Contribution à l'ordonnancement d'ateliers avec ressources de transports

Qiao Zhang

Université de Technologie de Belfort-Montbéliard

numéro d'ordre: 183

Contribution à l'ordonnancement d'ateliers avec ressources de transport

THESE

soutenue publiquement le 25 Juillet 2012

pour l'obtention du grade de

Docteur de l'Université de Technologie de Belfort-Montbéliard

(en Automatique)

par

Qiao ZHANG

Composition du jury

Rapporteurs :	M. Pierre CASTAGNA	Prof. IUT de Nantes
	M. Aziz MOUKRIM	Prof. Université de Technologie de Compiègne
Examinateurs :	Mme. Valérie BOTTA-GENOULAZ	Prof. INSA de Lyon
	Mme. Christelle BLOCH	M.d.C Université de Franche-Comté
	Mme. Marie-Ange MANIER	M.d.C. (HDR) UTBM (encadrante)
	M. Hervé MANIER	M.d.C. UTBM (co-encadrant)

IRTES-SET de l'Université de Technologie de Belfort-Montbéliard

Remerciements

Le bon déroulement de ma thèse et la rédaction de ce mémoire n'auraient pu avoir lieu sans le concours de nombreuses personnes que je tiens à remercier :

Marie-Ange MANIER DIRECTRICE DE THÈSE, HDR, Hervé MANIER CO-DIRECTEUR DE THÈSE, MCF, qui m'ont accueillie au sein de l'équipe de recherche de ECS du laboratoire SET et suivie tout au long de la thèse, en donnant leurs conseils, leur disponibilité et leur accueil chaleureux.

Le Pr. Pierre CASTAGNA (IUT de Nantes), et le Pr. Aziz MOUKRIM (Université de Technologie de Compiègne), pour avoir accepté de rapporter ma thèse.

Le Pr. Valérie BOTTA-GENOULAZ (INSA de Lyon), pour avoir accepté le rôle d'examinatrice de ma thèse.

Christelle BLOCH, MCF au département DISC du laboratoire FEMTO-ST, qui m'a aidée à résoudre des problèmes au cours de la thèse et m'a soutenue à distance.

Denise CHOFFEL, MCF à l'ENSTIB, qui m'a contactée et accueillie pour le poste d'ATER.

L'ensemble des doctorants du SeT à l'UTBM, pour leur patience et leur aide.

Un merci tout particulier à Frédéric, Sylvaine, Wafaa, pour leur sympathie ainsi que pour m'avoir supportée et aidée à apprendre la langue française pendant mes études.

Un grand merci à ma famille qui m'a soutenue et encouragée sans relâche durant toutes ces années.

Table des matières

Liste des tableaux

Table des figures

Introduction générale

Dans le contexte industriel actuel où la concurrence fait rage, les entreprises sont amenées à minimiser leurs coûts de production et à s'adapter aux fluctuations du marché. Cette réactivité par rapport à la demande nécessite une flexibilité du système productif permettant aux entreprises de changer rapidement de type de production, sans perte d'efficacité. Le respect des délais est ainsi devenu un facteur de performance qui nécessite une attention soutenue de la part des industriels. Pour atteindre cet objectif, plusieurs leviers d'action sont envisageables : soit constituer des stocks permettant d'anticiper les besoins des clients et de les livrer aux dates souhaitées ; soit de planifier les flux dans les ateliers de production. La première alternative implique des coûts de stockage et des risques d'obsolescence excessifs. Elle n'est en général envisagée que dans des cas spécifiques tels que les productions saisonnières ou à faibles variabilités relativement aux types de produits. L'optimisation des flux est donc le plus souvent la stratégie privilégiée. Elle se traduit par une utilisation efficace de l'ensemble des ressources des systèmes de production : les ressources de traitement, les moyens de stockage et les ressources de transport intervenant aux différentes étapes du processus de fabrication. Autrement dit, il s'agit d'ordonnancer l'ensemble des activités ou tâches à réaliser sur ces ateliers en fonction des gammes opératoires des produits à réaliser.

Les problèmes d'ordonnancement classiques consistent à trouver un ordre et des dates d'exécution pour un ensemble de tâches en minimisant ou maximisant une fonction objectif considerée (comme le *makespan*, la somme totale des temps d'attente...) sous un ensemble de contraintes données. Les problèmes varient en fonction des types d'ateliers et des contraintes de production. Pour les cas classiquement étudiés, les contraintes de transport sont souvent ignorées, ce qui en réduit la complexité. Il n'y a donc pas de temps de déplacement, ni de problème de collision entre les ressources de transport circulant dans la même zone d'un atelier. Toutefois, pour certains types d'ateliers de production, les déplacements des produits entre les ressources de traitement ne peuvent plus être ignorés. Par conséquent, les méthodes dédiées aux problèmes classiques ne fonctionnent plus. Il est donc nécessaire de trouver de nouveaux modèles et méthodes plus adaptés aux cas avec transport.

