**107**	128	19.63	137	28.04	127	18.69	120	12.15	120	12.15	121	13.08	118	10.28	118	10.28	118	10.28	118	10.28	118	10.28
EX71	**76**	133	75.00	119	56.58	123	61.84	122	60.53	119	56.58	118	55.26	115	51.32	111	46.05	111	46.05	111	46.05	111	46.05
EX81	**161**	165	2.48	189	17.39	161	0.00	164	1.86	161	0.00	161	0.00	161	0.00	161	0.00	161	0.00	161	0.00	161*	0.00
EX91	**115**	130	13.04	125	8.70	123	6.96	120	4.35	120	4.35	117	1.74	118	2.61	116	0.87	116	0.87	116	0.87	116	0.87
EX101	**116**	156	34.48	169	45.69	153	31.90	155	33.62	153	31.90	150	29.31	147	26.72	148	27.59	147	26.72	147	26.72	146	25.86
EX12	**82**	85	3.66	83	1.22	82	0.00	82	0.00	82	0.00	82	0.00	82	0.00	82	0.00	82	0.00	82	0.00	82*	0.00
EX22	**76**	80	5.26	86	13.16	80	5.26	80	5.26	80	5.26	76	0.00	76	0.00	76	0.00	76	0.00	76	0.00	76*	0.00
EX32	**84**	90	7.14	88	4.76	88	4.76	88	4.76	88	4.76	85	1.19	85	1.19	85	1.19	85	1.19	85	1.19	85	1.19
EX42	**70**	103	47.14	94	34.29	93	32.86	93	32.86	93	32.86	88	25.71	88	25.71	88	25.71	88	25.71	87	24.29	87	24.29
EX52	**69**	88	27.54	69	0.00	72	4.35	69	0.00	69	0.00	69	0.00	69	0.00	69	0.00	69	0.00	69	0.00	69*	0.00
EX62	**97**	102	5.15	113	16.49	100	3.09	100	3.09	100	3.09	98	1.03	98	1.03	98	1.03	98	1.03	98	1.03	98	1.03
EX72	**65**	90	38.46	90	38.46	91	40.00	90	38.46	90	38.46	85	30.77	79	21.54	79	21.54	79	21.54	79	21.54	79	21.54
EX82	**151**	151	0.00	158	4.64	151	0.00	151	0.00	151	0.00	151	0.00	151	0.00	151	0.00	151	0.00	151	0.00	151*	0.00
EX92	**102**	112	9.80	110	7.84	107	4.90	104	1.96	104	1.96	102	0.00	104	1.96	102	0.00	102	0.00	102	0.00	102*	0.00
EX102	**131**	143	9.16	149	13.74	139	6.11	144	9.92	139	6.11	137	4.58	136	3.82	135	3.05	135	3.05	135	3.05	135	3.05
EX13	**84**	94	11.90	84	0.00	86	2.38	84	0.00	84	0.00	84	0.00	84	0.00	84	0.00	84	0.00	84	0.00	84*	0.00
EX23	**82**	90	9.76	92	12.20	90	9.76	86	4.88	86	4.88	86	4.88	86	4.88	86	4.88	86	4.88	86	4.88	86	4.88
EX33	**86**	94	9.30	86	0.00	86	0.00	86	0.00	86	0.00	86	0.00	86	0.00	86	0.00	86	0.00	86	0.00	86*	0.00
EX43	**71**	108	52.11	95	33.80	100	40.85	95	33.80	95	33.80	91	28.17	89	25.35	89	25.35	89	25.35	89	25.35	89	25.35
EX53	**70**	86	22.86	77	10.00	76	8.57	83	18.57	76	8.57	75	7.14	74	5.71	74	5.71	74	5.71	74	5.71	74	5.71
EX63	**100**	112	12.00	116	16.00	105	5.00	104	4.00	104	4.00	104	4.00	104	4.00	103	3.00	103	3.00	103	3.00	103	3.00
EX73	**64**	104	62.50	98	53.13	100	56.25	91	42.19	91	42.19	88	37.50	86	34.38	84	31.25	83	29.69	83	29.69	83	29.69
EX83	**153**	153	0.00	164	7.19	153	0.00	153	0.00	153	0.00	153	0.00	153	0.00	153	0.00	153	0.00	153	0.00	153*	0.00
EX93	**102**	118	15.69	113	10.78	113	10.78	110	7.84	110	7.84	105	2.94	106	3.92	105	2.94	105	2.94	105	2.94	105	2.94
EX103	**119**	148	24.37	155	30.25	150	26.05	143	20.17	143	20.17	143	20.17	141	18.49	139	16.81	139	16.81	138	15.97	137	15.13
EX14	**94**	144	53.19	109	15.96	108	14.89	108	14.89	108	14.89	103	9.57	103	9.57	103	9.57	103	9.57	103	9.57	103	9.57
EX24	**75**	131	74.67	124	65.33	117	56.00	116	54.67	116	54.67	113	50.67	108	44.00	108	44.00	108	44.00	108	44.00	108	44.00
EX34	**91**	130	42.86	116	27.47	118	29.67	116	27.47	116	27.47	113	24.18	111	21.98	111	21.98	111	21.98	111	21.98	111	21.98
EX44	**80**	163	103.75	134	67.50	126	57.50	126	57.50	126	57.50	126	57.50	126	57.50	121	51.25	125	56.25	121	51.25	121	51.25
EX54	**85**	129	51.76	99	16.47	103	21.18	99	16.47	99	16.47	97	14.12	96	12.94	96	12.94	96	12.94	96	12.94	96	12.94
EX64	**88**	154	75.00	138	56.82	129	46.59	120	36.36	120	36.36	123	39.77	120	36.36	120	36.36	120	36.36	120	36.36	120	36.36
EX74	**69**	159	130.43	136	97.10	150	117.39	138	100.00	136	97.10	128	85.51	127	84.06	126	82.61	126	82.61	126	82.61	126	82.61
EX84	**163**	193	18.40	205	25.77	172	5.52	163	0.00	163	0.00	163	0.00	163	0.00	163	0.00	163	0.00	163	0.00	163*	0.00
EX94	**102**	164	60.78	133	30.39	129	26.47	125	22.55	125	22.55	123	20.59	122	19.61	120	17.65	120	17.65	120	17.65	120	17.65
EX104	**119**	173	45.38	182	52.94	174	46.22	171	43.70	171	43.70	164	37.82	159	33.61	160	34.45	159	33.61	159	33.61	157	31.93
Deviation totale moyenne		***34.34***		***26.22***		***22.74***		***20.46***		***19.76***			***17.61***		***16.13***		***15.46***		***15.50***		***15.32***		***15.23***

L'étoile montre les solutions optimales prouvées par la modélisation linéaire

Le Tableau 6-8 , donne un aperçu général sur les performances de toutes les méthodes y compris notre algorithme. Dans ce tableau on s'intéresse à la déviation moyenne totale par rapport à la borne inférieure et à la déviation totale par rapport à la meilleure solution trouvée par toutes les méthodes. La dernière colonne donne le nombre de solutions améliorées par notre algorithme par rapport à chaque méthode de la littérature.

Tableau 6-8 : Performances des méthodes utilisant le critère C2 sur les instances de Bilges et Ulusoy avec 2 chariots.

Source	Method	Déviation moyenne totale par rapport à la borne inférieure	Déviation moyenne totale par rapport à la meilleure solution connue	Nombre de solutions améliorées par MA_R2 par rapport à chaque méthode pour les 40 instances
Bilge et Ulusoy (1995)	BSH:	34.34	17.58	38
	ALG1:	26.22	9.31	37
	ALG2:	22.74	6.01	34
	R1MM:	20.46	4.21	30
	STW:	19.76	3.61	28
Bilge et al., (1997)	UGA	17.61	1.85	23
Abdelmaguid et al., (2004)	GA	16.13	0.70	15
Deroussi et al., (2008)	ILS:	15.46	0.18	5
	SA:	15.50	0.20	5
	SALS	15.32	0.07	3
MA_R2	MA	15.23	0.00	-

La synthèse de ces résultats montre la supériorité de l'algorithme *MA_R2*. Ce dernier donne des résultats pareils sinon meilleurs que ceux de toutes les autres approches, en plus il a permis d'obtenir de nouvelles meilleures solutions pour 3 instances. *MA_R2* améliore 23 solutions de*(UGA)*, 15 *(GA), 5 (ILS,SA)* et 3 pour *(SALS)*.

6.3. Conclusion sur l'approche proposée

Notre approche est plus générale, elle permet de traiter des problèmes de job-shop avec plusieurs robots différents qui peuvent avoir des temps de déplacements à vide et à charge différents. Les tests montrent aussi que l'algorithme MA_R2 surpasse toutes les méthodes de la littérature que se soit les algorithmes dédiés pour le cas d'un seul robot ou l'utilisation de deux robots. Les instances testées varient de 25 opérations machines pour la plus petite à 100 opérations machines pour les plus grandes. Le Tableau 6-9 nous donne une synthèse des performances de notre approche par rapport aux approches existantes dans la littérature. En se servant des bornes inférieures trouvées par notre ***PL***, nous avons calculé des déviations par rapport à ces bornes, qui donnent l'avantage à notre approche. Nous avons aussi calculé une déviation par rapport aux meilleurs solutions obtenues par toutes les méthodes, y compris la notre et le résultat de cette comparaison est à l'avantage de notre algorithme qui a aussi le mérite d'introduire de nouvelles meilleures solutions pour différents problèmes, à un robot ou à plusieurs, et de prouver que certaines de ces solutions sont optimales en se servant des bornes inférieures trouvées par le PL.

Sur les moyennes instances (P01) de d'Hurink et Knust avec l'utilisation d'un seul robot,

MA_R1 fournit toujours les meilleures solutions. Pour les grandes instances *(P02)* avec l'utilisation d'un seul robot minutes (BFS^{30min}) (BFS^{long}) donnent une déviation totale moyenne respectivement de 1.63 , 1.49 par rapport à la *BKS* , pendant que la meilleure approche d'Hurink et Knust (combined approach) UB^{lg} donne 2.47.

Pour les instances à 1 robot *(P01)* deux solutions sont trouvées et d'autres sont confirmées alors que et 9 nouvelles meilleures solutions sont trouvées pour les grandes instances (P02). En ce qui concerne les instances avec deux robots de Bilge et Ulusoy, 5 nouvelles meilleures solutions sont introduite par MA_R2. Les bornes inférieures fournies par notre PL prouvent que 18 solutions trouvées par notre algorithme sont optimales.

Le Tableau 6-9 montre que MA_R1 est plus général que les autres approches car il est capable de traiter les instances à 1 et plusieurs robots et les résultats qu'il fourni lui donnent l'avantage.

Tableau 6-9 : Performances de MA_R2

	Instances à 1 robot				Instances à 2 robots	
	Instance de Hurink Knust Hurink et Knust (2005)7 moyenne instances *(P01)*	Instance de Hurink et Knust (2005) 16 grandes instances *(P02)*			Instance de Bilge et Ulusoy (1995) 40 instances moyennes et critère *C1*	Instance de Bilge et Ulusoy (1995) 40 instances moyennes et critère *C2*
		BKS^{10min}	BKS^{30min}	BKS^{long}		
Déviation total moyenne par rapport à la *LB*	2.31	17.06	16.09	15.93	30.26	15.23
Déviation total moyenne par rapport à la *BKS*	0.00	2.47	1.63	1.49	0.00	0.00
Nombre de solutions optimales	5	-	-	-	2	11
Nombre de nouvelles meilleures solutions	2	3	4	5	2	3

Avant de choisir l'algorithme génétique comme méta-heuristique retenue pour la résolution du job-shop avec plusieurs robots, nous en avons exploré d'autres telles que : le kangourou, la recherche tabou, le recuit simulé et le Grasp. Les tests réalisés avec cette dernière méta-heuristique ont donné des résultats proches de ceux de l'algorithme MA_R2.

7. Conclusion

Dans ce chapitre nous avons essayé d'étendre les modèles existants du job-shop pour inclure les contraintes liées au transport des jobs entre les différentes machines. Nous sommes passés par une phase de compréhension de problème et l'étude de l'état de l'art du job-shop, puis nos efforts se sont orientés vers une extension du job-shop pour lui inclure le transport. Cette extension n'est jamais abordée dans la littérature dans le cas général, aucune modélisation n'est disponible pour le job-shop avec n robots différents. Se basant sur les travaux de Knust (1999) et Hurink et Knust (2002)Hurink et Knust (2005), nous avons étendu le graphe disjonctif pour l'adapter à la présence de plusieurs robots Lacomme et al., (2007), par la suite nous avons proposé une modélisation linéaire du problème du job-shop, et plusieurs heuristiques de construction de solutions. Nous avons aussi proposé des méta-heuristiques pour résoudre le job-shop avec transport avec plusieurs robots, ces méta-heuristiques sont les algorithmes mémétiques et le Grasp. Une large série de test est réalisée pour évaluer les performances de nos approches et les résultats obtenus sont meilleurs que ceux de la littérature pour tous les benchmarks testés.

Ces modélisations et les approches de résolution constituent un outil d'aide à la décision

pour établir une stratégie de conception ou d'évolutions des ateliers de production nécessitants des moyens de transport. En effet, connaître le gain apportté par l'ajout d'un robot à une flotte constituée initialement de ***n*** robots constitue une information de taille pour le chargé de conception de l'atelier, car l'ajout d'un robot supplémentaire engendre des coûts d'acquisition de maintenance et de consommation d'énergie etc. Si le gain est supérieur aux coûts alors, l'ajout est profitable à l'entreprise, dans le cas contraire, l'ajout est systématiquement écarté.

Dans notre analyse, et les différents tests que nous avons réalisés, il en ressort que minimiser le makespan revient à une gestion minutieuse des déplacements des robots. En effet les déplacements à charge sont inévitables, ce qui n'est pas le cas des déplacements à vide, minimiser ces déplacements conduits dans la plupart des cas à minimiser le Makespan. Nous avons remarqué, que si le robot possède une capacité supérieure à 1 il peut facilement minimiser certains déplacements et ainsi bien optimiser ces trajets. Ce problème est plus complexe que celui du job-shop avec des robots de capacité unitaire, il sera traité dans le chapitre suivant.

.

Chapitre IV : Job-shop avec transport avec des robots de capacité non unitaire

Ce chapitre est consacré à la présentation et à la résolution du problème de job-shop avec transport et chariots à capacité non unitaire.

Liste des tableaux

Liste des figures

Liste des algorithmes

.

1. Introduction

Nous avons vu que, dans le cas du job-shop avec transport utilisant un seul robot, pour réduire la date finale d'exécution (makespan) il fallait réduire au maximum les déplacements à vide. Afin d'augmenter la performance du système, il est possible d'utiliser plusieurs robots comme nous l'avons montré dans le chapitre précédent. Cette réponse se heurte à des considérations économiques et aux contraintes liées à la conception même de l'atelier (volume, nombre maximal de robots, gestion des collisions etc.). Une autre solution consiste à augmenter la capacité du robot (nombre de jobs que peut transporter le robot au même moment). Cette solution permet d'envisager le déplacement de plusieurs pièces simultanément.

Le but de se chapitre est d'étudier le job-shop avec transport avec un seul robot possédant une capacité supérieur à 1, ce qui signifie que le robot peut transporter plusieurs jobs à la fois. Dans ce chapitre, est réalisé un état de l'art de cette extension. A notre connaissance, très peu de publications abordent ce problème. Toutefois, il est possible de citer un article de Abdelmaguid et Nassef (2010) qui traite du job-shop avec plusieurs robots de capacité non unitaire et nos publications qui datent de 2009 et 2010 Lacomme et al., (2009b), Lacomme et al., (2010) et Lacomme et al., (2009a).

L'étude de problème d'ordonnancement avec robot de capacité non unitaire revêt un intérêt stratégique dans la conception des ateliers de production, où on cherche à répondre à la question : ***vaut-il mieux utiliser un seul robot de capacité k>1 ou plusieurs robots de capacité unitaire ?***

Avoir la réponse à cette question nous aide à dimensionner l'atelier ou à le faire évoluer. Comme au précédent chapitre, celui-ci commence par une définition formelle du problème puis une formalisation linéaire du problème est donnée et enfin une métaheuristique permettant de résoudre le problème est mise en œuvre.

2. Définition du job-shop avec un robot de capacité non unitaire

Le job-shop avec un robot de capacité non unitaire consiste en l'utilisation dans un atelier de type job-shop d'un robot disposant d'une capacité de chargement supérieure à 1. Il convient de signaler que dans plusieurs ateliers de ce type les robots peuvent transporter plusieurs pièces et ainsi de réduire les déplacements à vide. Ces robots sont souvent appeler multi-purpose robots l'équivalent en français de robot-multi-objectifs.

Le problème du job-shop avec transport consiste à ordonnancer un ensemble de n jobs $J=\{J_1,J_2,..J_n\}$ sur un ensemble de m machines $M=\{M_1,M_2,..M_m\}$. Les jobs sont transportés par un seul robot de capacité limitée et supérieure à 1. Chaque job J_i est composé de n_i opérations notées $O_{i,1},O_{i,2},\ldots,O_{i,n_i}$ pour lesquelles on définit des contraintes de précédence. Chaque opération O_{ij} du job J_i doit s'exécuter sur la machine $\mu_{i,j}$ sans préemption pendant une durée p_{ij}. Chaque machine ne peut exécuter qu'une seule opération à la fois. De même, chaque job ne peut s'exécuter que sur une et une seule machine à la fois. Les temps de transport doivent être pris en compte à chaque fois qu'un job change de machine pour réaliser sa prochaine opération machine. Le transport entre deux machines est réalisé par un robot de capacité supérieure à un. Le robot peut charger des jobs tant que sa capacité n'est pas atteinte (les jobs peuvent être déchargés dans un ordre différent de celui de leur chargement). Le temps $t^k_{\mu_i,\mu_j}$ est considéré à chaque fois que le

robot se déplace de la machine μ_i vers la machine μ_j en transportant k jobs, $k \in [1..K]$. On considère que chaque machine dispose des buffers (stocks) d'entrée/sortie illimités. Les jobs attendent dans les buffers de sortie des machines la disponibilité du robot pour les transporter vers les machines suivantes, de la même manière les jobs déposés sur les machines doivent attendre dans les buffers d'entrée de ces dernières la disponibilité de la machine pour être exécutés. On note que l'ordre d'arrivée des jobs peut différer de l'ordre de leur exécution par les machines. Les temps de transport satisfont l'inégalité triangulaire et les temps de chargement (respectivement déchargement) S_i sont considérés pour chaque opération impliquant un chargement ou un déchargement. L'objectif est de trouver un ordonnancement faisable qui minimise le makespan $C_{\max} = \max\{C_i\}$ où C_i représente la fin d'exécution de la dernière opération du job i.

2.1. Modélisation du problème sous la forme d'un Graphe disjonctif

Dans cette partie, on propose une extension, du graphe disjonctif proposé pour le job-shop avec plusieurs robots. Le but est de fournir une modélisation capable de représenter les changements de la charge du robot ainsi que de respecter les contraintes liées à la capacité non unitaire de celui-ci.

Le graphe disjonctif que nous proposons est une extension du graphe disjonctif utilisé pour le job-shop avec transport. Dans ce graphe les sommets modélisent d'une part, les opérations machines, et d'autre part les opérations liées aux chargements et aux déchargements des jobs. Les arrêtes sont de trois types :

- les arrêtes conjonctives qui relient les différentes opérations de chaque job et qui expriment les contraintes de précédences ;
- les arrêtes disjonctives machine qui expriment l'ordre de passage des jobs sur les différentes machines ;
- les arêtes disjonctives transport qui définissent l'ordre des opérations transport.

Pour gérer la capacité du robot on propose la décomposition de l'opération de transport en deux opérations qui vont nous permettre en suite d'intégrer le fait que le robot garde un job et va charger un autre puis les décharger dans un autre ordre suivant les solutions. Donc chaque opération de transport se voit décomposé en :

- une opération de chargement ;
- une opération de déchargement.

Les opérations de chargement et de déchargement sont modélisées chacune par un sommet dans le graphe conjonctif-disjonctif. Le graphe disjonctif $G = (SM \cup ST, C \cup DM \cup DT)$ est une représentation du problème où :

- SM est l'ensemble des sommets machine plus deux sommets additionnels, l'un au début du graphe $\{0\}$ et l'autre à la fin du graphe $\{*\}$.
- ST est l'ensemble des sommets transport contenant à la fois les opérations de chargement et les opérations de déchargement ;
- DM : est l'ensemble des disjonctions machine ;
- DT : est l'ensemble des disjonctions transport ;
- C : est l'ensemble des conjonctions représentant les contraintes de précédence à l'intérieur du même job. L'opération de chargement successeur de l'opération $O_{i,j}$ est

notée $l_{i,j}$ et l'opération de déchargement suivante est notée $u_{i,j}$ les conjonctions sont alors comme suit : $O_{i,j} \rightarrow l_{i,j} \rightarrow u_{i,j} \rightarrow O_{i,j+1}$

Une solution du problème consiste à définir les dates de début de toutes les opérations machines et transports en définissant :

- les disjonctions machines qui décrivent l'ordre de passage des jobs sur les machines;
- les disjonctions transports qui décrivent les déplacements du robot.

Considérons l'exemple *JS4* suivant composé de deux jobs et de trois machines.