Par ailleurs et d'un point de vue recherche, si la littérature est très riche en études sur les problèmes d'ordonnancement d'atelier, elle l'est un peu moins pour les ateliers avec

ressources de transport. De plus dans ces derniers cas, les méthodes développées sont principalement dédiées à une configuration donnée de système (par exemple pour des systèmes flexibles avec des AGVs (*Automated Guided Vehicules*)). Et bien que les problèmes soient fondamentalement identiques, ces méthodes ne sont en général pas directement utilisables pour d'autres types d'atelier aux caractéristiques voisines, mais malgré tout différentes. Au final, il y peu de travaux visant à élaborer un modèle générique pour les différents types d'atelier avec contraintes de transport.

D'un point de vue plus pratique, les industriels confrontés à une problématique d'ordonnancement ont besoin de méthodes de résolution efficaces pour leurs ateliers, quelle que soit la nature de ceux-ci. Mais ils ne sont pas forcément en mesure d'analyser/d'identifier à quel type leur problème appartient, ni de faire la démarche de choix de telle ou telle méthode.

Partant de ce double constat, l'objectif de notre travail est d'abord de proposer un modèle générique qui peut s'adapter à divers types d'ateliers et prendre en compte les contraintes de transport, ensuite de développer une ou plusieurs méthodes pour résoudre le modèle proposé. Pour cela, la démarche que nous avons adoptée consiste à :

- identifier, analyser et comparer les classes d'ateliers et leurs caractéristiques, les problèmes d'ordonnancement associés et les méthodes utilisées pour les résoudre ;
- généraliser les données, les variables et les contraintes, afin de proposer un modèle générique dans lequel s'inscrivent les différentes classes de systèmes et qui soit apte à représenter l'ensemble des problèmes d'ordonnancement associés ;
- élaborer une ou des méthodes de résolution pour résoudre le modèle proposé, quel que soit le contexte ;
- valider nos méthodes en confrontant nos résultats avec les meilleurs de la littérature, pour des benchmarks des différents types d'atelier considérés.

Notre travail peut être appréhendé comme une étude de faisabilité, notre objectif étant de montrer que ce que nous proposons est réaliste. Une des finalités à plus long terme est le développement d'un système d'aide à la décision à destination des industriels, qui leur permettra de s'affranchir de cette phase d'identification : de système, de problème associé, et de choix résultant de méthode.

Dans ce mémoire, nous introduisons tout d'abord les ateliers avec ressources de transport (chapitre 1), les problèmes d'ordonnancement associés et l'état de l'art du domaine (chapitre 2). Dans ces deux chapitres, nous définissons les caractéristiques d'un atelier que nous appelons générique, ainsi que le problème d'ordonnancement qui en découle. Celui-ci est une extension du problème de *job shop* classique. Il inclut en particulier la possibilité ou non de stockage et d'attente aux postes de travail, des durées opératoires bornées, et des contraintes liées à la présence de ressources de transport. Dans le chapitre 3, nous proposons deux modélisations de ce problème, l'une mathématique, l'autre sous forme

d'un graphe disjonctif. Le chapitre 4 traite le sous-problème de séquencement des tâches, pour lequel nous avons élaboré deux méthodes de résolution, basées d'une part sur une approche Tabou, d'autre part sur une combinaison d'heuristiques utilisant le graphe disjonctif précédemment défini. Le dernier chapitre introduit le sous-problème d'affectation dans le cas de systèmes flexibles. Pour sa résolution, nous enrichissons les deux algorithmes précédents en utilisant notamment des procédures à base d'algorithme génétique. Au final, nous proposons une méthode de résolution hybride du problème générique, dont nous validons l'efficacité par des tests sur des benchmarks de la littérature représentatifs des diverses classes d'ateliers considérés. Ce mémoire s'achève sur un bilan général et ouvre diverses perspectives à nos travaux.

Chapitre 1

Ateliers de production avec ressources de transport

1.1 Introduction

Dans ce mémoire, nous nous intéressons aux différents types d'ateliers de production avec transport. Nous avons identifié trois systèmes principaux : les systèmes flexibles de production ([Brauner et al., 2005]), les cellules robotisées et les ateliers de traitement de surface. Ces ateliers sont composés d'un ensemble de ressources de traitement, autour duquel s'articule un réseau de transport. Les produits passent donc sur les ressources de traitement et sont déplacés par les ressources de transport. La planification de ces systèmes dépend de leurs caractéristiques et des produits qui y sont réalisés. Ces ateliers se composent généralement de trois sous-systèmes :

- un système de production : défini par un ensemble de ressources de traitement et de stockage, leurs caractéristiques techniques, ainsi que la topologie de ces installations ;
- un système de manutention : défini par les ressources de transport qui s'occupent du déplacement des produits entre les ressources de traitement, la topologie du réseau de transport, et les règles de guidage sur ce réseau ;
- un système décisionnel qui permet de piloter les deux sous-systèmes précédents, en assurant leur synchronisation, tout en respectant l'ensemble de leurs contraintes de fonctionnement.

1.2 Ateliers avec ressources de transport

Dans l'industrie et dans la littérature, nous avons identifié trois principaux types d'ateliers qui comportent des ressources de transport dont les caractéristiques et contraintes ne peuvent pas être ignorées dans les modèles correspondant. Il s'agit des systèmes de production flexibles (SFP ou FMS pour *Flexible Manufacturing Systems*), des cellules robotisées (RC pour *Robotic Cells*), et des ateliers de traitement de surface (ATS), cer-

FIGURE 1.1 – Un atelier de type FMS

tains de ces ateliers étant parfois considérés comme des configurations particulières d'un ou des autres systèmes.

1.2.1 Systèmes flexibles de production

Un système de fabrication flexible ou en anglais *flexible manufacturing system* (FMS), est un système qui se compose d'un ensemble de moyens de production dits flexibles (voir la figure 1.1). Un FMS est donc capable de réaliser des opérations diverses à partir d'un nombre limité de ressources.

Un atelier flexible peut s'adapter à une évolution plus ou moins imprévue de la production, sans changer ou ajouter un nouvel équipement. Il est également capable de gérer une grande variété de pièces sans intervention humaine. Le terme flexible décrit la souplesse du système de production. Il y a en général au moins deux catégories de flexibilité :

- machine flexible (ou polyvalente) : la flexibilité se traduit ici par la capacité d'une machine à effectuer différentes opérations ou différents types de produits. Les machines flexibles sont en général automatisées. Ce sont par exemple des machines outils à commande numérique (CNC : *Computer Numerical Control*) ;
- production flexible (routage flexible) : elle offre la possibilité d'utiliser plusieurs machines pour effectuer la même opération sur une pièce. Elle se traduit aussi par la capacité du système à absorber des changements à grande échelle, comme un changement de volume ou de capacité de production, voire de nature des pièces ou des produits en cours de production, sans besoin d'engager de nouveaux investissements.

Les ressources de transport utilisées dans les FMS sont souvent des chariots autoguidés (AGV pour *Automated Guided Vehicle*). Les AGVs déplacent sans conducteur les produits pour les mouvements horizontaux. Ils ont été introduits en 1955 ([Müller, 1983]). Le nombre de domaines d'application et de variation des types d'AGVs a augmenté sig-

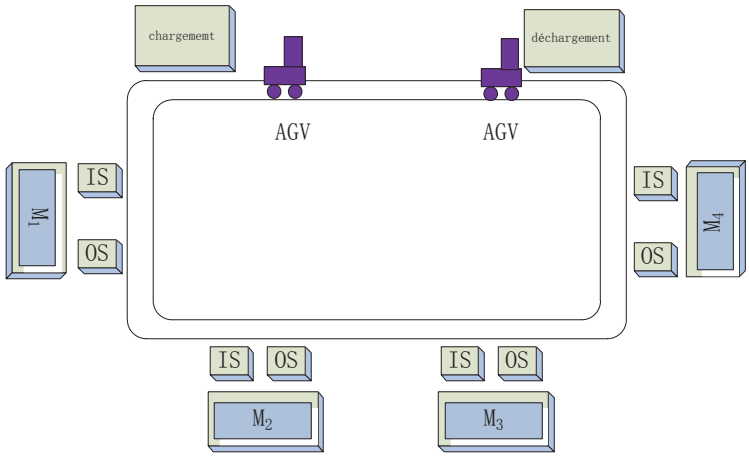

FIGURE 1.2 – Exemple de FMS simple

nificativement.

La figure 1.2 montre une configuration physique pour un FMS simple. Elle est composée d'un poste de chargement/déchargement, deux AGVs, quatre machines à commande numérique, et leurs emplacements de stockage en entrée et sortie (IS/OS) ([Li et al., 2006]).

Les hypothèses généralement adoptées dans un modèle de FMS, et dans un contexte prédictif, sont décrites ci dessous :

- la capacité du poste de chargement et déchargement est illimitée, les deux postes peuvent être identiques ou différents ;
- les ressources de traitement ne peuvent traiter qu'un lot de pièces à la fois ;
- les ressources de transport ne peuvent transporter qu'un lot à la fois ;
- le temps de traitement de la machine comprend les temps de préparation (démontage/montage des outillages et réglage) ;
- les opérations de maintenance et les pannes des machines et des AGV ne sont pas considérées ;
- si une pièce arrive sur une machine qui est occupée, elle peut attendre dans le stock d'entrée de cette machine ;
- si une pièce a fini son traitement sur une machine et qu'il n'y a pas de ressource de transport disponible tout de suite, elle peut attendre dans le stockage de sortie de cette machine.

Pour concevoir et gérer un système flexible, il faut résoudre plusieurs problèmes d'optimisation, en déterminant : l'implantation du système physique, le nombre et la localisation des points de chargement et déchargement,le nombre de véhicules nécessaires, le routage des véhicules, la gestion du trafic, la prévention des collisions, l'ordonnancement des machines et des ressources de transport.