Tableau 2-1 : Instance *JS4* du job-shop

Jobs\OP	**Op1**	**Op2'**	**Op3**
Job1	(M1,10)	(M2,5)	(M3,10)
Job2	(M4,15)	(M1,10)	(M2,20)

Tableau 2-2 : Temps de déplacement du robot en transportant 0,1 ou 2 jobs

	M1	M2'	M3	M4
M1	0	2	2	2
M2	2	0	2	2
M3	2	2	0	2
M4	2	2	2	0

Une solution de l'exemple précédant est donnée par la Figure 2-1 sous la forme d'un graphe disjonctif orienté qui définit l'ordre de passage des jobs sur les différentes machines et l'ordre des opérations transport.

Les ordres de passage des jobs sur les différentes machines sont les suivants :

- Machine 1 : Job 1, Job 2 ;
- Machine 2 : Job 1, Job 2 ;
- Machine 3 : Job 1 ;
- Machine 4. Job 2.

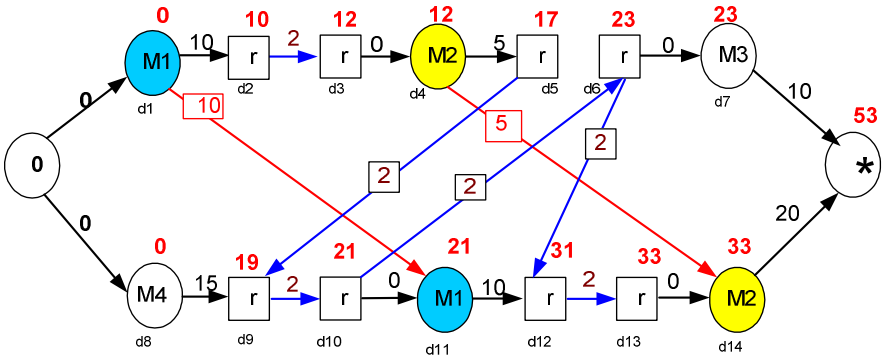


Figure 2-1 : Représentation avec séparation de l'opération transport

La première opération de transport est le chargement du job 1 sur la machine M1 : $l_{1,1}$. Cette opération est suivie de $u_{1,1}$ qui représente le déchargement du job 1 sur la machine M2. Le robot charge ensuite, à la date 17, le job 1 qui termine son traitement sur la machine M2. Puis chargé de la pièce 1 il se déplace vers la machine M4 sur laquelle il arrive à la date 19 ; le job 2 est donc chargé sur le robot à la date 19. Le robot transportant deux jobs simultanément, commence son déplacement de M4 vers M1 à la date 19 et termine le déplacement à la date 21. A la date 21 le job 2 commence son traitement sur la machine M1 et le robot se déplace en transportant uniquement le job 1 de la machine M1 vers la machine M3.

La Figure 2-1 donne un exemple de graphe disjonctif orienté et valué avec toutes les dates de début des opérations. On peut visualiser cette solution sous la forme d'un diagramme de gant en modélisant les opérations transport sous la forme de flèches reliant les opérations machine. Ce type de représentation est utilisé sur la Figure 2-2.

Le robot est une ressource tout comme les machines. Une représentation naturelle consiste à faire apparaitre le robot sur l'axe des ordonnées avec les machines. Sur la Figure 2-2, les opérations de transport, sont identifiées par deux chiffres x-y où x représente le nombre de pièces transport au début de l'opération de transport et y le nombre de pièces transportées à la fin de l'opération de transport.

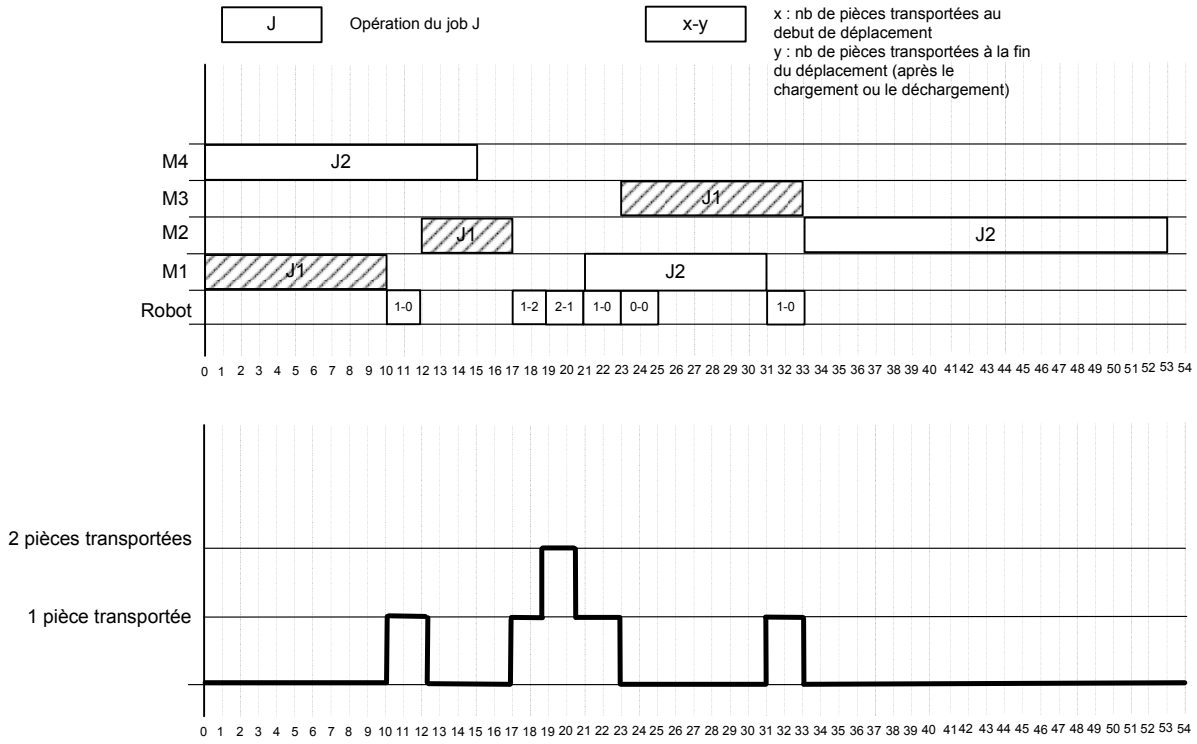


Figure 2-2 : Diagramme de Gantt avec le profil de charge du robot représentant la solution donnée par la Figure 2-1

L'avantage de cette deuxième représentation est de permettre une construction facile du profil de charge du robot, c'est-à-dire d'une courbe montrant l'évolution du nombre de pièces transportés au cours du temps.

2.2. Modélisation linéaire du problème

Dans cette section nous proposons une modélisation basée sur la notion de flot pour représenté la charge du robot au cours du temps. D

Les opérations machines sont des opérations classiques du job-shop. Les opérations transports vont être traitées en considérant que le nombre de place libre sur le robot est un problème de flot.

2.2.1. Notations

J : ensemble des jobs $J = \{1;2;...;n\}$

M : ensemble des machines $M = \{1;2;...;m\}$

O : ensemble des opérations machines avec o_i^k la $k^{ième}$ opération machine du job i

O' : ensemble des opérations machines à l'exception des opérations qui sont les dernières opérations d'un job

O^f ensemble des opérations machines qui sont les dernières opérations d'un job $O^f = O - O'$

P : ensemble des opérations machine qui sont les premières dans la gamme tel que O_{i1} c'est la première opération machine du job i $(\forall i \in P, (i-1)$, n'existe pas)

T : ensemble des opérations de transport où t_{ij}^c représente la durée du trajet entre la machine

i et la machine j, du robot transportant c jobs

L : ensemble des opérations de chargement (sortie)

L_i : ensemble des opérations de chargement du job i, $L_i \in L = \bigcup_{i=1}^{n} L_i$ tel que l_i^j représente la j^{ieme} opération de chargement du job i

U : ensemble des opérations de déchargement (entrée)

U_i : ensemble des opérations de déchargement du job i et $U_i \in U = \bigcup_{i=1}^{n} U_i$ tel que u_i^j représente la j^{ieme} opération de déchargement du job i

E : ensemble des opérations de chargement et de déchargement $E = L \cup U$

S : ensemble de toutes les opérations du problème $S = O \cup E$

N_i : le nombre d'opérations machine du job i, auquel correspondent $(N_i - 1)$ opérations de chargement et $(N_i - 1)$ opérations de déchargement.

K : la capacité maximum du robot

r_0 : Le nombre de places disponibles sur le robot au début du problème plus une, on a donc $r_0 = K + 1$

y_i : job de l'opération i

μ_i : machine utilisée par l'opération machine $i \in O$

p_i : temps opératoire de l'opération machine $i \in O$

$T_{i,j}^c$: la durée du trajet du robot entre la machine μ_i et la machine μ_j transportant c jobs

$(i-1)$: le prédécesseur de l'opération i dans le même job

$(i+1)$: successeur de l'opération i dans le même job

(0) : le premier sommet dans le graphe, il est prédécesseur de toutes les opérations machines premières dans leur gamme et prédécesseur aussi de la première opération de chargement

$(*)$: le sommet final dans le graphe, il est successeur des dernières opérations machines dans leurs gammes et successeur de la dernière opération de déchargement

H : grand entier positif

ε très petit nombre, $\varepsilon \in]0,1[$ permettant de rendre une inéquation de type $a \prec b$ en $a \leq b\text{-}\varepsilon$ ou de type $a \succ b$ en $a \geq b + \varepsilon$

2.2.2. Variables:

t_i : date de début de traitement de l'opération $i \in S$

$$b_{ij} = \begin{cases} 1 & \text{si l'opération machine } i \text{ est exécutée avant l'opération machine } j \\ 0 & \text{sinon} \end{cases}$$

$$\varphi_{i,j} = \begin{cases} k & \text{si le } \textit{flot entre l'opération } i \in E \text{ et l'opération } j \in E \text{ vaut k} \in [0, K] \\ 0 & \text{sinon} \end{cases}$$

$$y_{i,j}^k = \begin{cases} 1 & \text{si il y a flot entre l'opération } i \in E \text{ et l'opération } j \in E \text{, flot de valeur k} \\ 0 & \text{sinon} \end{cases}$$

S_i : temps nécessaire pour le robot de charger ou de décharger l'opération $i \in E$

Toutes les opérations de S sont rangées d'une manière croissante. Pour accéder à un élément de S, on définit une fonction λ qui renvoi l'indice d'une opération dans S d'un job $i \in J$ et d'un rang k. Ainsi :

- l'indice de l'opération machine o_i^k est donné par $\lambda(o_i^k) = p$;
- l'indice de l'opération de sortie l_i^k (chargement) est donné par $\lambda(l_i^k) = p+1$;
- l'indice de l'opération d'entrée u_i^k (déchargement) est donné par $\lambda(u_i^k) = p+2$.

La Figure 2-3 montre les relations entre deux opérations machines successives d'un même job en incluant l'opération de transport. Chaque job est définit par une séquence d'opération machine et à chaque opération machine o_i est associée une opération de chargement l_i et une opération de déchargement u_i à l'exception de la dernière. Ainsi, le nombre d'opération de chargement et de déchargement est égal à $(n-1)$ pour (n) opérations machine.

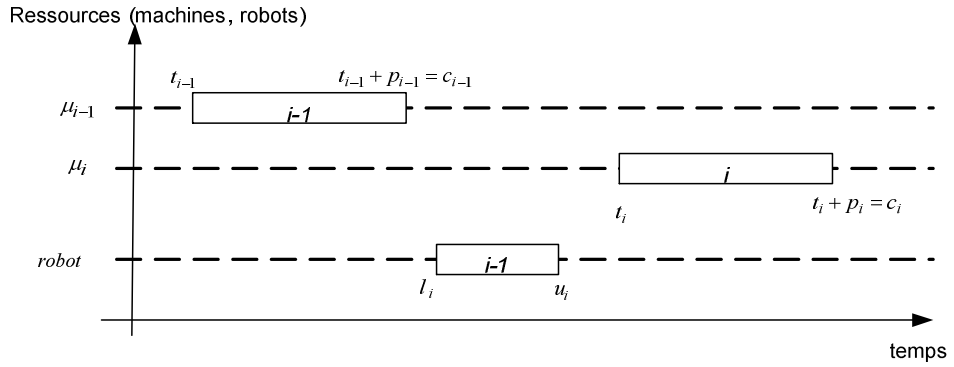


Figure 2-3 : Variables liées au problème

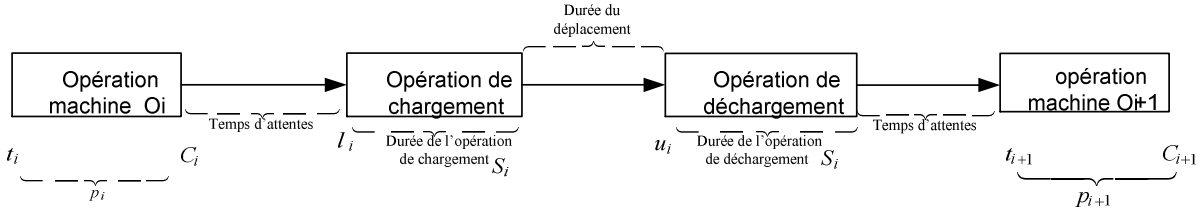


Figure 2-4 : Relations entre les variables entre liées au problème

2.2.3. Contraintes pour les conjonctions et les disjonctions

Contrainte conjonctives entre les opérations du même job.

Date de début des opérations de chargement : Le début de l'opération de chargement ne peut commencer qu'une fois le traitement du job sur la machine terminé. Cette contrainte assure que la date à laquelle la pièce est chargée sur le robot est supérieure ou égale à la date de début t_i plus le temps opératoire p_i sur la machine m, (cf. contrainte (1.1))

$$t_{i+1} \geq t_i + p_i + S_i \qquad \forall j \in J, \forall k \leq n_j - 1 / i = \lambda(o_j^k) \qquad \textbf{(1.1)}$$

Contraintes de précédence entre l'opération chargement et l'opération de déchargement : La date de début t_i de l'opération de déchargement i doit être supérieure à la date de chargement $i-1$ où $i-1$ précède immédiatement i dans la gamme du job j $\forall j \in J$.

$$t_i \geq t_{i-1} + S_{i-1} \qquad \forall j \in J, \forall k \leq n_{i-1} / i = \lambda(u_j^k) \qquad \textbf{(1.2)}$$

Contraintes de précédence entre les opérations machine et les opérations de déchargement : La date de début t_p de l'opération machine p ne peut pas commencer avant la fin de l'opération de déchargement q où p suit immédiatement q dans la gamme du job $\forall j \in J$.

$$t_p \geq t_q + S_q \qquad \forall j \in J, \forall k \leq n_{i-1} / p = \lambda(o_j^{k+1}), q = \lambda(u_j^k) \qquad \textbf{(1.3)}$$

Les contraintes disjonctives entre les opérations machines : Cette contrainte assure que deux opérations machine $i \in O$ et $j \in O$ se réalisant sur la même machine $\mu_i = \mu_j$ ne peuvent pas s'exécuter en même temps. Ces contraintes font intervenir la variable binaire b_{ij} qui vaut 0 si j est avant i et 1 dans le cas contraire.

$$t_i \geq t_j + p_j - H.b_{ij} \qquad \forall (i,j) \in O \ / \mu_i = \mu_j \qquad \textbf{(2.1)}$$

$$t_j \geq t_i + p_i - H.(1 - b_{ij}) \qquad \forall (i,j) \in O \ / \mu_i = \mu_j \qquad \textbf{(2.2)}$$

$$b_{ij} + b_{ji} = 1 \qquad \forall (i,j) \in O \ / \mu_i = \mu_j \qquad \textbf{(2.3)}$$

Si $b_{ij} = 0$, la contrainte (2.2.) est trivialement vérifiée puisque H est un grand entier positif et la contrainte (2.1) est active et peut être réécrite en $t_i \geq t_j + p_j$ ce qui signifie que l'opération machine i ne peut commencer sur la machine $\mu_i = \mu_j$ qu'une fois le traitement de l'opération machine j terminé.

Dans le cas où $b_{ij} = 1$, la contrainte (2.2) est active et la contrainte (2.1) est trivialement vérifiée puisque H est un grand entier positif ce qui signifie que l'opération machine j ne peut commencer sur la machine $\mu_i = \mu_j$ qu'une fois que l'opération machine i a terminé son traitement.

2.2.4. Contraintes sur le flot

La formulation linéaire du robot de capacité non unitaire peut se faire en étendant les propositions faites par (Artigues et al., (2003) pour le RCPSP en introduisant la formulation « AON-flow ».

En effet, le robot peut être assimilé à une ressource renouvelable : le nombre d'exemplaire de la ressource étant la capacité du robot. Ainsi, on place sur le sommet fictif (0) un flot disponible de valeur $r_0 = K + 1$ avec comme valeur de K la capacité du robot. Si la capacité du robot $K = 2$ alors le flot disponible vaut $r_0 = 2 + 1 = 3$.

Pour se convaincre de l'intérêt de cette modélisation, il suffit de constater que l'évolution de la charge du robot obéit à des règles simples qui sont les suivantes :

- Règle 1 : le flot sortant d'une opération de chargement est diminué de 1.
- Règle 2 : le flot sortant d'une opération déchargement est augmenté de 1.
- Règle 3 : une opération de chargement doit recevoir un flot >1.
- Règle 4 : une opération de déchargement doit recevoir un flot $< r_0$.
- Règle 5 : comme il y a un et un seul robot, le flot se dirige en sortie vers une et une seule destination.

Il est facile de constater qu'une solution respectant les contraintes précédentes, définie une solution du problème. Prenons le problème *JS4* de taille 3x3 que nous avons déjà utilisé au début du chapitre.

Dans cet exemple les temps opératoires et les gammes ne sont pas renseignés car ils ne sont pas nécessaires à la compréhension de la modélisation. Le robot utilisé possède une capacité de deux places. Le flot disponible sur le nœud (0) est $r_0 = 2+1=3$. La Figure 2-5 donne une illustration d'une solution au problème du flot : cette solution définit les mouvements du robot.

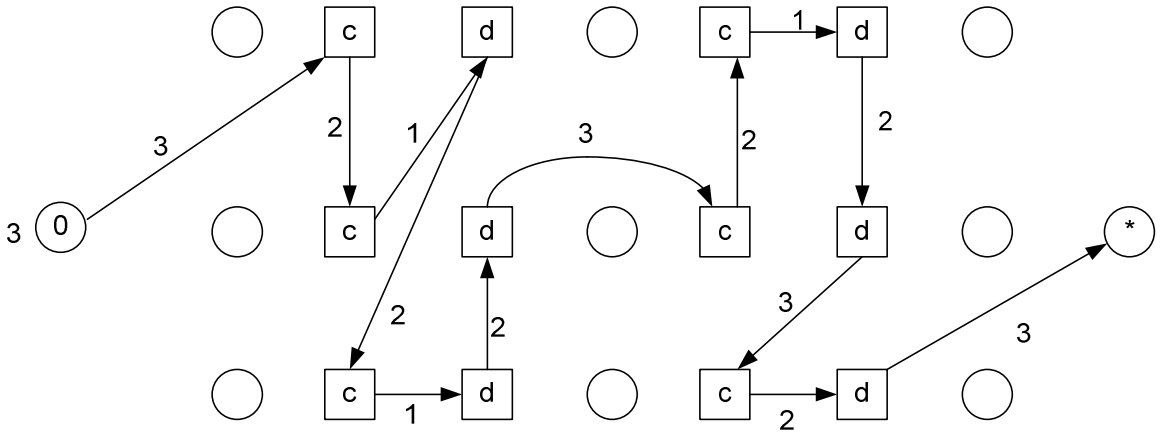


Figure 2-5 : Exemple de solution au problème de flot

Le flot de la Figure 2-6 ne satisfait pas les règles 3. Car le robot fait 3 chargements sans déchargement donc il porte 3 pièces.

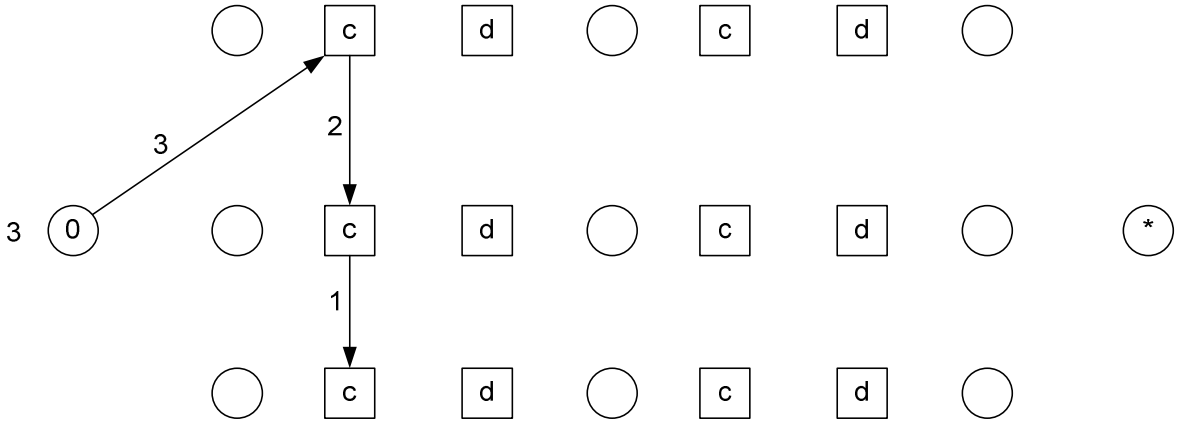


Figure 2-6 : Flot ne respectant par la règle 3

Un autre exemple est donné par la Figure 2-7 : le sommet (0) possède deux arcs sortant à flot non nul ce qui est interdit par la règle 5.