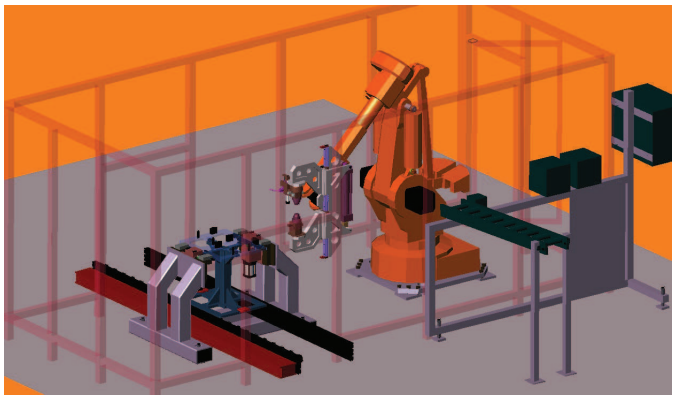

FIGURE 1.3 – Une cellule robotisée

1.2.2 Cellules robotisées

Les robots jouent un rôle important dans les systèmes de fabrication, en particulier dans le système de manutention. Une cellule de fabrication est appelée cellule robotisée quand des robots sont responsables du chargement et du déchargement des pièces sur les ressources de traitement. Dans les cellules robotisées, le robot est fixe, ou se déplace sur des rails pour transporter des pièces entre les machines. Après le chargement d'une pièce sur une machine disponible, le robot peut soit attendre la fin du processus en face de cette machine, ou bien passer à la station d'entrée pour prendre un produit et le charger sur la première machine, ou bien encore se déplacer pour décharger une autre machine. La figure 1.3 montre une cellule robotisée.

Dans un système robotisé basique, il n'y a pas de stock à côté de chaque machine. Par conséquent, à tout moment, une pièce est soit sur une machine, soit sur un robot, soit à une station d'entrée ou de sortie. Pour les cellules robotisées qui autorisent des stocks, une pièce peut aussi être dans un emplacement de stockage (*buffer*).

Les machines utilisées dans ces systèmes sont principalement des machines outils à commande numérique, comme dans les FMS. Elles sont capables d'effectuer plusieurs opérations différentes ([Gultekin et al., 2007]).

Les robots utilisés dans une cellule robotisée sont souvent de capacité unitaire. Certains robots peuvent avoir un double préhenseur, qui permet au robot de décharger une machine par un des préhenseurs et de charger simultanément un autre produit sur la même machine avec le second.

Il y a principalement trois implantations différentes pour les cellules robotisées dans la littérature. Dans la cellule robot-centrée, le robot est fixe et ses mouvements sont des rotations. Dans les cellules robotisées en ligne, le mouvement du robot est linéaire. Enfin, les cellules robot mobile sont une généralisation de la cellule robot-centrée et en ligne ([Gultekin et al., 2007]). Dans des études récentes, de nombreux chercheurs étudient les problèmes de cellule robotisée en ligne. Les hypothèses sont en général les suivantes :

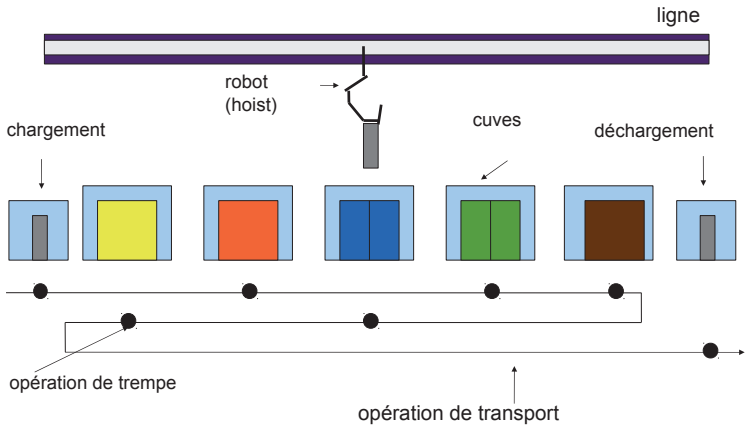

FIGURE 1.4 – Une ligne de traitement de surface ([Manier and Lamrous, 2008])

- les postes de chargement et déchargement ont une capacité infinie ;
- pour les cellules classiques, il n'y pas de stock intermédiaire entre les machines ;
- une machine ne peut traiter qu'une seule pièce à la fois ;
- un robot ne peut transporter qu'une seule pièce à la fois ;
- les opérations de maintenance et les pannes des ressources ne sont pas considérées.

Notons qu'un FMS peut être constitué par plusieurs cellules robotisées desservies par des AGVs ou d'autres moyens de transport.

1.2.3 Ateliers de traitement de surface

Les cellules robotisées concernent principalement l'usinage de pièces mécaniques. D'autres types de systèmes s'intéressent plus particulièrement aux traitements chimiques, ce qui induit des caractéristiques et contraintes liées à ce domaine. On trouve ainsi des ateliers de traitement de surface, qui se composent d'une ou plusieurs lignes. Le long de ces lignes, un ou plusieurs palans de manutention (ou robot ou *hoist*) se déplacent. Les machines sont des cuves contenant des bains chimiques. Les produits sont successivement immergés dans ces bains suivant leur gamme opératoire. Les postes d'entrée et sortie sont assimilés à des cuves fictives utilisées pour le chargement et le déchargement des produits sur des porteurs (cadres, tonneaux, paniers...). En général et dans la suite de ce mémoire, on assimile le porteur au lot de produits ou "travail" associé. Les temps de trempe sont bornés par valeurs inférieure et supérieure, pour garantir la qualité des produits, en réponse à des impératifs d'ordre chimique. Les ressources de transport sont souvent les ressources critiques de ces ateliers. Les figures 1.4 et 1.5 montrent deux ateliers de traitement de surface qui se composent d'une ou plusieurs lignes.

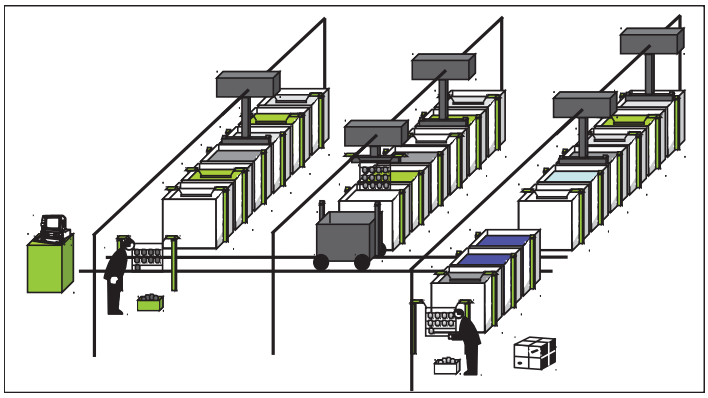

FIGURE 1.5 – Un atelier de traitement de surface avec plusieurs lignes

Les contraintes suivantes doivent être respectées :
- dans chaque cuve, le temps de trempe doit être compris dans un intervalle de temps qui varie selon le produit et le bain chimique ;
- il n'y pas de stock entre les cuves (no-wait) ;
- chaque ressource de transport est de capacité unitaire dans la plupart des cas. Dans certains cas particuliers, elle peut avoir un double préhenseur ;
- le robot ne peut pas s'arrêter quand il transporte un produit (afin d'éviter l'oxydation des pièces) ;
- le robot a besoin de temps pour charger ou décharger un produit, le laisser s'égoutter et/ou se stabiliser avant dépose.

1.3 Un atelier générique avec ressources de transport

1.3.1 Synthèse des caractéristiques des différents ateliers

L'analyse des caractéristiques des ateliers avec ressources de transport, nous a conduit à établir une synthèse basée sur les éléments précédemment définis : la topologie du système, les ressources de traitement, le stockage avant et après chaque machine, et les ressources de transport.

Ressources de traitement

Dans les différents ateliers, la nature des ressources de traitement peut varier de manière importante. En général, on distingue deux principaux types de ressources servant à traiter des pièces : les ressources de transformation à proprement parler (machines, cuves, ...), et les postes de chargement et déchargement en entrée et sortie de l'atelier.

Machines

FIGURE 1.6 – Exemples des ressources de traitement

Nous adopterons le terme général machines pour désigner les ressources sur lesquelles passent les pièces pour réaliser leurs traitements. Il peut s'agir de machines dédiées à un type d'opération ou de pièce, ou de machines flexibles. Par exemple, les cuves sont considérées comme les machines des ateliers de traitement de surface (ATS). Ces ressources peuvent avoir des caractéristiques diverses. La figure 1.6 montre les ressources de traitement dans les trois systèmes considérés.

Dans la plupart des cas, les machines sont de capacité unitaire. Chacune ne peut donc réaliser simultanément qu'une seule tâche. Dans les ateliers de traitement de surface, il existe des cuves dites multi-bacs qui peuvent traiter plusieurs pièces à la fois. En réalité, il s'agit de cuves critiques qui ont été dupliquées pour améliorer la productivité de la ligne. On désigne aussi ces cuves dupliquées sous le terme "bacs". Le nombre maximal de pièces pouvant être traitées simultanément est égal au nombre de bacs de ces cuves. Dans notre atelier, et sans perte de généralité, on assimilera les bacs d'une cuve à des machines parallèles de capacité unitaire.

L'atelier que nous considérons doit être représentatif des trois classes de systèmes identifiés. Pour cela, nous considérons que le temps de traitement sur chaque machine est borné par valeurs inférieure et supérieure. Sur une machine, les durées de traitement doivent donc être incluses dans des fenêtres temporelles. Suivant les cas, la borne maximale de cette fenêtre peut prendre différentes valeurs : celle de la borne inférieure, une valeur finie supérieure à cette borne, ou encore une valeur infinie.