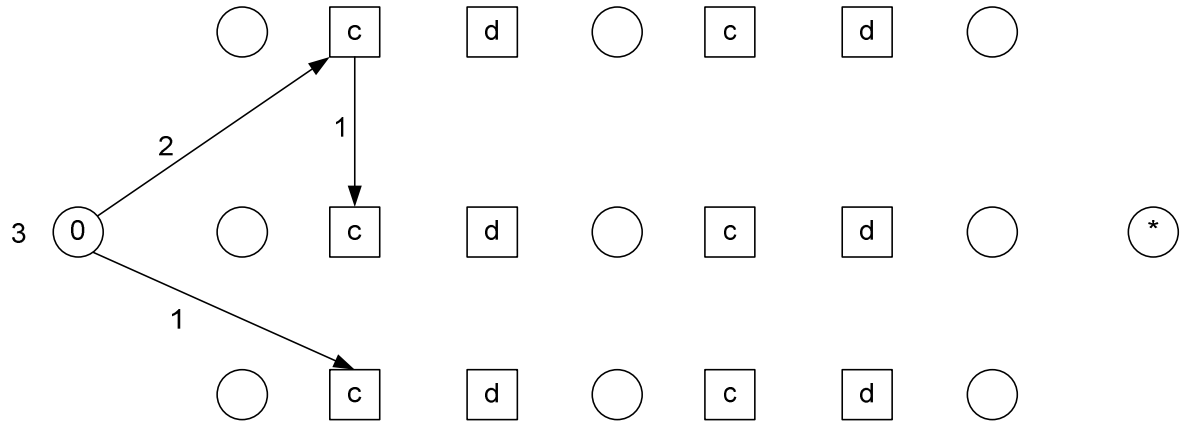


Figure 2-7 : Flot ne respectant la règle 5

Dans ce qui suit on introduit les contraintes qui gèrent la charge du robot en respectant les cinq règles précitées.

Les contraintes de capacité du robot : $y_{i,j}^{c}$ est une variable égale à 1 si l'opération transport i précède immédiatement l'opération transport j avec le robot transportant exactement c pièces. Par exemple, $y_{i,j}^{2}=1$, signifie qu'il existe un arc entre l'opération transport i et l'opération transport j, ce qui représente un déplacement du robot transportant exactement 2 pièces. Le robot se déplace, avec une charge unique donc dans notre exemple soit il ne transporte aucune pièce ($y_{i,j}^{0}=1$), soit il transporte une pièce $\left(y_{i,j}^{1}=1\right)$ soit il transporte 2 pièces $\left(y_{i,j}^{2}=1\right)$

$$\sum_{c=0}^{K} y_{i,j}^{c} \le 1 \qquad \forall (i,j) \qquad \textbf{(3)}$$

Contrainte de flot disponible sur le nœud $\{0\}$: Le sommet des flots sortant du nœud $\{0\}$ à destination des sommets de chargement, $\forall i \in L$, est égale à r_0.

$$\sum_{i=1}^{n} \varphi_{0,i} = r_0 \qquad \textbf{(4.1)}$$

Contrainte de flot entrant vers le nœud $\{*\}$: La somme des flux entrant sur le nœud $\{*\}$ est égale à r_0.

$$\sum_{i=1}^{n} \varphi_{i,*} = r_0 \qquad \forall i \in U \qquad \textbf{(4.2)}$$

Les contraintes 4.1 et 4.2 permettent de respecter la contrainte de conservation de flot : la somme des flots sortant du sommet $\{0\}$ est égale à la somme des flots entrant dans le sommet final noté $\{*\}$

Contrainte sur la charge des opérations de chargement : Ces contraintes expriment le fait que le robot ne peut pas se déplacer d'une opération de chargement vers une autre opération de chargement s'il est vide ou si sa capacité est atteinte.

$$y_{i,j}^{0} = 0 \qquad \forall i \in L,\ \forall j \in L \qquad \textbf{(5.1)}$$

$$y_{i,j}^{K} = 0 \qquad \forall i \in U,\ \forall j \in L \qquad \textbf{(5.2)}$$

Contrainte sur les opérations de chargement : Pour aller vers une opération de chargement, il faut qu'il y ait 1 place de disponible sur le robot i.e. le flux entrant doit être strictement plus grand que 1.

$$\sum_{j \in E} \varphi_{j,i} \geq 1 + \varepsilon \qquad \forall i \in L \qquad \textbf{(6.1)}$$

Contrainte sur les opérations de déchargement : Il faut qu'il y est une pièce dans le robot i.e. que le flux entrant soit strictement inférieur à *K*.

$$\sum_{j \in E} \varphi_{j,i} \leq K - \varepsilon \qquad \forall i \in U \qquad \textbf{(6.2)}$$

Contrainte sur la place occupée au moment du chargement : Le flux sortant est égal au flux entrant moins 1.

$$\sum_{j \in E} \varphi_{i,j} = \sum_{j \in E} \varphi_{j,i} - 1 \qquad \forall i \in L \qquad \textbf{(7.1)}$$

Contrainte sur la place libérée au moment du déchargement : Après une opération de déchargement, une place libre supplémentaire apparait.

$$\sum_{j \in E} \varphi_{i,j} = \sum_{j \in E} \varphi_{j,i} + 1 \qquad \forall i \in U \qquad \textbf{(7.2)}$$

2.2.5. Contrainte de disjonction transport

Ces contraintes assurent que si un flot circule entre deux opérations transport (cela signifie que les deux opérations sont consécutives), alors les dates de début de ces deux opérations sont distantes d'une durée égale au temps de transport plus le temps de chargement ou de déchargement S_i.

$$t_j \geq t_i + T_{i,j}^{c} + S_i - (1 - y_{i,j}^{c}) \times H \qquad \forall i \in T, \forall j \in T \qquad \textbf{(8)}$$

Dans la contrainte (8), le fait d'ajouter la durée S_i évite l'apparition de cycles et cette durée est souvent requise dans les problèmes réels d'ateliers de production, son emploi nous rapproche dons de la réalité.

Relation entre les variables $y_{i,j}^{k} = 1$ et $\varphi_{i,j}$: Les contraintes (9 .1) assurent que si aucun flot ne circule entre deux opérations i et j ($\varphi_{i,j} = 0$), alors tous les $y_{i,j}^{k}$ seront nulles. Si un flot non nul circule entre les opérations i et j il existe une variable $y_{i,j}^{k}$ prenant la valeur 1. A chaque $y_{i,j}^{k}$ on associe une constante $(r_0 - k)$.

$$\sum_{k=0}^{K}(r_0-k)y_{i,j}^{k}=\varphi_{i,j} \qquad \forall i\in U,\forall j\in E \qquad \textbf{(9.1)}$$

La contrainte assure que $\varphi_{i,j}=r_0$ si $y_{i,j}^{0}=1$ et de manière générale $\varphi_{i,j}=x$ si $y_{i,j}^{x-1}=1$.

Considérons un exemple où $r_0=3$ c'est-à-dire le cas d'un robot de capacité $K=2$. Dans ce cas particulier la contrainte (9.1) se réécrit : $3.y_{i,j}^{0}+2.y_{i,j}^{1}+1.y_{i,j}^{2}=\varphi_{i,j}$.

Pour bien comprendre le fonctionnement de cette contrainte, nous nous servons du graphe présenté dans la Figure 2-8. Dans ce graphe les opérations sont numérotées d'une manière croissante de 0 à *, toutes les opérations appartenant au même job, possède des numéros consécutifs.

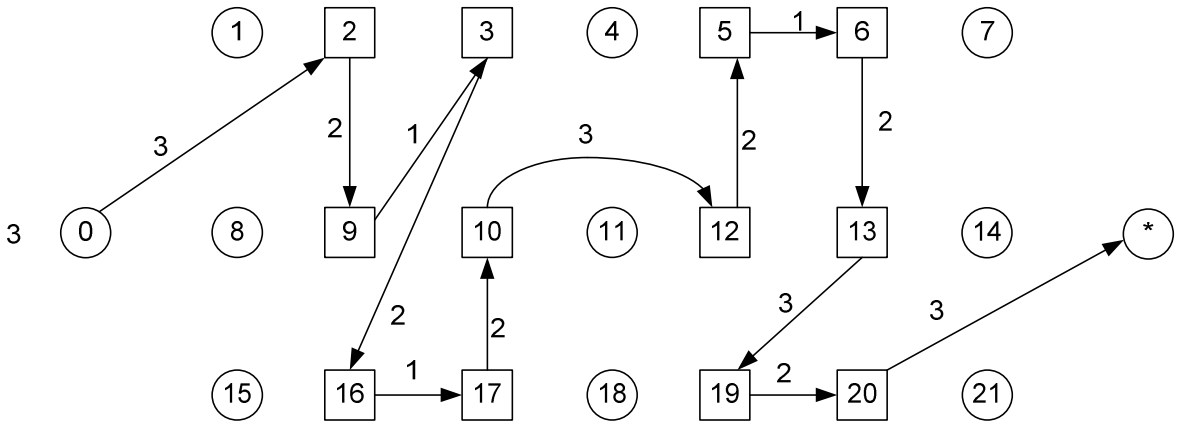


Figure 2-8 : Explication de la contrainte (9.1)

On constate que le robot se déplace entre l'opération 2 et l'opération 9 en portant une pièce (flux de 2), ce qui veut dire que : $y_{2,9}^{0}=0$, $y_{2,9}^{1}=1$ et $y_{2,9}^{2}=0$. De la même manière en se déplaçant de l'opération 9 à l'opération 3 le robot porte 2 pièces alors on : $y_{9,3}^{0}=0$, $y_{9,3}^{1}=0$, $y_{9,3}^{2}=1$. On note qu'il existe un décalage, entre la valeur du flot $\varphi_{i,j}$ qui circule entre deux opérations $(i\rightarrow j)$ et la valeur k de la variable $y_{i,j}^{k}$ non nulle pour le couple (i,j) considéré, qui renseigne la charge présente sur le robot lors du déplacement $(i \text{ vers } j)$. Pour illustrer cette situation sur l'exemple de dessus, on a $y_{2,9}^{1}=1$ avec $\varphi_{2,9}=2$. La contrainte (9.1) nous donne $1.y_{2,9}^{0}+2.y_{2,9}^{1}+3.y_{2,9}^{2}=2\Leftrightarrow\varphi_{2,9}=2$ ce qui donne bien sur $y_{1,5}^{1}=1$ ce qui modélise bien ce décalage.

Le flot en sortant d'un nœud $i\in E$, n'a qu'une seule destination :

$$\sum_{k=0}^{K}\sum_{j\in E}y_{i,j}^{k}=1 \qquad \forall i\in E \qquad \textbf{(9.2)}$$

Le flot entrant vers le nœud $i\in E$, doit provenir d'une seule source nœud $j\in E$:

$$\sum_{k=0}^{K}\sum_{j\in E}y_{j,i}^{k}=1 \qquad \forall i\in E \qquad \textbf{(9.3)}$$

Contraintes d'élimination des cycles de longueur 2 : La contrainte (10) impose que soit $\varphi_{j,i}=1$ (c'est-à-dire que le robot circule entre l'opération transport *j* et l'opération transport *i*) soit la situation inverse se produit ($\varphi_{i,j}=1$) soit aucune de ces deux situations (le robot ne circule pas entre les deux opérations de transport *i* et *j*). Si $\sum_{k=0}^{K} y_{i,j}^{k}=1$ (le robot circule entre i et j) la contrainte (10) impose que $\sum_{k=0}^{K} y_{j,i}^{k}=0$ et inversement.

$$\sum_{k=0}^{K} y_{i,j}^{k}+\sum_{k=0}^{K} y_{j,i}^{k}\le 1 \qquad \forall i\in E,\ \forall j\in E \qquad \textbf{(10)}$$

2.2.6. La fonction objectif

La fonction objectif du ***PL*** peut être écrite comme une fonction linéaire des variables de décision pour exprimer des objectifs d'ordonnancement classiques qui ne se limite pas au makespan qui est le temps d'achèvement de la dernière opération du problème.

$$Min\ C_{\max}$$

$$C_{\max}\ge t_i+p_i \qquad \forall i\in O_f, \qquad \textbf{(11)}$$

On peut remarquer que toutes les variables sont des entiers non-négatifs et que ce PL inclut un nombre important de variables binaires.

2.2.7. Analyse des cycles

Dans cette partie on essaie d'analyser si cette modélisation permet l'apparition de cycles et d'étudier les conditions de leur apparition et d'y remédier s'ils apparaissent. Pour que les cycles apparaissent il faut satisfaire la condition suivante il doit y avoir une égalité entre les dates des opérations de transport. En effet, cette condition permet de boucler indéfiniment entre ces opérations.

Cette égalité apparait lorsque deux opérations de transport sont à réaliser auprès de deux machines se séparant par une distance nulle ; autrement dit lorsque c'est la même machine et que les temps de chargement et déchargement sont nuls. Supposons que les opérations $(i,j)\in E^2$ sont réalisées auprès de la même machine alors :

Pour que $t_i=t_j$ la contrainte (8) doit être pour les deux opérations comme suit :

$$t_j\ge t_i+T_{i,j}^{c}+S_i-(1-y_{i,j}^{c})\times H \Leftrightarrow t_j\ge t_i-(1-y_{i,j}^{c})\times H \qquad \forall(i,j)\in E^2 \qquad \textbf{(8)}$$

$$t_i\ge t_j+T_{j,i}^{c'}+S_j-(1-y_{j,i}^{c'})\times H \Leftrightarrow t_i\ge t_j-(1-y_{j,i}^{c'})\times H \qquad \forall(i,j)\in E^2 \qquad \textbf{(8)}$$

Donc il claire, que pour que les deux opérations aient les même dates de début il faut avoir $T_{i,j}^{c}=T_{j,i}^{c'}=S_i=S_j=0$, c-à-d que la distance soit nulle et les temps de chargement/déchargement également.

D'autre part il faut $y^{c}_{i,j} = y^{c'}_{j,i} = 1$ pour que les deux contraintes soient toutes les deux actives. Comme la contrainte (10) évite l'égalité $y^{c}_{i,j} = y^{c'}_{j,i} = 1$ alors le ***PL*** rend inactive l'une des deux contraintes et le cycle s'élimine. Les autres cas seront automatiquement éliminer par le fait que le chaque flot viens d'une seul source et n'a qu'une seule destination (cf. contraintes (9.3)).

2.2.8. Test de la modélisation linéaire

Nous avons réalisé un test sur un benchmark composé de petites instances *LLT*. LeTableau 2-3 récapitule les résultats de ce test.

Tableau 2-3 : Résultats des tests

	Capacité=1		Capacité=2		Capacité=3		Capacité=4	
Instances	**LB**	**UB**	**LB**	**UB**	**LB**	**UB**	**LB**	**UB**
LLT1	37	37	37	37	37	37	35	37
LLT2	77	-	41	41	-	-	32	-
LLT3	77	-	39	41	-	-	-	-
LLT4	83	83	-	-	-	-	-	-
LLT5	67	130	-	-	-	-	-	-

La formalisation linéaire proposée est une formalisation linéaire originale pour le problème de job shop avec transport avec un chariot à capacité non unitaire. Elle a le mérite d'unifier dans un même programme linéaire toutes les contraintes.

Malheureusement, la résolution directe ne permet pas d'obtenir des solutions optimales de problèmes de grande taille (6 jobs maximums). C'est pourquoi, nous proposons de développer des méthodes approchées basées sur le modèle de graphe conjonctif-disjonctif.

2.3. Schéma d'optimisation

Notre schéma d'optimisation se base sur l'utilisation de méthodes approchées telles que des méta-heuristiques comme les algorithmes mémétiques, c'est d'ailleurs cette dernière que nous avons retenu pour résoudre le job-shop avec un robot de capacité non unitaire. Le choix est guidé par les bons résultats obtenus avec l'emploi de cette méta-heuristique pour résoudre le job-shop avec plusieurs robots présenté au chapitre précédent.

Dans la même logique utilisée au précédent chapitre nous proposons une modélisation du problème en utilisant un vecteur par répétition avec quelques propriétés supplémentaires qui sont discutée ci-dessous.

2.3.1. Modélisation sous forme de vecteur à valeurs dupliquées

Le problème de job-shop avec un robot de capacité non unitaire peut facilement être représenté par un vecteur *MTS* ; ce dernier comporte autant de cases que d'opération existantesdans le problème et chaque job apparaît autant de fois qu'il possède d'opérations.

Cependant, contrairement à la modélisation utilisée pour représenter le job-shop avec plusieurs robot de capacité unitaire, où tous les vecteurs conduisent à des solutions correctes et admissibles, plusieurs vecteurs *MTS* représentant le job-shop avec un robot de capacité non unitaire conduisent à des solutions inadmissibles violant la capacité du robot. De plus le nombre de vecteurs donnant des solutions inadmissibles est d'autant plus élevé que la capacité est petite. Se servant de la solution de *JS4*, nous donnons une illustration du vecteur *MTS* représentant cette solution.

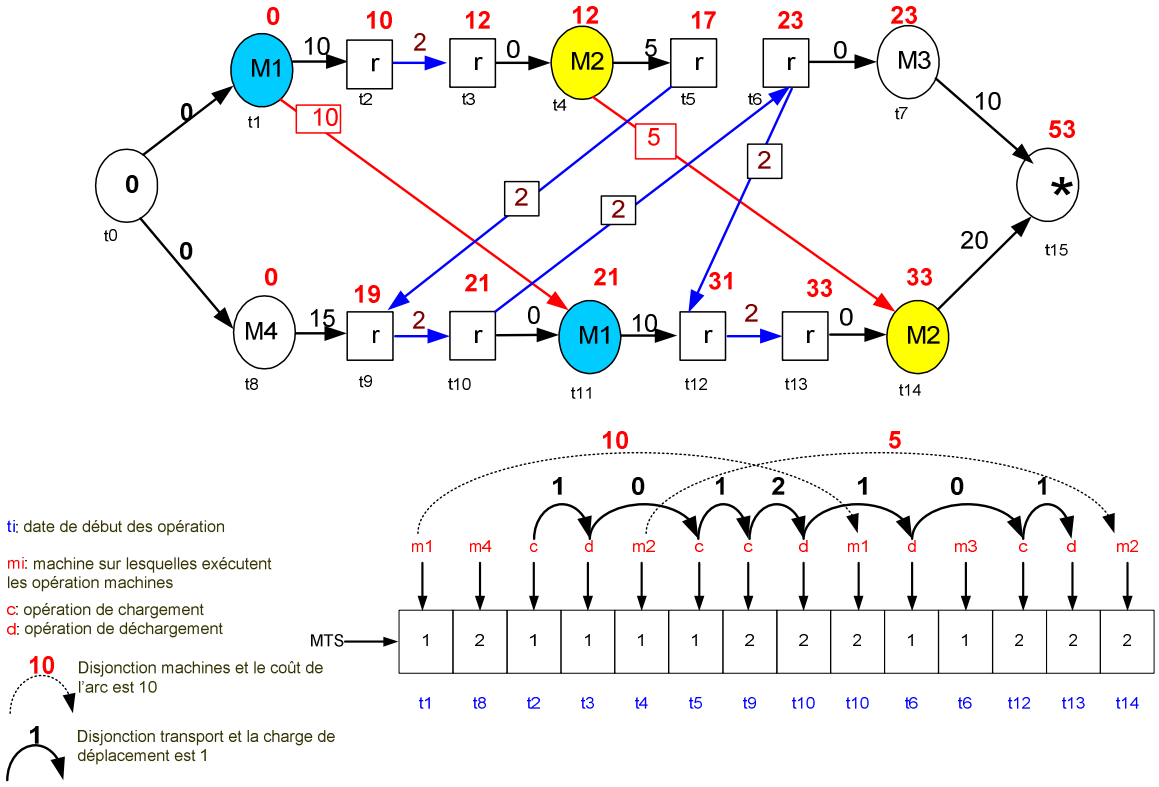


Figure 2-9 : Représentation par vecteur d'une solution de job-shop avec un robot de capacité 2

Comme le montre la Figure 2-9, la capacité du robot est bien prise en compte dans cette modélisation, et le vecteur *MTS* à lui seul permet de représenter toutes les contraintes du problème avec comme seule spécificité la précédence de gauche à droite. La première apparition du chiffre concerne la première opération machine, la seconde apparition concerne l'opération de chargement et la troisième l'opération de déchargement et ainsi de suite.

Il est facile de constater que la contrainte de capacité du robot n'est pas prise en compte dans le vecteur. Ainsi, certain vecteurs *MST* ne la respectent pas dans le sens où le nombre de pièces transportées dépasse la capacité maximale.

2.3.2. Démonstration qu'il est possible de corriger une séquence inadmissible

Supposons qu'il existe une séquence *SEQ* qui contient une solution inadmissible et soit *pos*[*i*] la fonction qui donne la position de l'opération *i* dans la séquence *SEQ*. *SEQ* contient toutes les opérations des différents jobs avec la précédence du gauche à droite. Prouvons maintenant qu'on peut toujours corriger cette séquence.

Soit K la capacité maximale du robot et k la variable qui servira à stocker la charge du robot à chaque case du vecteur.

On initialise la variable k à zéro, puis on parcoure le vecteur SEQ de gauche à droite puis à chaque fois qu'on rencontre une opération de chargement on incrémente k alors que lorsqué on rencontre une opération de déchargement on décrémente k .

Lors de cette opération, si $k = K$ et la prochaine case contient une opération de type chargement alors à partir de cette case on viole la capacité du robot ; soit i l'indice de cette case.

Pour corriger la séquence il suffit de faire une permutation de la prochaine opération de type chargement avec la prochaine opération de type déchargement. Une action toujours possible car il y a autant d'opérations de type chargement que d'opérations de type déchargement-

Si, par exemple, on viole la capacité à la case j et la prochaine opération de déchargement se trouve à la case i, la permutation suivante permet de décrémenter la charge $k = K - 1$: $tmp := SEQ[i], SEQ[i] := SEQ[j] et\ SEQ[j] := tmp$.