Postes de chargement et déchargement

Dans la plupart des systèmes, les produits passent au préalable par un poste de chargement, parcourent ensuite toutes les machines nécessaires à la réalisation de leurs traitements, et enfin sortent du système par un poste de déchargement. Ces deux postes d'entrée et sortie peuvent être différents (on les dit dissociés), ou être situés au même endroit (associés). Ce peut être le cas par exemple si un unique opérateur est utilisé pour réaliser les tâches de chargement et déchargement. Enfin le temps de chargement et déchargement n'est pas toujours pris en compte dans les problèmes associés. Dans ce cas, il est supposé

égal à zéro.

Ressources de stockage

L'existence d'une activité de stockage dépend des types d'atelier et des ressources de traitement. Pour les ateliers avec stockage, il y a généralement deux configurations possibles : soit le stockage est réalisé directement sur les machines, soit des emplacements (*buffers*) sont prévus entre les machines. Dans le premier cas, il n'y a pas de déplacement entre les tâches de traitement et de stockage réalisées sur la même machine. Mais cela suppose que les produits peuvent rester sur cette machine sans altération de leur qualité.

La comparaison des ressources de stockage est décrite dans le tableau 1.1. La façon de gérer la production et le transport, avec ou sans stock, est différente. En effet, la possibilité de stocker ou non les produits sur l'atelier au cours du processus de production induit les conséquences suivantes :

- En cas d'un emplacement de stockage en aval d'une machine : cette machine peut être libérée dès la fin d'une opération et peut donc commencer plus rapidement l'opération suivante. Cela permet aussi de pallier l'indisponibilité de la ressource de transport intervenant en aval de cette machine ;
- En cas d'un emplacement de stockage en amont d'une machine : inversement au cas précédent, c'est l'indisponibilité de cette machine qui est compensée, et cela permet de libérer la ressource de transport qui lui amène des pièces ;
- Si le stockage est possible sur une machine (sans stock supposé en amont et en aval) : l'indisponibilité de la ressource de transport en aval entraîne un blocage de cette machine. De plus cela peut aussi bloquer la ressource de transport en amont qui amène le produit suivant.
- Si aucun stockage n'est possible, et que la durée opératoire est bornée par valeur maximale (comme par exemple dans les ateliers de traitement de surface) : il n'y a pas d'attente autorisée avant ou après chaque tâche de traitement. Cela implique en particulier qu'à n'importe quel instant, une pièce est soit en cours de traitement sur une machine, soit sur une ressource de transport en direction de la machine suivante dans la gamme opératoire, soit aux postes d'entrée/sortie du système. Cela impose donc la synchronisation des ressources de transport et de traitement pour éviter les situations de blocage.

Il est donc nécessaire d'analyser au préalable la configuration du stockage de l'atelier considéré. Pour ceux qui contiennent des emplacements de stockage spécifiques, la capacité de ces postes est un élément à prendre en compte, puisqu'elle influe sur l'importance des risques de blocage évoquée ci-dessus. En général, cette capacité, si elle est non nulle, est supposée illimitée dans la littérature, c'est-à-dire toujours suffisante.

TABLE 1.1 – Comparaison des caractéristiques de stockage

type d'atelier	existence	capacité	contrainte de temps
FMS	oui	illimitée	illimité
RC	oui/non	illimitée/zéro	illimité/zéro
ATS	non	zéro	zéro

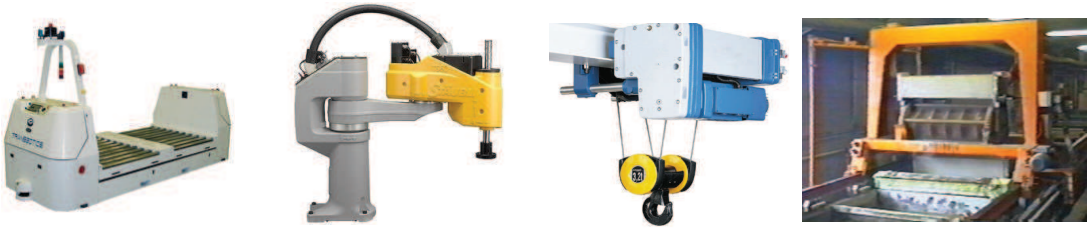

FIGURE 1.7 – Exemples de ressources de transport.

Système de manutention

Le système de manutention comporte deux parties : les ressources de transport et la topologie du réseau associé, à partir desquels on définit les règles de déplacement.

Ressources de transport

Une ressource de transport est un moyen permettant de déplacer une certaine quantité de produits entre deux machines éloignées géographiquement. Suivant les systèmes précédemment identifiés, il y a plusieurs types de ressources de transport : AGV, convoyeur, palan de manutention (*hoist*), robot, chariot, etc. La figure 1.7 montre divers types de ressources de transport qui ont comme point commun d'être automatisées.