On réitère cette opération autant de fois que nécessaire jusqu'à la fin de l'analyse de la séquence. Nommons cette procédure Corrige(MTS, K).

Il est donc toujours possible de corriger une séquence inadmissible (violant la capacité du robot) comme nous venons de le voir. Le paragraphe suivant propose une recherche locale permettant d'améliorer une solution représentée par le vecteur MTS à l'aide de quelques mouvements élémentaires. Cette procédure porte le nom de Local_Search : elle exploite le chemin critique.

2.4. Recherche Locale

La recherche locale exploite le chemin critique qui est le plus long chemin présent dans le graphe disjonctif orienté. Elle exploite les propriétés des blocs, qui dans notre cas, sont en nombre de deux :

- un bloc disjonctions machines représentant l'exécution d'opérations machines appartenant à des jobs différents d'une manière successive ;
- un bloc disjonctions transport comme dans le cas de du job-shop avec plusieurs robots (voir chapitre précédent).

L'amélioration des solutions passe par l'élimination d'un arc du chemin critique, et ceci par quelques mouvement élémentaires réalisés sur les blocs. Donc pour obtenir de nouveaux chemins critiques, il suffit de casser ceux existant aux extrémités des blocs. Afin d'exploiter facilement le graphe disjonctif on sauvegarde le chemin critique dans un vecteur. Le vecteur utilisé dans notre cas est appelé ***PRED***. Il est construit au fur et à mesure de l'évaluation du graphe.

Pour construire ce vecteur on introduit une bijection $f : \{1,2,..,n+1\} \to 0 + \{*\}$. Ainsi, chaque indice correspond à une opération du problème, tandis que chaque élément de vecteur ***PRED*** fait référence à l'indice dans ce même vecteur du prédécesseur du sommet en question dans le graphe, ainsi $PRED[i] = y$; $f(y)$ est alors l'opération précédante de l'opération $f(i)$ sur le chemin critique.

Pour casser le chemin critique on procède par l'inversion des arcs qui sont à l'extrémité des blocs, dans notre cas, on inverse les arcs à l'extrémité la plus à droite, car on analyse le chemin

critique à l'envers partant du nœud $\{*\}$ vers $\{0\}$. A chaque fois qu'on rencontre un bloc, on inverse l'arc et on réévalue la nouvelle configuration du graphe. Ces opérations sont effectuées sur le vecteur *MTS* . Le psedo code de cette recherche locale est donné par l'algorithme suivant :

Algorithme 2-1 : Recherche locale

```
Nom de la procédure : Local_Search
    S: solution donnée par (MTS,PRED, Cmax) ;
    K :capacité maximale du robot ;
Début
    i:=N+1                          // commencer l'analyse à partir du nœud {*}
    iter:=1                         // initialiser le compteur d'amélioration à 1
    C^best := S.Cmax                // Initialiser le meilleur coût au coût de la solution
                                       d'entrée
  Tant que (i≠0) and (iter<=nm) faire
    j:=PRED(i)                      // j le prédécesseur immédiat de l'opération i dans le
                                          chemin critique
    Si (job[i] ≠ job[j]) alors
      Si (i et j sont deux opérations machines) ou (i et j sont deux opérations
                                                          transport) alors
        Save_S :=S
        Permuter les opérations i et j dans MTS ;
        Corrige(S.MTS,K) ;
        Evaluer(S.MTS, S.PRED, S.Cmax);
        Si ( S.Cmax < C^best ) alors
            C^best := C ;           // amélioration
            i:= PRED (N+1) ;        // retour au nœud {*} du nouveau chemin critique
            iter :=iter+1 ;
        Sinon
            S:=Save_S ;             // annuler le changement
            i:= PRED(i) ;           // continuer l'analyse d'ancien chemin critique
        Fin Si
     Fin Si
     Fin Tant que
  Retourner S ;
Fin
```

La correction de la séquence consomme beaucoup d'autant plus de temps que la capacité du robot est petite et le nombre de job grand. En effet, le nombre de vecteurs représentant les séquences correctes est relativement petit par rapport au nombre de vecteurs représentant des solutions inadmissibles, ce qui nous a amené à chercher à développer un indicateur qui permet de privilégier les solutions correctes et de minimiser l'impact des vecteurs violant la capacité. Ainsi nous proposons une technique d'acceptation des solutions ne respectant la capacité des robots.

Dans un processus d'optimisation basé sur l'emploi des meta-heuritiques on associe à chaque solution un indicateur appelé « fitness » servant à renseigner la qualité de la solution. Pour minimiser le rôle des solutions inadmissibles sans les écarter on introduit une pénalité à chaque solution lors du calcul de fitness.

Si une solution S_i possède un vecteur MTS_i ne respecte pas la capacité sont fitness est donné par la relation suivante : $fitness_i = surcharge_{\max} \times 10000000 + C_{\max} + (n - i_v) \times 10000$ où :

- i_v indice dans le vecteur MTS à partir duquel la capacité est violée ;
- $surcharge_{\max} = (charge_{\max} - capa_{\max})$ représente la surcharge maximale atteinte à partir de l'indice i_v ; $charge_{\max}$ charge maximale atteinte à partir de l'indice i_v et $capa_{\max}$ la capacité maximale du robot. Cette surcharge est calculée uniquement pour les solutions inadmissibles, pour les autres solutions elle est égale à zéro ;
- $C_{\max}$ makespan trouvé à la fin de l'évaluation de la solution. A partir de i_v, les temps de déplacement du robot sont correspondent à ceux de sa capacité maximale.

Ainsi si une solution S_i ne viole pas la capacité du robot sont fitness est alors $fitness_i = C_{\max}$ car $surcharge_{\max} = 0$ et $i_v = n$, par contre si une solution dont le $C_{\max} = 125$, le nombre d'opérations $n = 100$, l'indice i_v à partir duquel la capacité est violée est $i_v = 70$ et cette dernière atteint une surcharge max de 3 alors le fitness est donné par : $fitness_i = 3 \times 10000000 + (100 - 70) \times 10000 + 125 = 30300125$

En lisant $fitness_i$ on a accès à toutes les informations utiles relatives à la qualité de la solution. En effet, nous avons la surcharge maximale atteinte à gauche (3), le $C_{\max}$ à droite (125), le nombre d'éléments dans le vecteur MTS_i ne réspectant pas la capacité du robot au milieu (30), le tout séparé par un zéro. Avec cette formalisation du fitness les solutions inadmissibles sont fortement pénalisées compte tenu du fait que la survie des individus dans la population est dépendent de leur fitness. Nous préférons de garder les solutions inadmissibles pour permettre l'accessibilité de l'espace de recherche.

2.5. Algorithme Mémétique

L'algorithme mémétique est l'approche que nous avons retenu pour résoudre le job-shop avec un robot possédant une capacité non unitaire, tous les éléments de cette approche sont décris dans les point suivants. Le fonctionnement général est donné à la fin de ce paragraphe par l'Algorithme 2-5.

2.5.1. Le codage

Comme le montre la Figure 2-10 on considère qu'un chromosome est une vecteur *MTS* auquel on associé un tableau PRED pour stocker le chemin critique et un signe pour gérer une fonction de hachage : *Evaluer(S.MTS,S.PRED,S.*C_{max}*,S.sign)*

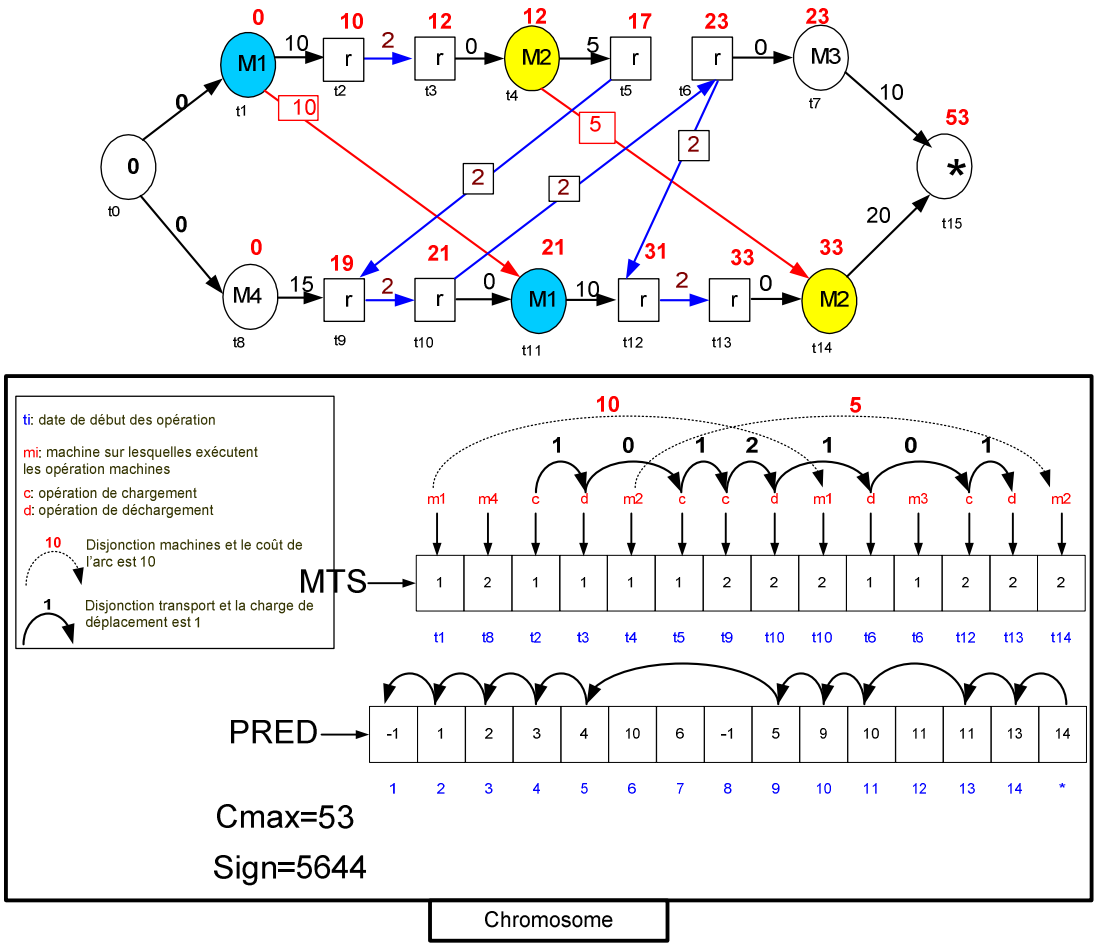


Figure 2-10 : Représentation d'un chromosome

2.5.2. Croisement

Le croisement est de type GOX : il s'applique au vecteur MTS .

Algorithme 2-2 **:** Croisement

```
Nom de la procédure : Crossover(P1.MTS,P2.MTS,enfant)
Données
         P1, P2  : chromosomes,
         n_i             : nombre d'opérations machine du job i
sorties
  enfant : chromosome
Début
  avec probabilité de 1/2 permuter les rôles des parents P1 et P2
  k = random (1.. Σ_{i=1}^{n} n_i )
  l = random (1.. Σ_{i=1}^{n} n_i )
  Pour i:=k à l faire
       enfant.MTS[i]:=P1.MTS[i]
       USE[P1.MTS[i]]:= USE[P1.MTS[i]]+1
  Fin Pour
  j :=0

  Pour i:=1 à n faire
   j :=i+1 ;
   Si (j=k) et (l+1<= Σ_{i=1}^{n} n_i ) alors
     j :=l+1 ;
   Fin Si
   Si (USE[P1.MS[i]< n_i )(j<= Σ_{i=1}^{n} n_i )  alors
       enfant.MTS[j]:=P2.MTS[i]
       USE[P2.MTS[i]]:=USE[P2.MTS[i]]+1
   Fin Si
  Fin Pour
  Evaluer(enfant.MTS, enfant.PRED, enfant.C_max, enfant.sign)
Fin
```

2.5.3. La détection des doubles ou fonction de hachage

La technique utilisée pour détecter les doublons est la même que celle utilisée dans l'algorithme mémétique du chapitre précédent. Elle repose sur l'emploi d'un vecteur $DOUBLE$ dont chaque case contient le nombre d'individu possédant une signature égale à l'indice de la case. De cette manière on peut vérifier en $O(1)$ si un chromosome C possédant une signature $C.sign$ existe et ceci en consultant la case $DOUBLE[C.sign]$. Pour que ça fonctionne, il faut envisager une taille relativement grande fixée en pratique de manière empirique. Pour donner une illustration de ce qu'occupera en mémoire un vecteur de taille 50000 ou chaque case contient un nombre allant de -128..127 (chaque case est codé par un octet en mémoire), la taille du vecteur $DOUBLE$ est alors : $Taille(DOUBLE) = {(50000)}/{1024} \approx 49\ MO$. Actuellement les tailles standard des mémoires vives sont supérieur ou égales à $2\ GO = 2 \times 1024 = 2048\ MO$ et permettent d'envisager des tailles plus grandes.

2.5.4. La génération de la population initiale

Nous avons choisi de générer des individus qui composent la population d'une manière aléatoire. Chaque individu C est généré en deux étapes :

- on génère le vecteur *C.MTS* d'une manière aléatoire ;
- on renseigne avec la procédure *Evaluer(C.MTS, C.PRED, C.C_{max}, C.sign)* le vecteur *C.PRED* et les variables $(C.C_{\max}, C.sign)$ ce qui permet ainsi de compléter le chromosome.

Puis on ajoute l'individu à la population s'il n'est pas un doublon par le biais de la fonction *add(C, pop, DOUBLE, i)* où *i* est l'indice du rajout. Le pseudo code de cette fonction est donné par l'Algorithme 2-3. L'Algorithme 2-4 présente le pseudo code de la génération de la population initiale.

Algorithme 2-3 : Fonction d'ajout des chromosomes dans la population

```
Nom de la fonction : add(C, pop, DOUBLE, i): boolean;
Paramètres de cette fonction
    pop         : population initiale
    c           : chromosome
    DOUBLE      : vecteur utilisé pour detecter les doublons
Début
    Si DOUBLE[C.SIGN] = 1 alors
        retourner faux  // ajout non réalisé
    Sinon
      DOUBLE[pop[i].SIGN] := 0 ;
      DOUBLE[C.SIGN] := 1
      pop[i] :=C
      retourner vrai // ajout réussi
    Fin Si
Fin
```

Algorithme 2-4 **:** Génération de la population initiale

```
Nom de la procédure : Pop_intiale (pop, n_demande)
Paramètres de cet algorithme
    n               : nombre de jobs présents dans le problème
    c               : chromosome
    nb_max_essai    : nombre maximal d'essais de génération de chromosome uniques
    K               : capacité du robot
    DOUBLE          : vecteur utilisé pour detecter les doublons

Début
    i:=1
    nb_essai=0
    Tant que (i< n_demande) and (nb_essai < nb_max_essai)  faire
      H3(C.MTS)
      Corrige (C.MTS,K)
      Evaluer(C.MTS, C.PRED, C.Cmax, C.SIGN)
      res := Add(pop,c,DOUBLE,i)
      Si (res=true) alors
          i :=i+1
          nb_essai :=0
      else
          nb_essai= nb_essai+1
      Fin si
    Fin Tant que
    pop.nb_pop := i
Fin
```

Nous venons de définir toutes les procédures de base nécessaires pour élaborer l'algorithme mémétique proposé, son le schéma général de fonctionnement est donné par l'Algorithme 2-5.

L'algorithme choisi de manière itérative les deux parents : *Select_deux_parent(P1,P2).* Puis, l'individu « *C* » est généré par le croisement GOX à partir des deux parents *P1* et *P2*. Avec une probabilité *Pm* une recherche locale est appliquée à « *C* » afin de l'améliorer, et S*elect(ind)* renvoi l'indice du chromosome à remplacer par « *C* » au cas où ce dernier n'est pas un doublon. Cette dernière action est faite par la fonction *add(Pop,C, DOUBLE,ind).*

Si le nombre d'itérations sans amélioration (*npi*) atteint le seuil (*np)* alors une opération de régénération de la population est exécutée. Ceci est réalisé par la procédure *Restart(Pop,p)* où *p* est le pourcentage des chromosomes à régénérer.

L'algorithme se termine lorsque le critère d'arrêt est satisfait. Celui-ci peut être un nombre maximal d'itérations à réaliser, un temps maximal d'exécution ou l'atteinte d'une valeur pour le meilleur individu de la population etc…

Algorithme 2-5 : Schéma globale de l'algorithme mémétique

```
Nom de la procédure : Memetic_Algorithme (pop, n_demande)
    pop : population            // population initiale
    C :chromosome               // chromosome utiliser pour stocker le résultat du
                                   croisement ( rôle de l'enfant)
Début
    Pop_intiale(pop,N_pop) // générer la population de départ pop

    Tant que nécessaire faire
       Select_deux_parent(P1,P2)              // sélectionner deux parrents
       C :=Crossover(P1.MTS,P2.MTS,)          // croiser les parrents P1 et P2 et
                                                 stocker le resulta dans C
       Select(ind)                            //  séctionner  l'indice  du  chromosome  à
                                                 remplacer
       Avec une probabilté Pm Local_Search(C) // faire une recherche locale sur C
       Add(pop, DOUBLE,C,ind)                 // ajouter le chromosome C à la
                                                 population
       Mettre_a_jour(meilleur,npi)            // mettre à jour le meilleur score et npi
       Si npi=np alors
          Restatrt(Pop,p)                     // regénérer p% d'individus par le
                                                 mécénisime d'élitisme
       Fin Si

    Fin Tant que
  Fin
```

2.6. Application numérique

Toutes les procédures sont implémentées sous Delphi 7.0 sous Windows XP Media Center. L'exécution est réalisée sur un ordinateur disposant d'un processeur Pentium E6300 de 1.8 Ghz, et de 2 GB de mémoire vive. Pour mesurer la performance de l'algorithme, nous avons envisagé 5 réplications avec des racines différentes pour le générateur aléatoire.

Dans le paragraphe suivant nous présentons les résultats des différents tests réalisés sur les instances de Bulge et Ulusoy. L'analyse de ces résultats est présentée dans le §2.6.2.

2.6.1. Test sur les instances de Bilge et Ulusoy

La notation suivante est utilisée dans les tableaux des résultats *Dev i :* déviation en pourcentage de la meilleure solution donnée dans la colonne capacité *i* par rapport à la solution donnée dans la colonne BFS. On la calcul par la formule $\frac{Capacité_i - BFS}{BFS}$

Les paramètres suivants sont utilisés pour algorithme

- nm: 60 // nombre maximal d'itération de la recherche locale ;
- nc: 115 // nombre de chromosomes dans la population ;
- np: 15000 // number of iterations before restart;
- pm: 32 // probabilité de la recherché local ;
- pr: 32 // pourcentage de chromosomes à remplacer pendant le restart.

On rappelle les deux critères utilisé pour les instances de Bilge et Ulusoy.

- C1: minimiser le Makespan en prenant en compte la dernière opération de transport de chaque job vers la machine de déchargement U (unload station), ce critère est notamment utilisé dans Deroussi et al., (2008).
- C2: minimiser le Makespan en ne prenant pas en compte la dernière opération de transport de chaque job vers la machine de déchargement U (unload station) pour lequel plusieurs résultats existent, Bilge et Ulusoy (1993). Bilge et Ulusoy (1995)Bilge et al., (1997) Abdelmaguid et al., (2004) et Deroussi et al., (2008).

Tableau 2-4 : Test sur les instances de Bilge et Ulusoy critère C1

Instances	**Meilleurs résultats à deux robots**	***Nos résultats avec un robot de capacité non unitaire***				***Déviation par rapport aux meilleurs résultats avec deux robots***			
	BFS^{10min}	***Capacité 1***	***Capacité 2***	***Capacité 3***	***Capacité 4***	***Dev2***	***Dev3***	***Dev3***	***Dev4***
EX11	114	226	162	134	134	98,25%	42,11%	17,54%	17,54%
EX21	116	226	162	135	122	94,83%	39,66%	16,38%	5,17%
EX31	121	244	174	148	138	101,65%	43,80%	22,31%	14,05%
EX41	138	276	190	173	155	100,00%	37,68%	25,36%	12,32%
EX51	110	218	153	127	123	98,18%	39,09%	15,45%	11,82%
EX61	129	240	155	152	143	86,05%	20,16%	17,83%	10,85%
EX71	133	290	194	154	142	118,05%	45,86%	15,79%	6,77%
EX81	167	260	186	172	175	55,69%	11,38%	2,99%	4,79%
EX91	127	235	163	140	140	85,04%	28,35%	10,24%	10,24%
EX101	153	295	199	180	180	92,81%	30,07%	17,65%	17,65%
EX12	90	162	104	94	92	80,00%	15,56%	4,44%	2,22%
EX22	82	138	101	98	92	68,29%	23,17%	19,51%	12,20%
EX32	89	152	103	90	102	70,79%	15,73%	1,12%	14,61%
EX42	100	192	133	122	111	92,00%	33,00%	22,00%	11,00%
EX52	81	154	104	84	81	90,12%	28,40%	3,70%	0,00%
EX62	102	157	120	109	117	53,92%	17,65%	6,86%	14,71%
EX72	86	181	124	107	94	110,47%	44,19%	24,42%	9,30%
EX82	155	164	157	155	155	5,81%	1,29%	0,00%	0,00%
EX92	106	169	113	114	114	59,43%	6,60%	7,55%	7,55%
EX102	139	226	163	150	154	62,59%	17,27%	7,91%	10,79%
EX13	98	182	108	94	92	85,71%	10,20%	-4,08%	-6,12%
EX23	89	158	109	90	95	77,53%	22,47%	1,12%	6,74%
EX33	96	155	108	102	98	61,46%	12,50%	6,25%	2,08%
EX43	102	196	130	112	105	92,16%	27,45%	9,80%	2,94%
EX53	89	177	102	86	86	98,88%	14,61%	-3,37%	-3,37%
EX63	105	170	118	112	118	61,90%	12,38%	6,67%	12,38%
EX73	93	180	136	107	103	93,55%	46,24%	15,05%	10,75%
EX83	155	169	155	155	155	9,03%	0,00%	0,00%	0,00%
EX93	107	179	122	110	114	67,29%	14,02%	2,80%	6,54%
EX103	139	227	161	155	152	63,31%	15,83%	11,51%	9,35%
EX14	140	267	176	140	138	90,71%	25,71%	0,00%	-1,43%
EX24	134	280	174	144	136	108,96%	29,85%	7,46%	1,49%
EX34	148	290	202	152	161	95,95%	36,49%	2,70%	8,78%
EX44	163	342	226	189	160	109,82%	38,65%	15,95%	-1,84%
EX54	134	262	170	137	130	95,52%	26,87%	2,24%	-2,99%
EX64	151	300	200	166	154	98,68%	32,45%	9,93%	1,99%
EX74	162	360	222	193	159	122,22%	37,04%	19,14%	-1,85%
EX84	178	348	225	203	191	95,51%	26,40%	14,04%	7,30%
EX94	149	273	186	157	151	83,22%	24,83%	5,37%	1,34%
EX104	183	354	222	206	196	93,44%	21,31%	12,57%	7,10%
Déviation totale moyenne						***83,22%***	***25,41%***	***9,86%***	***6,37%***

Le test suivant concerne les instances de bilge et Ulusoy avec le critère C2 en utilisant un robot de capacité non unitaire.