Au même titre que les ressources de transformation, les ressources de transport peuvent constituer les ressources critiques dans certains ateliers. Ceci explique que les ressources de transport et les contraintes induites ne peuvent pas être négligées dans les problèmes d'ordonnancement associés, comme c'est le cas dans les problèmes classiques de type *job shop* par exemple.

Le fonctionnement d'une ressource de transport est principalement caractérisé par :

- une capacité par transporteur ;
- une vitesse de déplacement, une accélération et une décélération, permettant de déterminer le temps qu'il faut pour se déplacer entre deux machines.

Dans la plupart des systèmes, la capacité de transport est unitaire. C'est le cas, par exemple, pour le *hoist* dans un atelier de traitement de surface, ou le robot dans une cellule robotisée. Mais il existe aussi des ressources de transport qui sont multi-capacitaires. Un robot avec deux préhenseurs peut prendre deux produits en même temps. Il sert fréquemment à traiter simultanément la sortie d'un produit d'une machine et le chargement d'un

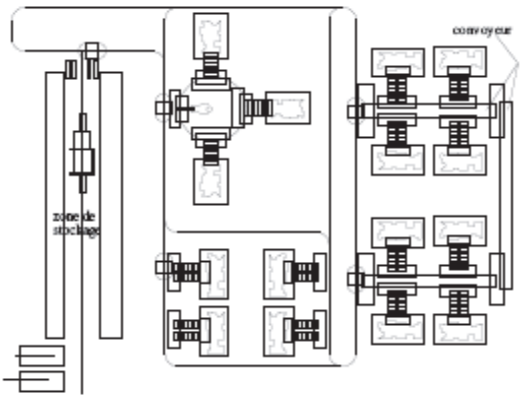

FIGURE 1.8 – La topologie linéaire ([Brauner et al., 2005])

autre produit sur la même machine. Dans un FMS, les AGVs peuvent théoriquement charger plusieurs unités de manutention.

Le temps de déplacement dépend de la distance, de la vitesse du transporteur et des règles de guidage. La distance dépend de la position des ressources de traitement. La vitesse du transporteur peut être indépendante ou dépendante du produit transporté. La vitesse de certains transporteurs peut changer selon le produit chargé (en fonction du type de produit, de sa masse, de son volume, de sa quantité, etc.). Dans la plupart des cas, la vitesse de déplacement est constante. Elle peut aussi varier suivant que le mouvement s'effectue à vide ou en charge. Enfin le temps de chargement/déchargement depuis une machine doit être compté s'il existe, en plus du déplacement effectif. Le plus souvent il est intégré dans le temps de transport.

Topologie et réseau de guidage du système de manutention

Le type de système de manutention utilisé détermine la topologie de l'atelier et le placement des machines ([Heragu and Kusiak, 1988]). Dans la plupart des systèmes de manutention, les ressources de transport sont des ressources automatisées. Dans les ateliers de traitement de surface, le déplacement des ressources de transport est réalisé le long d'une ou plusieurs lignes. Dans les cellules robotisées, le mouvement du robot est souvent circulaire ou linéaire. Dans les FMS, il peut y avoir plusieurs variétés de topologies ([Brauner et al., 2005]) : circulaire, en réseau circulaire unidirectionnel, une topologie comportant plusieurs lignes et colonnes, ou comportant une ou deux ligne(s). En conclusion, les topologies les plus fréquentes sont :

– linéaire : les ressources de traitement sont organisées sur une ou plusieurs ligne(s).
– circulaire : composée d'une ou plusieurs boucle(s).

Les figures 1.8 à 1.10 montrent les configurations les plus fréquentes : le déplacement de ressources de transport sur une ligne (figure 1.8) ; une configuration qui contient un chemin

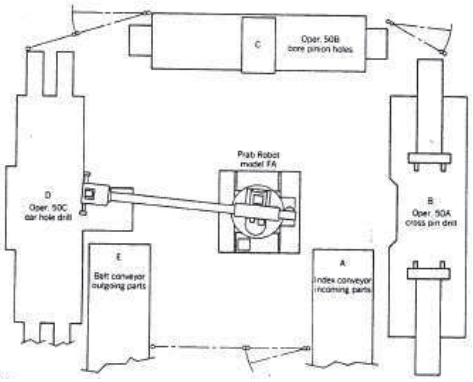

FIGURE 1.9 – La topologie circulaire ([Brauner et al., 2005])