Tableau 2-5 : Test sur les instances de Bilge et Ulusoy critère C2

Instances	**Meilleurs résultats à deux robots**	***Résultats avec un robot de capacité non unitaire***				***Déviation par rapport aux meilleurs résultats avec deux robots***			
	BFS^{10min}	***Capacité 1***	***Capacité 2***	***Capacité 3***	***Capacité 4***	***Dev2***	***Dev3***	***Dev3***	***Dev4***
EX11	96	161	125	109	107	67,71%	30,21%	13,54%	11,46%
EX21	104	174	124	108	112	67,31%	19,23%	3,85%	7,69%
EX31	105	164	129	123	115	56,19%	22,86%	17,14%	9,52%
EX41	116	206	162	146	141	77,59%	39,66%	25,86%	21,55%
EX51	87	158	117	100	95	81,61%	34,48%	14,94%	9,20%
EX61	121	201	148	136	130	66,12%	22,31%	12,40%	7,44%
EX71	118	221	146	124	106	87,29%	23,73%	5,08%	-10,17%
EX81	161	209	169	169	162	29,81%	4,97%	4,97%	0,62%
EX91	117	196	144	133	132	67,52%	23,08%	13,68%	12,82%
EX101	150	234	184	172	170	56,00%	22,67%	14,67%	13,33%
EX12	82	123	86	86	84	50,00%	4,88%	4,88%	2,44%
EX22	76	111	94	82	85	46,05%	23,68%	7,89%	11,84%
EX32	85	109	89	89	86	28,24%	4,71%	4,71%	1,18%
EX42	88	156	104	96	96	77,27%	18,18%	9,09%	9,09%
EX52	69	120	73	71	71	73,91%	5,80%	2,90%	2,90%
EX62	98	135	111	111	106	37,76%	13,27%	13,27%	8,16%
EX72	85	139	95	91	84	63,53%	11,76%	7,06%	-1,18%
EX82	151	153	151	151	151	1,32%	0,00%	0,00%	0,00%
EX92	102	165	116	108	106	61,76%	13,73%	5,88%	3,92%
EX102	137	208	147	139	146	51,82%	7,30%	1,46%	6,57%
EX13	84	129	90	84	78	53,57%	7,14%	0,00%	-7,14%
EX23	86	123	100	86	86	43,02%	16,28%	0,00%	0,00%
EX33	86	117	90	88	88	36,05%	4,65%	2,33%	2,33%
EX43	91	151	110	96	96	65,93%	20,88%	5,49%	5,49%
EX53	75	123	82	71	70	64,00%	9,33%	-5,33%	-6,67%
EX63	104	140	113	113	109	34,62%	8,65%	8,65%	4,81%
EX73	88	149	107	85	93	69,32%	21,59%	-3,41%	5,68%
EX83	153	158	153	153	153	3,27%	0,00%	0,00%	0,00%
EX93	105	158	118	113	107	50,48%	12,38%	7,62%	1,90%
EX103	143	207	153	151	148	44,76%	6,99%	5,59%	3,50%
EX14	103	191	126	108	108	85,44%	22,33%	4,85%	4,85%
EX24	113	200	130	114	104	76,99%	15,04%	0,88%	-7,96%
EX34	113	194	138	123	116	71,68%	22,12%	8,85%	2,65%
EX44	126	244	164	145	137	93,65%	30,16%	15,08%	8,73%
EX54	97	185	118	98	94	90,72%	21,65%	1,03%	-3,09%
EX64	123	223	149	132	127	81,30%	21,14%	7,32%	3,25%
EX74	128	266	159	135	110	107,81%	24,22%	5,47%	-14,06%
EX84	163	259	179	170	170	58,90%	9,82%	4,29%	4,29%
EX94	123	220	151	135	132	78,86%	22,76%	9,76%	7,32%
EX104	164	283	198	184	170	72,56%	20,73%	12,20%	3,66%
Déviation totale moyenne						***60,79%***	***16,61%***	***6,85%***	***3,70%***

2.6.2. Analyse des résultats

Comme le montre le schéma de la Figure 2-11, le makespan d'un système composé d'un seul robot de capacité 1 est très nettement supérieur aux makespans obtenus avec deux robots dans le système. Ceci n'a rien de surprenant.

Par contre on constate qu'avec un robot de capacité 2 les résultats sont très mauvais par rapport aux résultats avec deux robots de capacité 1. Par exemple, sur l'instance EX11 avec le critère C1, on avait trouvé 114 avec deux robots de capacité 1 alors que la meilleure solution trouvée est de 125 avec un robot de capacité 2. Il faut une capacité de 3 au robot pour mettre de trouver des solutions de coûts identiques au cas de deux robots de capacité unitaire.

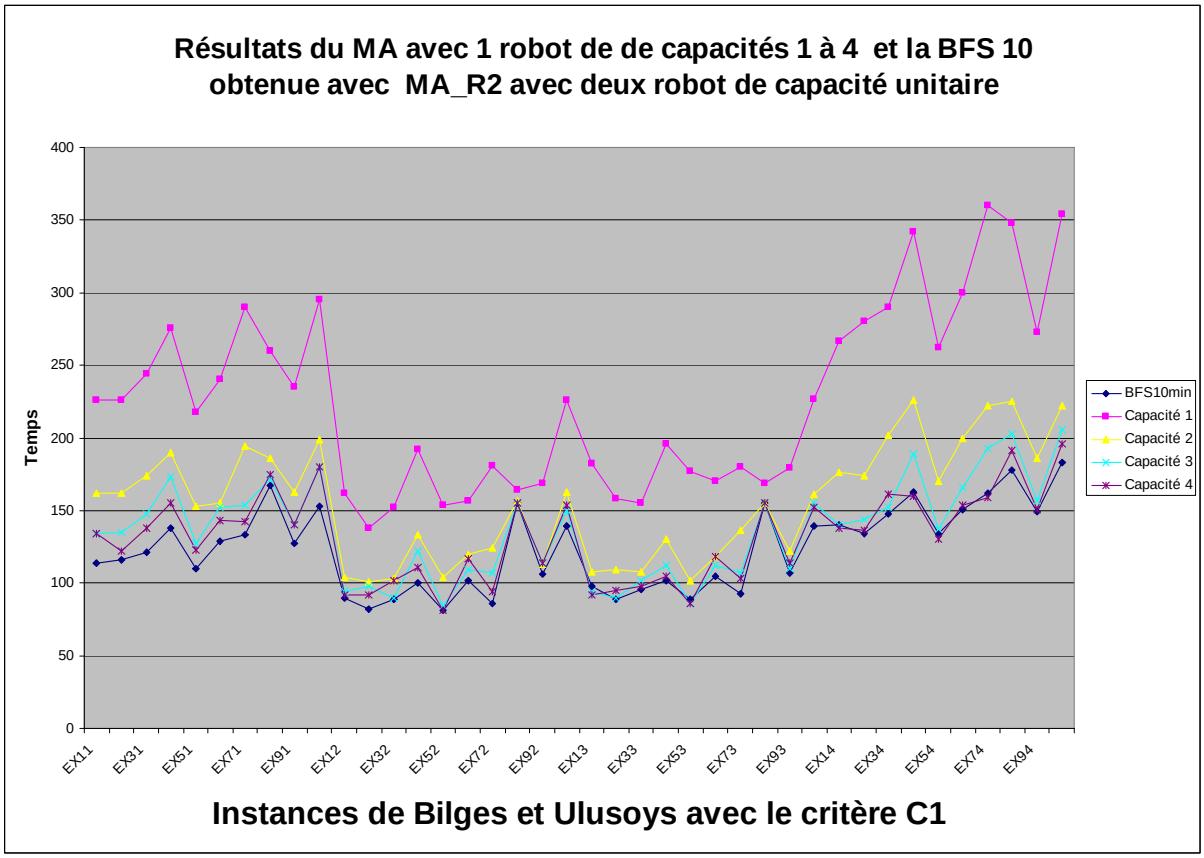


Figure 2-11 : Comparaison des résultats

Les résultats de ces tests montrent qu'un robot de capacité K n'est pas équivalent à K robots de capacité 1, mais par contre, ces résultats confirment que l'augmentation de la capacité du robot diminue le C_{max}, ce qui constitue un résultat très utile dans l'aide à la décision pour le concepteur des SFP. Nous soulignons qu'un robot de capacité 4 a une déviation de **6.32%** par rapport aux meilleurs résultats connus pour les instances de Bilge et Ulusoy en utilisant le critère d'évaluation C1 et de **3.70%** en utilisant le critère C2 sur les mêmes instances. Cette déviation montre qu'on peut substituer un robot de capacité 4 à deux robot de capacité 1. Le fait d'utiliser

un seul robot nous permet d'éliminer les problèmes de gestion des collisions et de faciliiter le pilotage du système de transport/manutention.

2.7. Conclusion

Dans cette partie nous avons essayé d'étendre les modèles existants du job-shop avec transport proposés au chapitre précèdent et ainsi inclure les contraintes liées à la présence d'un robot possédant une capacité de chargement non unitaire. Notre travail a consisté à trouver une nouvelle modélisation prenant en compte les nouvelles contraintes et à proposer une formalisation linéaire en première étape et l'utilisation des méthodes approchées pour résoudre le problème du job-shop avec un robot de capacité non unitaire en seconde étape. Ce travail ouvre un axe de recherche dans le sens où les résultats auxquels nous sommes arrivés constituent une nouvelle étape dans l'optimisation de l'utilisation des moyens de transport dans les ateliers de production. Ce travail est la première étape avant sa généralisation au job-shop avec plusieurs robots possédants des capacités de chargement différentes. Cette extension est traitée dans la partie suivante. Cette modélisation et les approches de résolution, constituent un outil d'aide à la décision pour établir une stratégie de conception ou d'évolutions des ateliers de production nécessitants des moyens de transport. Elle permet ainsi de savoir s'il vaut mieux utiliser ***n*** robots de capacité unitaire ou ***1*** robot de capacité *K* avec k supérieur à ***1***. L'utilisation d'un robot non unitaire permet de s'affranchir des problèmes que génère une flotte de plusieurs robots, ces contraintes sont liées à la gestion des collisions, le routage et la conception même de l'atelier. Si le gain est le même alors il vaut mieux utiliser un seul robot qu'une flotte ne serait ce que pour le volume qu'il occupera.

3. Le Job-shop avec plusieurs robots de capacités non unitaires

3.1. Introduction

Ce chapitre s'inscrit dans une optique d'extension de la modélisation précédente concernant le job-shop avec un robot de capacité non unitaire. Ayant modélisé ce dernier et proposé une approche de résolution, on a voulu l'étendre pour inclure plusieurs robots de capacité non unitaire. Premièrement, nous donnons une modélisation linéaire au problème et nous étendons le graphe disjonctif proposé pour le job-shop avec un robot de capacité non unitaire. Nous terminons par une proposition d'une approche de résolution basée sur les méthodes approchées.

3.2. Formalisation linéaire

Dans cette partie, nous présentons une formalisation linéaire du problème du job-shop avec plusieurs robots possédant des capacités de chargement non unitaire. Nous tenons à signaler que, parallèlement à notre travail et au même moment nous avons trouvé une publication traitant le même problème et proposant une formalisation linéaire et des heuristiques de construction, il s'agit de la modélisation d'Abdelmaguid et Nassef (2010). Dans ce qui suit nous donnons les différentes notations qui vont servir à la définition des contraintes modélisant le problème.

3.2.1. Notations

J : ensemble des jobs; $J = \{1;2;...;n\}$;

M : ensemble des machines $M = \{1;2;...;m\}$;

R : ensemble des robots $R = \{1;2;...;k\}$;

O : ensemble des opérations machines avec o_i^k la $k^{ième}$ opération machine du job i ;

O^f : ensemble des opérations machines qui sont les dernières opérations de chaque job ;

O^1 : ensemble des opérations machine qui sont les premières dans la gamme tel que O_{i1} est la première opération machine du job i ;

T : ensemble des opérations de transport avec τ_{ij} représente le déplacement entre la machine i et la machine j

L : ensemble des opérations de chargement (sortie)

L_i : ensemble des opérations de chargement du job i et $L_i \in L = \bigcup_{i=1}^{n} L_i$ tel que l_i^j soit la j^{ieme} opération de chargement du job

U : ensemble des opérations de déchargement (entrée)

U_i : ensemble des opérations de déchargement du job i et $U_i \in U = \bigcup_{i=1}^{n} U_i$ tel que u_i^j soit la j^{ieme} opération de déchargement du job i

E : ensemble des opérations de chargement et de déchargement $E = L \cup U$

S : ensemble de toutes les opérations du problème $S = O \cup E$

N_i : le nombre d'opérations machine du job i, auquel correspondent $(N_i - 1)$ opérations de chargement et $(N_i - 1)$ d'opération de déchargement.

μ_i : la machine utilisée par l'opération machine $i \in O$

p_i : temps opératoire de l'opération machine $i \in O$

y_i Job de l'opération $i \in S$

$T_{i,j}^{r,c}$: la durée du trajet du robot r entre la machine μ_i et la machine μ_j transportant c jobs

$(i-1)$: le prédécesseur de l'opération i dans le même job

$(i+1)$: successeur de l'opération i dans le même job

(0) : le sommet prédécesseur de tous les sommets du graphe

$(*)$: le sommet final dans le graphe

H : grand entier positif

c_i^{r} nombre de places consommées sur le robot r ou libérées par l'opération i ∈ E

S_i^{r} temps nécessaire pour le robot r pour charger ou de décharger le job de l'opération $i \in E$

C^{r} capacité du robot $r \in R$

ε très petit nombre, $\varepsilon \in]0,1[$ permettant de rendre une inéquation de type $a \prec b$ en $a \leq b\text{-}\varepsilon$ ou de type $a \succ b$ en $a \geq b + \varepsilon$

3.2.2. Variables:

t_i : date de début de traitement de l'opération $i \in S$

$$b_{ij} = \begin{cases} 1 & \text{si l'opération machine } i \text{ est exécutée avant l'opération machine } j \\ 0 & \text{sinon} \end{cases}$$

$$\varphi_{i,j}^{r} = \begin{cases} k & \text{si le flot du robot r entre l'opération } i \in E \text{ et l'opération } j \in E \text{ vaut k} \in [0, C^r + 1] \\ 0 & \text{sinon} \end{cases}$$

$$y_{i,j}^{r,k} = \begin{cases} 1 & \text{si l'opération } j \in E \text{ est réalisée imédiatement après l'opération } i \in E \text{, par le robot } r \in R \text{ en transportant } k \text{ job} \\ 0 & \text{sinon} \end{cases}$$

$$f_i^{\mathrm{r}} = \begin{cases} 1 \text{ si l'opération } i \in E \text{ est réalisée par le robot } r \\ 0 \text{ sinon} \end{cases}$$

Toutes les opérations de S sont rangées d'une manière croissante. Pour y accéder à un élément de S on définit une fonction λ qui renvoie l'indice d'une opération dans S d'un job $i \in J$ et d'un rang k. Par exemple :

- l'indice de l'opération machine o_i^k est donné par $\lambda(o_i^k) = p$;
- l'indice de l'opération de sortie l_i^k (chargement) est donné par $\lambda(l_i^k) = p+1$;
- l'indice de l'opération d'entrée u_i^k (déchargement) est donné par $\lambda(u_i^k) = p+2$.

Les variables $y_{i,j}^{r,c}$ représentent l'exécution successive par le robot r des deux opérations $(i, j) \in E^2$, l'exposant c représente la charge du robot avant l'exécution de l'opération $i \in E$. Pour bien déclarer ces variables on considère les cas suivants :

- $(i \in L \rightarrow j \in L)$ la charge en i pour chaque robot r doit appartenir à l'intervalle $[0, C^r - 2]$;
- $(i \in L \rightarrow j \in U)$ la charge en i pour chaque robot r doit appartenir à l'intervalle $[0, C^r - 1]$ si (i, j) appartiennent au même job ;
- $(i \in L \rightarrow j \in U)$ la charge en i pour chaque robot r doit appartenir à l'intervalle $[1, C^r - 1]$ si (i, j) appartiennent à des jobs différents ;
- $(i \in U \rightarrow j \in L)$ la charge en i pour chaque robot r doit appartenir à l'intervalle $[1, C^r]$;

- $(i \in L \rightarrow j \in L)$ la charge en i pour chaque robot r doit appartenir à l'intervalle $[2, C^r]$.

Le Tableau 3-1 récapitule la variation de la charge du robot selon l'appartenance des opérations de départ et d'arrivée, ainsi que le flux associé.

Tableau 3-1 : Flux circulant entre les opérations transport

Sens de du déplacement	**Le flux**	**La charge**
Opération i➔opération j	$\varphi_{i,j}^r$	$y_{i,j}^{r,c}$
$(i \in L \rightarrow j \in L)$	$[2, C^r - 1]$	$[0, C^r - 2]$
$(i \in L \rightarrow j \in U)(y_i = y_j)$	$[1, C^r - 1]$	$[0, C^r - 1]$
$(i \in L \rightarrow j \in U)(y_i \neq y_j)$	$[1, C^r - 1]$	$[1, C^r - 1]$
$(i \in U \rightarrow j \in U)$	$[2, C^r]$	$[1, C^r]$
$(i \in U \rightarrow j \in L)$	$[1, C^r]$	$[2, C^r]$

3.2.3. Contraintes

Les contraintes sont très similaires à celles du cas avec un seul robot. La différence se situe dans le fait qu'il y a un flot par robot :

- comme il y a plusieurs robots il faut faire en sorte que chaque flot représentant la charge de chaque robot se dirige en sortie vers une et une seule destination.
- au nœud $\{0\}$ on dispose pour chaque robot r d'un flot de valeur $\varphi_{0,i}^r = C^r + 1$ de la même manière au nœud $\{*\}$ chaque robot r restitue un flot $\varphi_{i,*}^r = C^r + 1$.

Contrainte de flot disponible dans le nœud (0) **:** A partir du nœud (0) il sort *R* flots à destination d'une opération de chargement.

$$\sum_{i \in L \cup \{*\}} \varphi_{0,i}^r = C^r + 1 \qquad \forall r \in R \qquad \textbf{(3.1)}$$

Contrainte de flot entrant vers le nœud $(*)$. A partir d'une opération de déchargement on part vers le nœud $(*)$ avec la totalité du flot : ceci doit être vérifié pour tous les flots R.

$$\sum_{i \in U \cup \{0\}} \varphi_{i,*}^r = C^r + 1 \qquad \forall r \in R \qquad \textbf{(3.2)}$$

Contrainte sur les opérations de chargement **:** Dans le cas ou l'opération de transport i est affectée au robot r ($f_i^r = 1$), la contrainte se réécrit $\sum_{j \in E \cup \{0\}} \varphi_{j,i}^r > 1$ ce qui impose qu'il existe une autre

opération de transport j à partir de laquelle il existe un flot du robot r. Pour aller vers une opération de chargement, il faut qu'il y ait suffisamment de place de disponible sur le robot r *i.e.* le flux entrant soit strictement plus grand que 1. Dans le cas où l'opération de transport i n'est pas affectée au robot r, la contrainte est trivialement vérifiée.

$$\sum_{j\in E\cup\{0\}} \varphi^r_{j,i} \geq 1+\varepsilon-(1-f^r_i)H \qquad \forall i\in L\forall j\in E\forall r\in R \qquad \textbf{(4.1)}$$

Contrainte sur les opérations de déchargement **:** Si l'opération de déchargement est affectée au robot r, $f^r_i=1$, la contrainte est active et peut se réécrire : $\sum_{j\in E}\varphi^r_{j,i} < C^r$. Elle implique alors que le robot r transporte au moins une pièce pour se déplacer jusqu'à l'opération de déchargement i.

$$\sum_{j\in E}\varphi^r_{j,i} < C^r-\varepsilon-(f^r_i-1)H \qquad \forall i\in U\forall j\in E\forall r\in R \qquad \textbf{(4.2)}$$

Contrainte de gestion des flux pour les opérations de chargement : Le flux sortant est égal au flux entrant moins la place que va occuper la pièce de l'opération i sur le robot r, c'est-à-dire que $\forall i\in L,\forall r\in R$ on a : $\sum_{j\in E}\varphi^r_{i,j}=\sum_{j\in E}\varphi^r_{i,j}-1$. Comme on dispose de plusieurs robots on transforme cette égalité en deux inégalités qui sont activées uniquement si le flot circule entre (i,j) et ceci pour le robot assurant le transport de *i* vers *j*.

$$\sum_{j\in E}\varphi^r_{i,j} \geq \sum_{l\in E\cup\{0\}}\varphi^r_{l,i}-c^r_i+(f^r_i-1)H \qquad \forall i\in L,\forall r\in R \qquad \textbf{(5.1)}$$

$$\sum_{j\in E}\varphi^r_{i,j} \leq \sum_{l\in E\cup\{0\}}\varphi^r_{l,i}-c^r_i+(1-f^r_i)H \qquad \forall i\in L,\forall r\in R \qquad \textbf{(5.2)}$$

Contrainte de gestion des flux pour les opérations de déchargement : De la même manière que les deux contraintes (5.1) et (5.2) après l'exécution d'une opération de déchargement i, une place supplémentaire est disponible sur le robot ce qui implique qu'on ne peut pas aller vers une opération de déchargement i si le nombre de places disponibles sur le robot est égal à 0. Pour le robot r assurant la séquence (i,j), il faut vérifier la contrainte suivante : $\forall i\in U\forall r\in R\ \sum_{j\in E}\varphi^r_{i,j}=\sum_{l\in E}\varphi^r_{i,l}+1$. On peut transformer cette égalité en deux inégalités actives uniquement lorsque la variable $f^r_i=1$. Ceci permet d'obtenir les deux nouvelles contraintes (5.3) et (5.4).

$$\sum_{j\in E\cup\{*\}}\varphi^r_{i,j} \geq \sum_{l\in E}\varphi^r_{l,i}+1+(f^r_i-1)H \qquad \forall i\in U\forall r\in R \qquad \textbf{(5.3)}$$

$$\sum_{j\in E\cup\{*\}}\varphi^r_{i,j} \leq \sum_{l\in E}\varphi^r_{l,i}+1+(1-f^r_i)H \qquad \forall i\in U\forall r\in R \qquad \textbf{(5.4)}$$

Contraintes de disjonctions transport ajoutant des délais supplémentaires entre les opérations de transport : Ces contraintes assurent que si un flot circule entre deux opérations transport (cela signifie

que les deux opérations sont consécutives), alors les dates de début de ces deux opérations sont distantes d'une durée égale au temps de transport plus la durée de chargement ou de déchargement dépendante du robot et du job. La variable $y_{i,j}^{r,k}$ est égale à 1 si les deux opérations sont réalisées de manière consécutive par le robot r qui transport k pièces. La contrainte se réécrit : $t_j \geq t_i + T_{i,j}^{r,c} + S_i^r$.

$$t_j \geq t_i + T_{i,j}^{r,c} + S_i^r - (1 - y_{i,j}^{r,k}) \times H \qquad \forall i, j \in E, \forall r \in R, \forall k \in [0, C^r] \qquad \textbf{(6)}$$

Relations entre les variables du flot $\varphi_{i,j}^r$ *et la variables binaires* $y_{i,j}^{r,k}$: Cette contrainte a deux rôles. Le premier consiste à imposer que si aucun flot ne circule entre deux opérations i et j, alors on a $\sum_{k=0}^{C^r} y_{i,j}^{r,k} = 0$ c'est-à-dire que toutes les variables $y_{i,j}^{r,k}$ sont nulles. Le deuxième consiste à affecter la valeur 1 à la variable $y_{i,j}^{r,k}$ telle que k est égal à la valeur du flux entre (i, j). Le fait que les valeurs du flux sont différentes selon qu'on part d'un nœud représentant une opération de chargement ou une opération de déchargement on introduit les 6 contraintes suivantes, qui distinguent les 6 cas possibles.

$$\left(C^r + 1\right) y_{0,i}^{r,0} = \varphi_{0,i}^r \qquad \forall i \in L \cup \{0\}, \forall j \in E, \forall r \in R \qquad \textbf{(7.1)}$$

$$\sum_{k=0}^{C^r - c_i^r} (C^r + 1 - c_i^r - k) y_{i,j}^{r,k} = \varphi_{i,j}^r \qquad \forall i \in L \cup \{0\}, \forall j \in U, \forall r \in R / y_i = y_j \qquad \textbf{(7.2)}$$

$$\sum_{k=1}^{C^r - c_i^r} (C^r + 1 - c_i^r - k) y_{i,j}^{r,k} = \varphi_{i,j}^r \qquad \forall i \in L \cup \{0\}, \forall j \in U, \forall r \in R / y_i \neq y_j \qquad \textbf{(7.3)}$$

$$\sum_{k=0}^{C^r - c_i^r} (C^r + 1 - c_i^r - k) y_{i,j}^{r,k} = \varphi_{i,j}^r \qquad \forall i \in L, \forall j \in L, \forall r \in R / y_i \neq y_j \qquad \textbf{(7.4)}$$

$$\sum_{k=1}^{C^r - c_i^r} (C^r + 2 - k) y_{i,j}^{r,k} = \varphi_{i,j}^r \qquad \forall i \in U, \forall j \in L, \forall r \in R \qquad \textbf{(7.5)}$$

$$\sum_{k=1}^{C^r + 1 - c_i^r} (C^r + 2 - k) y_{i,j}^{r,k} = \varphi_{i,j}^r \qquad \forall i \in U, \forall j \in U \cup \{0\}, \forall r \in R / y_i \neq y_j \qquad \textbf{(7.6)}$$

Les contraintes de capacité du robot : $y_{i,j}^{r,c}$ est une variable dont la valeur est égale à 1 si l'opération transport i précède immédiatement l'opération transport j avec un robot r transportant exactement c jobs. Par exemple, $y_{i,j}^{2,1} = 1$, signifie qu'il existe un arc entre l'opération transport i et l'opération transport j, ce qui représente un déplacement du robot 2 en transportant exactement 2 jobs de la machine de départ de l'opération i vers la machine de départ de l'opération j.

Lorsque le robot 2 se déplace, il transporte soit aucune pièce $\left(y_{i,j}^{2,0} = 1\right)$, soit une pièce $\left(y_{i,j}^{2,1} = 1\right)$ soit c pièces $\left(y_{i,j}^{2,c} = 1\right)$. Pour permettre cette unicité de la charge présente sur le

robot on ajoute les contraintes (8.1) et (8.2) qui permettent aussi de s'assurer que chaque nœud transport possède un seul arc entrant et un seul arc sortant. Ainsi l'unicité de la charge est implicitement représentée par le fait de positioner à 1 une seule variable $y_{i,j}^{r,k}$.

Unicité de l'arc sortant d'un nœud transport (une seule destination possible) : Le flot en sortant d'un nœud $i \in E$ n'a qu'une seule destination, le déplacement $(i \to j)$ est exécuté par un robot en transportant une seule charge.

$$\sum_{r}^{R}\sum_{k=0}^{K}\sum_{j\in E\cup\{*\}} y_{i,j}^{r,k} = 1 \qquad \forall i \in E \qquad \textbf{(8.1)}$$

Unicité de l'arc entrant dans d'un nœud transport (une seule source possible) : Le flot entrant vers le nœud $i \in E$, doit provenir d'un seul nœud source $j \in E$. Le déplacement $(j \to i)$ est exécuté par un robot en transportant une seule charge

$$\sum_{r}^{R}\sum_{k=0}^{K}\sum_{j\in E\cup\{0\}} y_{j,i}^{r,k} = 1 \qquad \forall i \in E \qquad \textbf{(8.2)}$$

Les contraintes sur l'affectation des robots aux opérations transport : La contrainte (9.1) assure que chaque opération de transport est affectée à un seul robot, la contrainte (9.2) affecte le même robot à l'opération de chargement et l'opération de déchargement qui suit.

$$\sum_{r}^{R} f_i^r = 1 \qquad \forall i \in E \qquad \textbf{(9.1)}$$

$$f_i^r = f_{i+1}^r \qquad \forall i \in L \forall r \in R \qquad \textbf{(9.2)}$$

Les contraintes (9.3) et (9.4) permettent de renseigner les variables binaires f_i^r en fonction des $y_{i,j}^{r,c}$.

$$f_i^r = \sum_{j\in E\cup\{*\}}\sum_{c=0}^{C^r} y_{i,j}^{r,c} \qquad \forall i \in E \forall r \in R \qquad \textbf{(9.3)}$$

$$f_i^r = \sum_{j\in E\cup\{0\}}\sum_{c=0}^{C^r} y_{j,i}^{r,c} \qquad \forall i \in E \forall r \in R \qquad \textbf{(9.4)}$$

3.2.4. La fonction objectif

La fonction objectif du ***PL*** est similaire à celle pour la modélisation précédente et peut s'écrire ainsi :

$$Min\, C_{\max}$$

$$C_{\max} \geq t_i + p_i \qquad \forall i \in O_f, \qquad \textbf{(10)}$$

Notons que ce PL fait intervenir un plus grand nombre de variables binaires que le programme linéaire précédent et qu'il s'avèrera donc difficile à d'obtenir la solution optimale.

3.3. Modélisation du problème sous forme d'un graphe disjonctif

Le graphe disjonctif proposé au chapitre précédent est facilement extensible, pour représenter le problème du job-shop avec plusieurs robots de capacité non unitaire.

3.3.1. Le graphe disjonctif

Le graphe disjonctif que nous proposons est une extension du graphe disjonctif utilisé pour le job-shop avec transport. Dans ce graphe les sommets modélisent d'une part, les opérations machines, et d'autre part les opérations liées aux chargements et aux déchargements des jobs. Les arêtes sont de trois types : *(i)* les arrêtes conjonctives qui relient les différentes opérations de chaque job et qui expriment les contraintes de précédences ; *(ii)* les arêtes disjonctives qui représentent l'ordre de passage des jobs sur les différentes machines ; *(iii)* les arêtes disjonctives transport qui définissent l'ordre des opérations transport.

3.3.2. Modélisation sous forme de vecteur

Par rapport au problème précédent, une solution définit en plus, une affection des robots aux différentes opérations de transport. Le problème de job-shop avec plusieurs robots possédant des capacités non unitaires peut être représenté par deux vecteurs :

- un vecteur MTS qui comporte autant de cases que d'opération existantes dans le problème, où chaque job apparaît autant de fois qu'il possède d'opérations. Il s'agit du même vecteur que celui utilisé dans le cas à un seul robot.
- un vecteur AR qui contient les affectations des robots aux opérations transports.

La Figure 3-1 donne une illustration de cette modélisation sur l'exemple 3.1 du chapitre précédent en utilisant deux robots. Le robot 1 avec une capacité unitaire et le robot 2 avec une capacité 2.

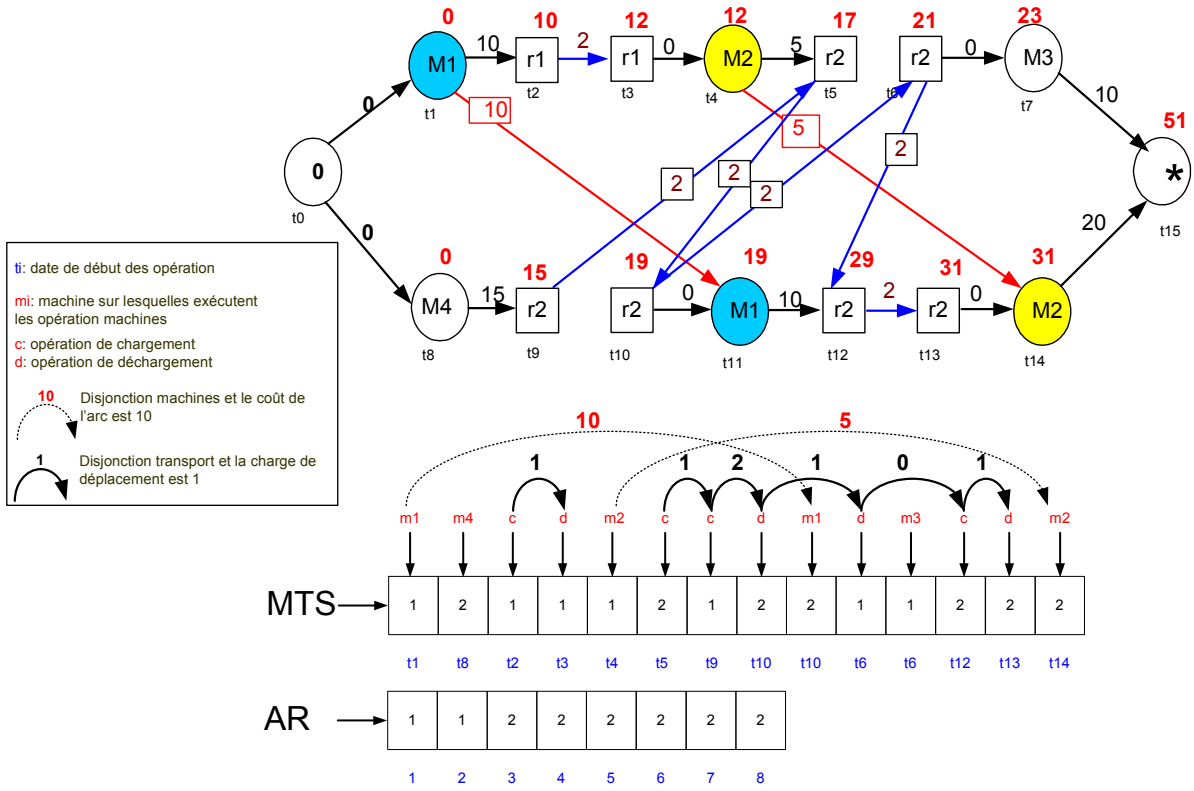


Figure 3-1 : Représentation par vecteur d'une solution de job-shop avec plusieurs robots possédant des capacités non unitaires

Comme pour le cas d'un unique robot de capacité non unitaire, il peut exister des séquences MTS/AR qui ne permettent pas de respecter la capacité des robots.

3.3.3. Recherche Locale

Elle exploite, comme précédemment le chemin critique qui se compose de blocs machine et de blocs transport. Le nombre de robots étant supérieur à 1, on possède un degré de liberté supplémentaire pour casser le chemin critique en modifiant l'affectation des robots aux opérations transport. L'analyse du chemin critique montre l'existence de 8 cas différents qui sont illustrés par la Figure 3-2, ces cas doivent être traités différemment lors du processus de recherche locale.

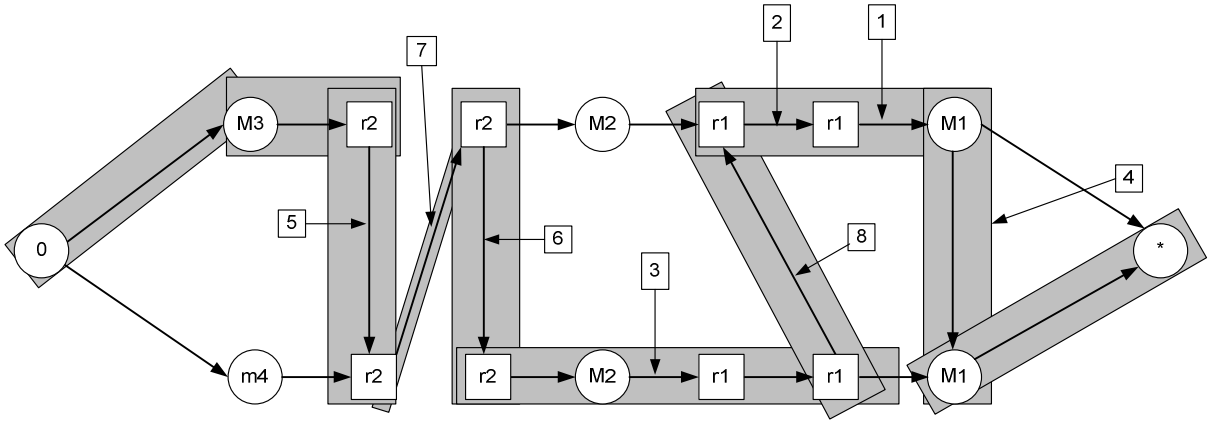


Figure 3-2 : Les 8 cas à prendre en compte lors de processus de la recherche locale

Un arc disjonctif entre deux opérations machine sur le chemin critique ne peut être supprimé qu'en inversant l'ordre des deux opérations machines. C'est le cas, par exemple de l'arc entre les deux opérations machines sur M1 de la Figure 3-2. Il s'agit du cas numéro 4.

Les situations numéro 5, 6, 7 et 8 sont des arcs disjonctifs entre deux opérations transport. Avec un seul robot, la seule solution consistait à inverser l'ordre relatif de ces opérations. Dans le cas à plusieurs robots, on peut aussi modifier l'affection des robots aux opérations de transport. On peut par exemple affecter le robot 1 à la première opération de transport du job 1 faisant ainsi disparaitre l'arc disjonctif entre cette opération et la première opération de transport du job 2.

Les situations 1 et 2 sont particulières dans le sens où si l'on modifie l'affectation d'un robot à une opération de chargement, il faut modifier l'affection du robot à l'opération de déchargement. La remarque reste vraie dans le sens opération de déchargement, opération de chargement. Elles sont aussi particulières dans le sens où la modification de l'affectation des robots peut entrainer une diminution du makespan par la disparation d'arcs disjonctifs mais aussi par la modification de l'évaluation des arcs entre une opération de chargement et une opération de déchargement, et entre une opération de déchargement et une opération machine. En effet dans le cas où la flotte de robot est hétérogène en terme de vitesse de déplacement, modifier l'affection d'un robot, modifie la valuation des arcs.

L'algorithme de cette recherche locale est donné par l'Algorithme 3-1. Dans celui-ci le choix des robots se fait selon une règle r. Cette dernière peut être l'utilisation du robot le plus rapide, où celui possédant la capacité la plus élevé ou bien utiliser le robot le plus économique, sachant que les règles énumérées ci-dessus n'étaient pas notre préoccupation on a choisi lors de l'opération de changement de l'affectation de choisir le robot offrant la date de disponibilité au plus tôt ; ce choix est beaucoup plus guidé par la minimisation du makespan. Sachant que le robot possède une capacité non unitaire, les déplacements ne sont pas directes, c'est pour cela qu'on n'a pas choisi les même règles que celle employés dans le job-shop ave plusieurs robot de capacité unitaire.

Algorithme 3-1**.** Recherche locale

```
Nom de la procédure : Local_Search
       S   : solution donnée par (MTS,AR,PRED, Cmax) ;
       n   : nombre de robots
       K   : vecteur où la case i comporte la capacité maximale du robot i;
       R   : règle de choix et de changement des affectations des robots dans AR
   Début
       i:=N+1                                  // commencer l'analyse à partir du nœud {*}
       iter:=1                                 // initialiser le compteur d'amélioration à 1
       C^best := S.Cmax                        // Initialiser le meilleur coût au coût de la
                                                  solution d'entrée
     Tant que (i≠0) and (iter<=nm) faire
        j:= PRED [i]                          // j le prédécesseur immédiat de l'opération i dans le
                                                 chemin critique
        Si (job[i] ≠ job[j]) alors
           Si (i et j sont deux opérations machines) ou (i et j sont deux opérations
                                                                transport)alors // cas 4,5,6,7,8,
            Save_S :=S
            Permuter les opérations i et j dans MTS
            Corrige(S.MTS,K)
            Evaluer(S.MTS, S.PRED, S.Cmax)
            Si ( S.Cmax < C^best ) alors
                C^best := S.Cmax ;                // amélioration
                i:= PRED [N+1]                    // retour au nœud {*} du nouveau chemin
                                                     critique
                iter :=iter+1
            Sinon
                S:=Save_S
                Si (cas= 4)ou (cas=5)ou (cas=6)ou (cas =7)ou (cas =8) alors
                     Save_S :=S
                     Changer le robot AR selon la règle r
                     Corrige(S.MTS,K) ;
                     Evaluer(S.MTS, S.PRED, S.Cmax);
                     Si ( S.Cmax < C^best ) alors
                            C^best := S.Cmax ;    // amélioration
                            i:= PRED [N+1]
                            iter :=iter+1
                     Sinon
                        i:= PRED(i)               // continuer l'analyse d'ancien chemin
                                                  // critique
                     Fin Si
                Sinon
                     S:=Save_S
                     i:= PRED [i]                 // continuer l'analyse d'ancien chemin
                                                  // critique
                 Fin Si
            Fin Si
        Sinon // (job[i] = job[j])                // deux operation du meme job cas 1,2,3
           Save_S :=S
           Changer le robot AR selon la règle r
           Corrige(S.MTS,K)
           Evaluer(S.MTS, S.PRED, S.Cmax);
           Si ( S.Cmax < C^best ) alors
              C^best := S.Cmax ;                  // amélioration
              i:= PRED [N+1]
              iter :=iter+1
           Sinon
            i:= PRED [i]                          // continuer le parcours du chemin
                                                  // critique
           Fin Si

         Fin Tant que
      Retourner S
   Fin
```

La correction de la séquence consomme beaucoup de temps d'autant plus que les capacités des robots soit petite et le nombre de jobs grand, alors le nombre de vecteurs représentant les

séquences correctes est relativement petit par rapport au nombre de vecteurs représentant des solutions inadmissibles, ceci nous a amené à chercher à développer une indicateur qui permet de privilégier les solutions correctes et de minimiser l'impacte des vecteurs violant la capacité. Nous avons repris le mêm de indicateur retenu (pour plus de détails cf.§2.4 de ce chapitre).

3.4. Algorithme Mémétique

L'algorithme mémétique est l'approche que nous avons retenue pour résoudre le job-shop avec plusieurs robots possédant une capacité non unitaire. Le schéma algorithmique retenu est très proche de celui proposé pour le cas 1 robot de capacité non unitaire (cf Algorithme 2-5).

3.5. Expérimentation numérique

Notre objectif est de mettre en évidence le champ d'application de la méthode. Ainsi, nous testons notre algorithme mémétique sur des instances variées :

- les instances LLT ;
- les instances étendues de Hurink et Knust (2005) ;
- les instances de Bilge et Ulusoy (1995) ;
- les instances de grande taille que nous avons générées ;

Les notations suivantes sont utilisées lors de nos tests :

- *N* : nombre total d'opérations à ordonnancer ;
- *R :* nombre de robots
- *LB*: la borne inférieur introduite dans Hurink et Knust (2005)
- *LB*: la borne inférieur trouvé par l'exécution de notre programme linéaire MILP;
- *BFS*: meilleure solution trouvée (pour best found solution) par notre Algorithme MA_R2 pour les instances de Bilge et Ulusoy en 10 minutes. BFS contient les meilleurs résultats connus pour ces instances;
- *Capacité i*: résultats obtenus avec un robot de capacité *i*.
- *Stop_cr:* critère d'arrêt
- *Dev i %:* déviation en pourcentage de la meilleure solution donnée dans la colonne capacité *i* par rapport à la solution donnée dans la colonne BFS. Le calcul est donné par la formule $\frac{Capacité_i - BFS}{BFS}$

Pour calculer le gain engendré en augmentant la capacité ou le nombre de robots pour montrer l'influence de l'augmentation de la capacité sur le Makespan, nous nous servons des variables suivantes :

- **AMC 1==> 2 :** c'est l'amélioration obtenu en passant des robots ayant la capacité 1 vers des robots de capacité 2.
- **AMC 2==> 3 :** c'est l'amélioration obtenu en passant des robots ayant la capacité 2 vers des robots de capacité 3.
- **AMC 3==> 4 :** c'est l'amélioration obtenu en passant des robots ayant la capacité 3 vers des robots de capacité 4
- **AMC 1==> 4 :** c'est l'amélioration obtenu en passant des robots ayant la capacité 3 vers des robots de capacité 4

- **AMR 1==> 2 :** c'est l'amélioration obtenu en passant de 1 à 2 robots pour les différentes capacités.
- **AMR 2==> 3 :** c'est l'amélioration obtenu en passant de 2 à 3 robots pour les différentes capacités
- **AMR 3==> 4 :** c'est l'amélioration obtenu en passant de 3 à 4 robots pour les différentes capacités
- **AMR 1==> 4 :** c'est l'amélioration obtenu en passant de 1 à 4 robots pour les différentes capacités
- ***AM*** Amélioration moyenne totale

Pour tous les tests nous calculons l'amélioration moyenne, cette dernière nous renseigne sur le gain moyen engendré par l'augmentation du nombre de robots ou l'augmentation de la capacité.

Les paramètres suivants sont utilisés pour l'algorithme mémétique :

- nm: 60 // nombre maximal d'itération de la recherche locale ;
- nc: 115 // nombre de chromosomes dans la population ;
- np: 15000 // nombre d'itérations avant le redémarage;
- pm: 32 // probabilité de la recherché local ;
- pr: 32 // pourcentage de chromosomes à remplacer pendant le restart.

Dans la suite nous présentons une synthèse des résulats obtenus concernant les différentes instances. Dans les tests que nous avons réalisés, une analyse croisée a été faite en fixant la capacité et augmentant le nombre de robot, ensuite en faisant l'inverse c'est-à-dire en fixant le nombre de robots et en augmentant la capacité.

Nous présentons les résultats des différentes expériences numériques. L'analyse des résultats est donnée dans le §2.6.2.

3.5.1. Instances LLT

Le premier test concerne le benchmark LLT composé de 5 instances. Les Tableau 3-2 et Tableau 3-3 donnent les résultats concenant l'amélioration moyenne totale des expérimentations numériques des instances LLT en faisant varier la capacité et le nombre de robots.

Tableau 3-2 : Amelioration moyenne totale : variation du nombre de robots

	Robots			
	AMR 1==>2	**AMR 2==>3**	**AMR 3==>4**	**AMR 1==>4**
Capacité 1	-26,90%	-2,95%	-1,09%	-29,49%
Capacité 2	-3,71%	-0,60%	0,48%	-3,75%
Capacité 3	0,00%	-0,63%	0,00%	-0,63%
Capacité 4	-10,20%	-1,39%	-0,21%	-11,29%
Amélioration moyenne totale	**-10,20%**	**-1,39%**	**-0,21%**	**-11,29%**

Tableau 3-3 : Amelioration moyenne totale : variation de la capacité

	Capacités			
	AMC 1==>2	**AMC 2==>3**	**AMC 3==>4**	**AMC 1==>4**
1 Robot	-30,65%	-4,49%	-0,63%	-33,59%
2 Robot	-8,80%	-0,90%	-0,63%	-10,07%
3 Robot	-6,61%	-0,92%	0,00%	-7,46%
4 Robot	0,00%	0,00%	0,00%	0,00%
Amélioration moyenne totale	**-11,51%**	**-1,58%**	**-0,31%**	**-12,78%**

3.5.2. Instances étendues de Hurink et Knust (2005)

Le deuxième test concerne le benchmark d'Hurink et Knust que nous avons étendu à plusieurs robots de capacité non unitaire. Les Tableau 3-4 et Tableau 3-5 donnent les résultats concenant l'amélioration moyenne totale des expérimentations numériques des instances étendues de Hurink et Knust (2005) en faisant varier la capacité et le nombre de robots.

Tableau 3-4 : Amelioration moyenne totale : variation du nombre de robots

	Robots			
	AMR 1==>2	**AMR 2==>3**	**AMR 3==>4**	**AMR 1==>4**
Capacité 1	-4,31%	-0,23%	-0,26%	-4,91%
Capacité 2	-0,86%	-2,09%	0,82%	-2,19%
Capacité 3	-0,81%	-0,32%	0,09%	-1,15%
Capacité 4	-0,16%	-0,82%	0,27%	-0,78%
Amélioration moyenne totale	**-1,53%**	**-0,87%**	**0,23%**	**-2,26%**

Tableau 3-5 : Amelioration moyenne totale : variation de la capacité

	Capacités			
	AMC 1==>2	**AMC 2==>3**	**AMC 3==>4**	**AMC 1==>4**
1 Robot	-30,65%	-4,49%	-0,63%	-33,59%
2 Robot	-8,80%	-0,90%	-0,63%	-10,07%
3 Robot	-6,61%	-0,92%	0,00%	-7,46%
4 Robot	0,00%	0,00%	0,00%	0,00%
Amélioration moyenne totale	**-11,51%**	**-1,58%**	**-0,31%**	**-12,78%**

3.5.3. Instances de Bilge et Ulusoy (1995) ;

Pour le benchmark de Bilge et Ulusoy nous rappelons les deux critères utilisés pour toutes les instances :

- C1: minimiser le Makespan en prenant en compte la dernière opération de transport de chaque job vers la machine de déchargement U (unload station), ce critère est notamment étudié dans Deroussi et al., (2008) ;
- C2: minimiser le Makespan en ne prenant pas en compte la dernière opération de transport de chaque job vers la machine de déchargement U (unload station);pour lequle plusieurs résultats existent, Bilge et Ulusoy (1993). Bilge et Ulusoy (1995)Bilge et al., (1997) Abdelmaguid et al., (2004).et Deroussi et al., (2008).

Les tableaux 3-6 à Tableau 3-9 donnent les résultats concenant l'amélioration moyenne totale des expérimentations numériques en faisant varier la capacité et le nombre de robots pour les deux critères *C1* et *C2*.

Tableau 3-6 : Amelioration moyenne totale critère *C1* : variation du nombre de robots

	Robots			
	AMR 1==>2	**AMR 2==>3**	**AMR 3==>4**	**AMR 1==>4**
Capacité 1	-30,87%	-11,97%	-2,90%	-30,87%
Capacité 2	-13,55%	-4,12%	-0,59%	-13,55%
Capacité 3	-5,05%	-0,33%	0,05%	-5,05%
Capacité 4	-0,75%	-0,14%	-0,02%	-0,75%
Améliorationmoyenne totale	-12,55%	-4,14%	-0,86%	-12,55%

Tableau 3-7 : Amelioration moyenne totale critère *C1* : variation de la capacité

	Capacités			
	AMC 1==>2	**AMC 2==>3**	**AMC 3==>4**	**AMC 1==>4**
1 Robot	-41,09%	-16,08%	-4,94%	-52,01%
2 Robot	-26,16%	-7,82%	-0,61%	-31,58%
3 Robot	-19,61%	-4,24%	-0,43%	-22,94%
4 Robot	-17,91%	-3,65%	-0,49%	-20,99%
Amélioration moyenne totale	-26,20%	-7,95%	-1,62%	-31,88%

Tableau 3-8 : Amelioration moyenne totale critère *C2* : variation du nombre de robots

	Robots			
	AMR 1==>2	**AMR 2==>3**	**AMR 3==>4**	**AMR 1==>4**
Capacité 1	-26,51%	-8,05%	-2,88%	-34,05%
Capacité 2	-10,40%	-2,30%	-0,34%	-12,60%
Capacité 3	-3,98%	-0,02%	0,05%	-3,95%
Capacité 4	84,96%	-33,43%	-11,63%	-6,61%
Amélioration moyenne totale	20.93%	-10,95%	-3,70%	-14,30%

Tableau 3-9 : Amelioration moyenne totale critère *C2* : variation de la capacité

	Capacités			
	AMC 1==>2	**AMC 2==>3**	**AMC 3==>4**	**AMC 1==>4**
1 Robot	-36,11%	-11,19%	-3,58%	-44,30%
2 Robot	-22,04%	-4,81%	-0,18%	-25,40%
3 Robot	-17,29%	-2,63%	-0,10%	-19,24%
4 Robot	-15,18%	-2,26%	-0,17%	-16,99%
Amélioration moyenne totale	-22,65%	-5,22%	-1,01%	-26,48%

3.5.4. Instances de grande taille

Les benchmarks présents dans la littérature sont relativement petit pour justifier l'intérêt d'utiliser plusieurs robots de capacité non unitaire. En effet, il y a peut de jobs en attente et pour cette raison nous avons réalisé un benchmark composé de 20 instances conçues à partir des instances de Bilge et d'Ulusoy. C'est un benchmark de grande taille comportant au minimum 19 jobs et au maximum 36 jobs, le nombre d'opérations total est alors entre 300 à 600 opérations.

La composition de ce benchmark est la suivante. Nous avons généré cinq ensembles de problèmes appelés SET1 à SET5, en les combinant avec les quatre topologies introduites par Bilge et Ulusoy. Ainsi, nous avons obtenus 20 instances appelées SETnm où n désigne le problème et m la topologie utilisée. Ainsi, les instances sont les suivantes :

- SET1 se compose de : EX1, EX2, EX3, EX4 et EX5 de Bilge et Ulusoy.
- SET2 se compose de : EX6, EX7, EX8, EX9 et EX10 de Bilge et Ulusoy.
- SET3 se compose de : EX1, EX4, EX7, EX8 de Bilge et Ulusoy.
- SET4 se compose de : EX7, EX8 et EX9 de Bilge et Ulusoy

- SET5 se compose de : 6 fois le EX10 de Bilge et Ulusoy.

Tableau 3-10 : Amelioration moyenne totale critère *C1* : variation du nombre de robots

	Robots			
	AMR 1==>2	**AMR 2==>3**	**AMR 3==>4**	**AMR 1==>4**
Capacité 1	-33,08%	-12,93%	-6,18%	-45,35%
Capacité 2	-17,91%	-6,30%	-2,17%	-24,60%
Capacité 3	-8,39%	-2,87%	-2,34%	-12,99%
Capacité 4	-4,91%	-1,51%	-0,26%	-6,52%
Amélioration moyenne totale	-16,07%	-5,90%	-2,73%	-22,36%

Tableau 3-11 : Amelioration moyenne totale critère *C1* : variation de la capacité

	Capacités			
	AMC 1==>2	**AMC 2==>3**	**AMC 3==>4**	**AMC 1==>4**
1 Robot	-48,64%	-27,26%	-13,69%	-67,47%
2 Robot	-36,87%	-18,80%	-10,43%	-53,50%
3 Robot	-31,99%	-15,84%	-9,14%	-47,30%
4 Robot	-29,11%	-15,99%	-7,24%	-44,16%
Amélioration moyenne totale	-36,65%	-19,47%	-10,12%	-53,11%

3.5.5. Analyse des résultats

Ces résultats montrent que sur l'ensemble des benchmarks que notre approches a un comportement cohérent, et confirme nos suppositions de départ à savoir que l'augmentation de la capacité permet de baisser le makespan.

Cependant l'analyse croisée montre aussi qu'on gagne plus en augmentant le nombre de robots que lorsqu'on augmente leurs capacité et ceci est logique, car même si le fait d'avoir une capacité non unitaire permet de supprimer certains déplacement, mais pas tous car le robot ne peut se trouver à deux endroit différents au même moment ce qui le cas de plusieurs robots.

Le gain moyen en augmentant la capacité sur l'ensemble des benchmarks varie de 2% à 22% tandis que le gain engendré par l'augmentation du nombre de robot varie entre 6% à 53% environ, ce qui est considérable. Ces gains diminuent en augmentant le nombre de robots et ceci s'explique par la bonne qualité des solutions proposées et les marges d'amélioration sont minimes.

4. Conclusion du chapitre

Dans dernier chapitre nous avons fait une extension du job-shop avec un robot de capacité non unitaire, en proposant une modélisation mathématique, et un graphe disjonctif permettant de prendre en compte la capacité non unitaire. Cet effort de modélisation est complété par une propostion d'un schéma itératif de type algorithme mémétique permettant de résoudre des problèmes de grandes tailles dans des temps acceptables.

Les résultats obtenus permettent de mesurer l'impact de la capacité de chargement du robot sur la performance du système. Notre travail peut être considéré comme un moyen de dimensionnement lors de processus de conception des ateliers de production utilisant des moyens de transport.

Cette étape est ensuite complété par une étude des problèmes de type job-shop avec plusieurs robots de capacité non unitaire. Nous montrons que notre modélisation linéaire peut être étendue et que le schéma d'optimisation de type algorithme mémétique peut, lui aussi, moyenant des modifications assez mineures, être adapté.

Conclusion générale

Nous nous sommes intéressés dans cette thèse à la modélisation et à la résolution de problèmes difficiles venant du job-shop. Nous avons étudié une extension du job-shop en incluant des contraintes de transport de deux types : gestion d'une flotte de robot de capacité unitaire et gestion d'une flotte de robots de capacité non unitaires.

Dans un premier temps nous avons abordé la résolution du problème du job-shop en tant que problème théorique. Dans un deuxième temps, nous avons traité le cas du job-shop avec transport. Ceci nous a conduits à définir une nouvelle modélisation et une nouvelle méthode d'optimisation. Nous avons fournis des études numériques pour évaluer nos propositions par rapport aux précédentes propositions du domaine. Pour terminer nous nous sommes intéressés à la modélisation et à la résolution d'un problème nouveau à savoir un problème de job-shop avec transport mais dont la flotte de robots est de capacité non unitaire.

Pour chaque problème que nous avons étudié nous avons proposé des modélisations linéaires qui ont été impossible à résoudre dans des temps de calcul raisonnables.

De plus, notre démarche d'optimisation nous a permis d'adapter le modèle de graphe disjonctif-conjonctif et les outils associés de manière à les mettre en œuvre sur des algorithmes itératifs.

L'étude des problèmes de job-shop avec transport nous a permis de mettre en pratique nombre de compétences qui ont été développées dans ce mémoire. Ces compétences ont surtout été appliquées au travers du choix des outils de modélisation et des méthodes utilisées. Ce choix dépend de nombreux facteurs dont principalement la complexité des problèmes, des objectifs visés et du temps dont on dispose pour résoudre les problèmes.

Les perspectives que nous souhaitons donner à ce travail concernent :

1. La proposition de méthodes exactes de résolution ;
2. L'optimisation de systèmes de type job-shop avec transport mais soumis à des aléas ;
3. La définition de nouveau schémas d'optimisation de type multi-objectif afin d'équilibrer la charge de travail entre les robots.

Bibliographie

Aarts et Laarhoven (1985) — Aarts, E. H. L., and V. Laarhoven (1985), *Statistical cooling: A general approach to combinatorial optimization problems*. Philips J. Res, 40(4): 193-226.

Abdelmaguid et al., (2004) — Abdelmaguid, T.F., Nassef, O.N., Kamal, B.A., Hassan, M.F. (2004), *A hybrid GA/heuristic approach to the simultaneous scheduling of machines and automated guided vehicles*. International Journal of Production Research; 42: 267-281.

Abdelmaguid et Nassef (2010) — Abdelmaguid, T.F., and Nassef, O.N. (2010), *A constructive heuristic for the integrated scheduling of machines and multiple-load material handling equipment in job shops*, International Journal of Advanced Manufacturing Technology, 46(9-12): 1239-1251.

Adams et al., (1988) — Adams, J., Balas, E., Zawack, D. (1988), *The shifting bottleneck procedure for job-shop scheduling*. Management Science; 34 (3): 391-401.

Aiex et al., (2003) — Aiex, R.M, Binato, S, Resende, M.G.C. (2003), *Parallel GRASP with path-relinking for job-shop scheduling*. Parallrel Computing; 29: 393-430.

Applegate et Cook (1991) — Applegate, D., and Cook, W. (1991). *A computational study of the job-shop scheduling problem. ORSA Journal on Computing*; 3 (2): 149-156.

Artigues et al., (2003) — Artigues, C., Philippe Michelon, Stéphane Reusser (2003), *Insertion techniques for static and dynamic resource-constrained project scheduling*. European Journal of Operational Research; 149(2): 249-267

Artigues et al., (2005) — Artigues C., Lopez, P. and Ayache P.-D. (2005), *Schedule Generation Schemes for the job-shop Problem with Sequence-Dependent Setup Times: Dominance Properties and Computational Analysis*. Annals of Operations Research; 138: 21-52.

Baker, (1974) — Baker,K.R., (1974), *Introduction to sequencing and scheduling*. Wiley&Sons.

Baptiste (1998) — Baptiste P. (1998), *une étude théorique et expérimentale de la propagation des contraintes de ressources*. Thèse de doctorat, université Technologique de Compiègne N° : 98 COMP 1141.

Bertel (2001) — Bertel, S. (2001). *Problèmes d'ordonnancement de type flowshop hybride avec recirculation et fenêtres de temps*. Thèse de Doctorat. Université de Tours, décembre 2001.

Bilge et Ulusoy (1993) — Ulusoy, G. and Bilge, U. (1993), *Simultaneous Scheduling of Machines and Material Handling System in an FMS*. International Journal of Production Research; 31(12): 2857-2873.

Bilge et Ulusoy (1995) — Bilge, U. and G. Ulusoy (1995), *A Time Window Approach to Simultaneous Scheduling of Machines and Material Handling System in an FMS*. Operations Research; 43(6): 1058-1070.

Bilge et al., (1997) — Ulusoy, G., Sivrikaya-Serfioglu, F. and Bilge, U. (1997), *A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles*. Computer and Operations Research 1997; 24(4): 335-351.

Bierwirth (1995) — Bierwirth C. (1995), *A generalized permutation approach to job-shop scheduling with genetic algorithms*. OR Spektrum; 17: 87-92.

Blazewicz et al., (1996) — J. Blazewicz, W. Domschke E. Pesch (1996), *The job Shop Scheduling Problem : Conventionnal and new solution techniques*. European Journal of Operational Research; 93: 1-33.

Blazewicz et al., (1998) — Blazewicz J. , Sterna M. et Persch E. (1998), *A Branch and Bound algorithm for the Job Shop scheduling problem*. Springer Verlag; :219-254.

Blum et Andrea (2003) — Blum C. and Andrea R. (2003), *Metaheuristics In Combinatorial Optimization: overview And Conceptual Comparaison*. ACM Computing Surveys; 35(3):268-308.

Bonabeau et al., (1996) — Bonabeau, E., Theraulaz, G., and Deneubourg, J.-L. (1996), *Quantitative Study of the Fixed Threshold Model for the Regulation of Divisionof Labour in Insect Societies*. In Proceedings Roy. Soc. London B, (263). References 349.

Bonabeau et al., (1998) — Bonabeau, E., Theraulaz, G., and Deneubourg, J.-L. (1998), *Fixed Response Thresholds and the Regulation of Division of Labor in Insect Societies*. Bulletin of Mathematical Biology; (60):753–807.

Bonabeau et al., (1999) — Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999), *Swarm Intelligence, From Natural to Artificial Systems*. Oxford University Press.

Bonetto (1987) — Roger Bonetto (1987), *les ateliers flexibles de production*. 2eme edition, HERMES.

Brauner et al., (2005) — N. Brauner, P. Castagna, M-L. Espinouse, G. Finke, Ph. Lacomme, P. Martineau, A. Moukrim, A. Soukhal, C. Tacquard, N. Tchernev,

Ordonnancement dans les systèmes flexibles de production. Journal Européen des Systèmes Automatisés; 39(8) : 925-964.

Brooks et White (1965) — Brooks, G.H. and White, C.R., *An algorithm for finding optimal or near optimal solution to the production scheduling problem.* J. Ind. Eng; 16(l): 34440.

Brucker (1988) — Peter Brucker (1988), Wolfgang Meyer: *Scheduling two irregular polygons.* Discrete Applied Mathematics; 20(2): 91-100.

Brucker et al., (1994) — Brucker, P., Jurisch, B. and Sievers, B. (1994), *A branch and bound algorithm for the job-shop scheduling problem.* Discrete Applied Mathematics; 49: 107-127.

Brucker et Knust (2000) — Brucker, P. and Knust, S. (2000), *A linear programming and constraint propagation based lower bound for the RCPSP.* European Journal of Operational Research; 127: 355–362.

Brucker et al, (2003) — Brucker P., Heitmann S. and Hurink J. (2003), *Flow-Shop problems with intermediate buffers.* OR Spectrum; 25: 549-574.

Buzacott et al., (1988) — John A. Buzacott, Kenneth N. Mckay, Frank R. Safayeni (1988), *Job-shop Schedulin Theory : what is relevant.* JSTOR: interfaces; 18(4): 84-90.

Byrkett et al., (1988) — Byrkett, D.L., M.H. Ozden and J.M. Patton (1988), *Integrating flexible manufacturing systems with traditional manufacturing, planning and control.* Production and Inventory Management Journal; 29: 15-21.

Carlier et Chrétienne (1988) — Carlier, J. et Chrétienne, P. (1988), *Problèmes d'ordonnancement : modélisation / complexité / algorithmes.* Edition Masson.

Carlier et Pinson (1989) — Carlier J., E. Pinson. (1989), *A Branch and Bound Method for Solving the Job Shop Problem.* Management Science;:164-176.

Caseau et Laburthe (1995) — Caseau Y. et Laburthe F. (1995), *Improving Branch and Bound for Job Shop Scheduling with Constraint Propagation.*

Caumond (2006) — Caumond A. (2006), Le *problème de job-shop avec contraintes: modélisation et optimisation.* These, Université Blaise Pascal, Clermont-Ferrand, France, 2006. disponible à l'adresse : http://www.isima.fr/~lacomme/doc/TheseAnthony.pdf

Caumond et al., (2008) — Caumond A., Lacomme P. and Tchernev N. (2008), *Memetic Algorithm for the job-shop with time-lags.* Computers and Operations Research; 35(7): 2331-2356.

Caumond et al., (2009) — Caumond, A., Lacomme P., Moukrim A. and Tchernev N. (2009), *A MILP for scheduling problems in an FMS with one vehicle.* European Journal of Operational Research; 199(3): 706-722.

Cook (1971) — Cook S. (1971), *The complexity of theorem-proving procedures.* In Proceedings of the 3rd ACM Symposium on the Theory of Computing, pages. ACM, New York ;: 151-158.

Crama et al., (2000) — Crama Y., Kats V., Van Knludert J. and Levner E. (2000), *Cyclic scheduling in robotic flowshop.* Annals of Operations Research; 96: 97-124.

Croce et al., (1995) — Croce FD., Tadei R., Volta G. (1995), *A genetic algorithm for the job-shop problem.* Computers & Operations Research; 22: 15-24.

Dantzig (1954) — Dantzig G., Fulkerson R, et Johnson S. (1954), *Solution of a Large-Scale Traveling Salesman Problem.* Operations Research; 2: 393-410.

Darwin (1859) — Darwin C. (1859), *On the origin of Species by Means natural Selection or the Preservation of Favored Races in the Struggle for Life.* Murray London.

Dell'amico et Trubian (1993) — Dell'amico M. and Trubian M. (1993), *Applying tabu-search to the job-shop scheduling problem.* Annals of Operations Research; 41:231-252.

Demassey et al., (2002a) — Demassey Sophie, Michelon Philippe, Artigues Christian (2002a), *Bornes inférieures pour le RCPSP : une approche hybride PPC/PL.* 3ème Conférence Internationale de Recherche Opérationnelle (CIRO'02), Marrakech.

Demassey et al., (2002b) — Demassey Sophie, Michelon Philippe, Artigues Christian (2002b), *Bornes inférieures pour le RCPSP : une approche hybride propagation de contraintes - programmation linéaire.* 4ème congrès de la société française de recherche opérationnelle et d'aide à la décision (ROADEF'02), 20-22 février 2002, Paris, Proc;:105-106.

Deroussi et al., (2008) — Deroussi, L., Gourgand, M. and Tchernev, N. (2008), *A simple metaheuristic approach to the simultaneous scheduling of machines and automated guided vehicles.* International Journal of Production Research; 46(8): 2143-2164.

Dréo et al., (2003) — Dréo Johann, Petrowski Alain, Siarry Patrick et Taillard Eric (2003), *Métaheuristique pour l'optimisation difficile.* Edition Eyrolles.

Drozdowski (1996) — M. Drozdowski, *Scheduling multiprocessor tasks – An overview*, European Journal of Operational Research 94 (1996), 215-230.

Egbelu et Tanchoco (1986) — Egbelu, P. J. and Tanchoco, J.M.A. (1986), *Potentials for Bi-Directional Guidepath*

for Automatic Guided Vehicle Based Systems. International Journal of Production Research; 24(5): 1075-1097.

El Fallahi et al., (2008) El Fallahi, A., Prins, C. and Calvo R. W. (2008), *A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem*. Computers & Operations Research; 35(5): 1725-1741.

Feo et Resende (1989) Feo and Resende (1989) , *A probabilistic heuristic for a computationally difficult set covering problem*. Operations Research Letters; 8:67–71.

Feo et Resende (1995) Feo T.A. and Resende M.G.C. (1995) *Greedy randomized adaptive search procedures*. Journal of Global Optimization ; 6:109-133.

Ferber (1995) Ferber Jacques (1995), *Les systèmes multi-agents : vers une intelligence collective*. Dunod.

Festa et al., (2001) Festa P. and Resende M. G. C (2001), *An annotated bibliography. In C. C. Ribeiro and P. Hansen, editors, Essays and Surveys on Metaheuristics*; : 325-367, Kluwer Academic Publishers.

Fisher (1973a) Fisher M.L. (1973), *Optimal solution of scheduling problems using Lagrange multipliers: part I*. Operation Research; 21: 1114-1127.

Fisher (1973b) Fisher M.L. (1973), *Optimal solution of scheduling problems using Lagrange multipliers: part II*. Symposium on the Theory of Scheduling and its Applications. Springer, Berlino.

Fisher et Thompson (1963) Fisher H. e Thompson G.L. (1963), *Probabilistic learing combination of local job-shop scheduling rules*. Industrial Scheduling, Prentice Hall;: 225-251.

Fleury (1995) Fleury, G. (1995), *Application de méthodes stochastiques inspirées du recuit simulé à des problèmes d'ordonnancement*. RAIRO A.P.I.I. (Automatique Productique - Informatique industrielle) ; 29(4–5):445–470.

Fortnow (2009) Fortnow Lance (2009), The *Status of the P versus NP Problem*, disponible à l'adresse: www.cs.uchicago.edu/~fortnow/papers/pnp-cacm.pdf

Garey et Johnson (1979) Garey M., et Johnson D.S. (1979), *A guide to the Theory of the NP-completeness*. Computers and intractability, Freeman New York.

Giffler et Thompson (1960) Giffler, B. et Thompson, G.L. (1960), *Algorithms for solving production scheduling problems*. Operations Research; 8(4): 487-503.

Glover (1986) Golver F. (1986), *Future paths for integer programming and links to artificial intelligence*. Computers & Operations Research; 13:533-549.

Glover et Laguna (1997) Golver F., Laguna M. (1997), *Tabu Search*, Kluwer Academic Publishers, ISBN0-7923-8187-4, Boston.

Goldberg (1989) Goldberg D.E. (1989), Genetic *Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading.

Gourgand (1984) Gourgand, M. (1984), *Outils logiciels pour l'évaluation des performances des systèmes informatiques*. Thèse d'Etat, Université Clermont Ferrand II, France.

Gourgand et Tchernev (1998) Gourgand M. et Tchernev N. (1998), *Un environnement de modélisation du processus logistique industrielle*. Paru dans les actes des deuxièmes rencontres internationales de "La recherche en logistique". Parutions par Nathalie Fabbe-Costes, Christine Roussat ;:539-557.

Grabowski et al., (1996) Grabowski J., Nowicki E. and Zdrzalka S. (1996), *A block approach for single machine scheduling with release dates and due dates*. European Journal of Operations Research; 26: 278–285.

Graham et al., (1979) Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan., A.H.G. (1979), *Optimisation and approximation in deterministic sequencing and scheduling: a survey,* Annals of Discrete Mathematics; 5: 236-287.

Greenberg (1968) Greenberg H. (1968), *A branch-and-bound solution to the general scheduling problem*, Operation. Research; 16: 353-361.

Gunasekaran et al., (1993) Gunasekaran, A., T. Martikainen and P. Yli-Olli (1993), *Flexible Manufacturing Systems: An Investigation for Research and Applications*. European Journal of Operational Research; 66(1):1-26.

Hao et al., (1999) Hao* Jin-Kao, Galinier Philippe et Habib Michel (1999), *Méthaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes*. Revue d'Intelligence Artificielle; 13: 283-324.

Haupt et Haupt (2004) Randly L. Haupt and Sue Ellen Haupt (2004), *Pratical genetic algorithms*. 2nd ed. John Wiley & Son, Inc., New Jersey.

Hentous (1999) Hentous A. (1999), *Contribution au pilotage des systèmes de production de type job-shop*. Thèse de doctorat, INSA Lyon, N° 99 ISAL 0028.

Hurink et Knust (2001) Hurink J. and Knust S. (2001), *Makespan minimization for flow-shop problems with*

transportation times and a single robot. Discrete Applied Mathematics; 112: 199-216.

Hurink et Knust (2002) 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): 181-203.

Hurink et Knust (2005) 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): 99-111.

Holland (1975) Holland, J. (1975), *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

Hölldobler et Wilson (1990) Hölldobler B. and Wilson E. (1990), *The Ants*. Springer Verlag.

Jackson (1956) JACKSON J. R. (1956), An *extension of Johnson's results on job lot scheduling*. Naval Res. Log. Quart; 3: 201-203.

Jain et Meerran (1999) Jain A.S. and Meeran S. (1999), *Deterministic job-shop scheduling: Past, present and future*. European Journal of Operational Research ;113 (2): 390-434.

Johnson et Papadimitriou (1985) Johnson D.S. et Papadimitriou C.A. (1985), *Computational complexity*. in : Lawler E.L., Lenstra J.K., A.H.G. Rinnooy Kan and SHMOYS D.B. (eds.) *The travelling Salsman Problem*, John Wiley & Sons, Chistester;:37-85.

Kaltwasser et al., (1986) Kaltwasser, J., Hercht, A. and R. Lang (1986), *Hierarchical Control of Flexible Manufacturing Systems*, In Proceedings of the IFAC Information Control Problems In Manufacturing Technology Suzdal, USSR ; :37-44.

Kise (1999) Kise H. (1999), *On an automated two-machine flowshop scheduling problem with infinite buffer*. Journal of the Operational Research Society of Japan; 34:354–361.

King et al., (1993) King R.E. and Hodgson T.F. and Chafee W. (1993), *Robot task scheduling in a flexible manufacturing cell*. IIIE Transactions; 25:80–87.

Kirkpatrick et al., (1983) S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi, (1983),*Optimization by Simulated Annealing*, Science;220(4598):671-680

Kochikar et Narendan (1992) V.P. Kochikar and T.T. Narendan (1992), *A framework for assessing the flexibility of manufacturing systems*. International Journal of Production Research; 30 (12): 2873-2895.

Kouvelis et Kim (1992) P. Kouvelis and M. Kim (1992), *Unidirectional Loop Network Layout Problem In Automated Manufacturing Systems*. Operations Research; 40(3): 533-550.

Knust (1999) Knust, S. (1999), *Shop-Scheduling Problems with Transportation*. Ph. D. Thesis, Fachbereich Mathematik /Informatik Universität Osnabrück available on: http://www.mathematik.uni-osnabrueck.de/preprints/shadow/disskn.rdf.html

Krampe et Lucke (1989) Krampe et Lucke (1989), Einfuerung in der Logistik. (Hussverlag).

Kusiak (1988) A. Kusiak (1988), *Artificial Intelligence and CIM Systems*. In A. Kusiak (Eds.), Artificial Inteligence Implication For CIM, (Kempston: IFS (Publications) Ltd, Bedfort MK42 7BT, UK, and Springer-Verlag, New-York): 3-23.

Laarhoven et Aarts, (1987) Van Laarhoven P.J.M. et Aarts E.H.L. (1987), *Simulated Annealing: theory and applications*. Reidel, Dordrecht.

Laarhoven et al., (1992) Laarhoven V. P.J.M, Aarts. E.H.L., Lenstra J.K (1992), *Job shop scheduling by Simulated Annealing*. Operation Research;40(1): 113-125.

Lacomme et al., (2003) Lacomme P., Prins C. et Seveaux M. (2003), *Algorithmes de graphes*. EYROLLES.

Lacomme et al., (2007) Lacomme P., Larabi M. and Tchernev N. (2007), *A disjunctive graph for the job-shop with several robots*. In: MISTA conference, Paris, France.

Lacomme et al., (2008) Lacomme P., Larabi M. and Tchernev N. (2008), *A System Generation Schedules for transportation problem in job-shop environment*. "COSI 2008",Tizi-Ouzou Algérie.

Lacomme et al., (2009a) Lacomme P., Larabi M. and Tchernev N. (2009), *Simultaneous scheduling of machines and automated guided vehicles: graph modelling and resolution*. "IESM'2009 " ; Montreal Canada.

Lacomme et al., (2009b) Lacomme P., Larabi M. and Tchernev N. (2009), *Job-shop avec un transporteur a capacité non unitaire*. ROADEF 2009, Nancy France.

Lacomme et al., (2010) Lacomme P., Larabi M. and Tchernev N. (2010), *Job-shop based framework for simultaneous scheduling of machines and automated guided vehicles*, to appear in International Journal of production Economics

Lageweg et al., (1981) Lawler, Lenstra and Rinnooy Kan (1981), *Computer-Aided Complexity Classification of Deterministic Scheduling Problems*. Report No. BW138, Matematisch Centrum, Amsterdam.

Langston (1987) — Langston M.A. (1987), *Interstage transportation planning in the deterministic flow-shop environment*. Operations Research, 35 :556–56.

Lawrence (1985) — Lawrence D. (1985), *Job-Scheduling with genetic algorithms*. In: First International conference on genetic algorithms, Mahwah, New Jersey;:136-140.

Lenstra et al., (1979) — Lenstra, J.K., and Rinnooy Kan, A.H.G. (1979), *Computational complexity of discrete optimization problems*. Annals of Discrete Mathematics; 4: 121-140.

Lenstra et al., (1981) — Lenstra, J.K., and Rinnooy Kan, A.H.G. (1981), *Complexity of vehicle routing and scheduling problems*. Networks ; 11: 221-227.

Letouzey (2001) — Letouzey A. (2001), *ordonnancement interactif basé sur des indicateurs : Application à la gestion de commandes incertaines et à l'affectation des opérateurs*. Thèse de doctorat, Institut National Polytechnique de Toulouse.

Looveren et al., (1986) — A.J.Van Looveren, L.F. Gelders and L.N. Van Wassehnove (1986), *A review of FMS planning models*. In A. Kusiak, editor, Modelling and design of flexible manufacturing systems, Elsevier Science Publishers.

Lopez et Esquirol (1999) — Lopez P. & Esquirol P (1999), *L'ordonnancement*. Economica, Paris.

Lourenço (1995) — Lourenço, H.R. (1995), *Job-shop scheduling : Computational study of local search and large-step optimization methods*. European Journal of Operational Research; 83: 347-364.

MacCarthy et Liu (1993) — MacCarthy B.L. et Liu J. (1993), *A new classification Scheme For Flexible Manufactoring Systems*. International Journal of Production; 31(2): 299-309.

Maniezzo et Mingozzi (1998) — Mingozzi A. and Maniezzo V. (1998), *An Exact Algorithm For The Resource Constrained Project Scheduling Problem Based on a New Mathematical Formulation*. Management Science; 44: 714-729.

Manne (1960) — Manne A.S. (1960), *On the Job Shop scheduling problem*. Operation Research; 8(1960): 219-223.

Martelo et al., (2000) — Martelo S., Pisinger D. et Toth P. (2000), *New Trends in Exact Algorithms for the 0-1 Knapsac Problème*. European Journal of Operational Research; 123: 325-332.

Matsuo et al., (1988) — Matsuo, H., Suh, C.J., Sullivan, R.S., (1988), *A controlled search simulated annealing method for the general job-shop scheduling problem*. Working Paper, 03-04-88, Graduate School of Business, University of Texas at Austin, Austin, Texas, USA

McMahon et Florian (1975) — McMahon G. and Florian. (1975), *On scheduling with ready times and due dates to minimize maximum lateness*. Operational Research; 23(3): 475-482.

Metropolis et al., (1953) — Metropolis N., Rozenbluth A. adn Rozenbluth M., Teller A. and Teller E. (1953), *Equation os State Calculations by Fast Computing Machines*. Journal of Chemical Physics; 21:1087-1092.

Minsky (1968) — Minsky, M.L (1968). *Matter, mind and models*. MIT press.

Nowicki et Smutnicki (1996) — Nowicki E. and Smutnicki C. (1996), *A fast taboo search algorithm for the job-shop problem*. Management Science; 42(6): 797-813.

Nowicki (1999) — Nowicki E (1999). *The permutation flow-shop with buffers : A tabu search approach*. European Journal of Operational Research; 116 :205–219.

Perregaard et Clausen (1995) — Perregaard M. et Clausen J. (1995), *Parallel branch-and-bound methods for the job-shop scheduling problem.*

Pinson (1995) — Pinson E. (1995), *The job shop scheduling problem: A concise survey and some recent developments*. In: Chrétienne P, Coffman EG, Lenstra JK, Liu Z (Eds.). *Scheduling Theory and Its Applications*. New York: Wiley;: 277-293.

Prins (2009) — Prins C. (2009), *A GRASP Evolutionary Local search Hybrid for the Vehicle Routing Problem*. Springer-Verlag Belin Heidelberg 2009. ISBN 978-3-540-85151-6;: 35-54.

Pham (2008) — Dinh Nguyen PHAM (2008), Phd thesis, *Complex Job Shop Scheduling: Formulation, Algorithms and Healthcare Application*. Faculty of Economics and Social Sciences, Fribourg, Switzerland.

Resende et Ribeiro (2002) — Resende M. G. C. and Ribeiro C. C. (2002), *Greedy randomized adaptive search procedures*. In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics;: 219–249, Kluwer Academic Publishers.

Rinnooy Kan et Timmer (1987a) — Rinnooy Kan A., Timmer G. (1987), *Stochastique global optimization methods.*- parts I : Clustering methods. Mathematical Programming; 39: 27-56.

Rinnooy Kan et Timmer (1987b) — Rinnooy Kan A., Timmer G. (1987b), *Stochastique global optimization methods.*- parts II : Multi level methods. Mathematical Programming; 39: 57-78.

Rogers et White (1990) — Rogers, R. V., and White JR K. (1990), *Job-shop scheduling: limits of the binary*

disjunctive formulation. International journal of production; 28(12): 2187-2200.

Roy et Sussmann (1964) — Roy B. and Sussmann B. (1964), *Les problèmes d'ordonnancement avec contraintes disjonctive*. In : Note DS N°9 bis, SEMA, Paris.

Schmidt, (2001) — Schmidt K. (2001), *Using Tabu Search to Solve the Job Shop Scheduling Problem with Sequence Dependant Setup Times*. Communication.

Schutten., (1998) — Schutten J.M.J. (1998), *Practical job shop scheduling*, Annals of Operations Research. 83(1998): 161-177.

Stecke (1984) — K.E. Stecke (1984), *Design, planning, scheduling and control problems of flexible manufacturing systems*. In Proceednigs of First ORSA/TIMS Special Interest Conference of Flexible Manufactoring Systems.

Storer et al., (1992) — Storer, R.H., Wu, S.D., Vaccari, R., (1992). *New search spaces for sequencing problems with applications to job-shop scheduling*. Management Science; 38 (10): 1495-1509.

Strusevich, (1999) — Strusevich V.A. (1999), *A heuristic for the two-machine open-shop scheduling problem with transportation time*. Discrete Applied Mathematics; 93(2-3): 287-304.

Tchernev (1997) — Tchernev Nicolay. (1997), *modélisation du processus logistique dans les systèmes flexibles de productions*. Gourgand, M.., Ph. D. Thesis, Université Blaise Pascal, Celrmont Ferrand, France, N° ordre D.U. 879, EDSPIC 131.

Timkovsky et Rubinov (1956) — Timkovsky VG. and Rubinov A. (1956), *Non-similarity combinatorial problems*. BioSystems;30: 81-92.

Varela et al., (2002) — Leonilde Rocha Varela, Joaquim Nunes Aparicio, and Silvio Carmo Silva, I. Dans H. Unger, T. Böhme, and A. Mikler (Eds.) (2002): I2CS 2002, LNCS 2346, pp. 63–74, 2002. Springer-Verlag Berlin Heidelberg.

Yamada et Nakano (1992) — Yamada, T. and Nakano, R. (1992), *A genetic algorithm applicable to large-scale job-shop problems*. In: Manner, R., Manderick, B. (Eds.), Proceedings of the Second International Workshop on Parallel Problem Solving from Nature (PPSN'2), Brussels, Belgium;: 281-290.