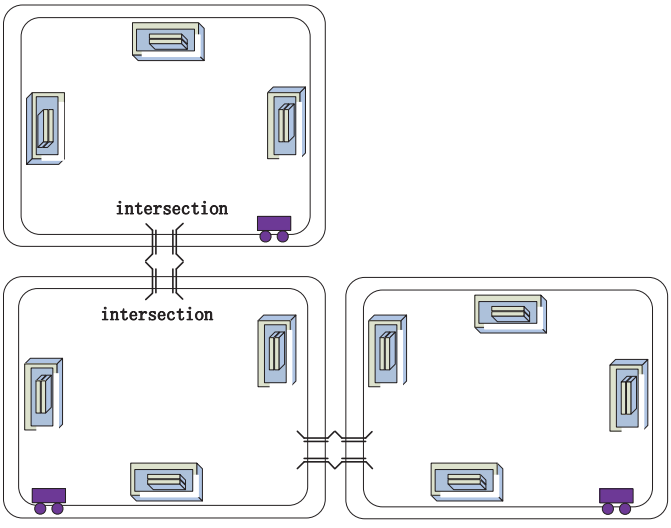

FIGURE 1.10 – La topologie multi-circulaire

TABLE 1.2 – Avantages et inconvénients des différents types de topologies

type	avantages	inconvénients
linéaire et mono boucle	– simple à contrôler – simple à gérer le transport sur la ligne	– routage moins flexible – blocage possible quand une des ressources de transport est en panne – plus long pour charger un nouveau produit – difficile à l'expansion
multi boucles	– simple à contrôler – le problème de collision est diminué si les règles de routage sont bien définies (par exemple, une ressource de transport par boucle) – simple à l'expansion – idéale à appliquer en bi-direction dans les boucles	– stockage supplémentaire souvent nécessaire – nombre de ressources de transport limité par boucle – moins de tolérance aux pannes du système
mixte (conventionnelle)	– routage plus flexible – facile de choisir un chemin alternatif – la distance de transport est plus courte – tolérance aux pannes du système	– difficile à contrôler – blocage et interférences susceptibles de se produire – difficile à l'expansion

circulaire et un système de manutention s'occupant de livraisons entre les machines et la boucle (figure 1.9). Pour les grands ateliers, les configurations des ressources sont souvent plus complexes. Ainsi, la figure 1.2 montre l'implantation d'un FMS simple, alors que la figure 1.10 montre une configuration qui contient plusieurs groupes de machines, chaque groupe de machines est installé dans une boucle. Deux boucles peuvent être connectées à des points d'intersection. Les intersections varient selon les systèmes. Par exemple, une intersection peut être prise en charge par un convoyeur.

Notons que les systèmes flexibles présentent sans doute le plus de diversité en termes de types d'implantation. Ils se composent souvent de plusieurs cellules, dans lesquelles la topologie peut être différente. Une cellule robotisée se compose souvent d'une ligne de traitement ou d'une boucle. Un atelier de traitement de surface contient souvent une ou plusieurs lignes.

Le tableau 1.2 montre les avantages et inconvénients des différentes topologies. Une fois que la topologie physique est décidée, les règles de routage peuvent être définies afin d'optimiser encore la performance du système.

Normalement, les déplacements des ressources de transport en charge s'effectuent sur des chemins prédéfinis et fixés. Il est clair que la configuration du réseau de guidage influe aussi sur la performance du système. Le guidage du réseau de transport doit définir :

- le sens du trafic sur chaque chemin possible dans l'implantation : unidirectionnel ou bidirectionnel ;
- le nombre maximal de ressources de transport simultanément présentes sur la même partie d'un chemin ou tronçon ([Brauner et al., 2005] pour les FMSs) ;

1.3.2 Définition d'un atelier générique

Sur la base de la synthèse des éléments identifiés dans les trois classes d'ateliers considérés, nous décrivons ici les principales caractéristiques physiques du système de production que nous étudions dans la suite de ce mémoire. Cet atelier, que nous qualifions de générique, englobe les caractéristiques communes à chaque système, mais intègre aussi les éléments spécifiques à chacun d'eux. Cet atelier est composé de ressources de traitement, de stockage et de transport, dont la description est détaillée ci-après. Les ressources de traitement sont supposées de capacité unitaire. Les temps opératoires y sont bornés par valeurs inférieure (min) et supérieure (Max). Le temps maximal peut prendre diverses valeurs : celle de la borne inférieure ($Max = min$), ou une valeur fixe strictement supérieure à cette borne ($Max > min$), ou une valeur infinie ($Max = +\infty$). Dans notre atelier, nous considérons qu'il n'y a pas de stockage autorisé sur une machine. En effet, le stockage est évalué à partir du dépassement de la borne supérieure de la durée de traitement. Si cette borne est atteinte, alors cela implique :

- soit que le produit doit être transféré sans délai à la machine suivante ou à l'emplacement de stockage en aval s'il existe ;
- soit que le produit est défectueux, ce qui correspondra à une solution infaisable du problème que nous traitons ici.

Enfin, un atelier peut comprendre plusieurs machines identiques en parallèle, et certaines machines peuvent être flexibles, c'est-à-dire capable de réaliser plusieurs types d'opérations (y compris pour un même produit). Chaque ressource de traitement comporte un emplacement de stockage en amont et en aval. On suppose que leur proximité géographique évite le recours à une opération de transport entre toute machine et ses *buffers*. La capacité de chaque emplacement est soit nulle (ce qui correspond en fait à une absence de stock intermédiaire), soit infinie (en réalité limitée mais suffisante pour absorber la charge dans toutes les situations). Cette configuration permet de prendre en compte l'ensemble des cas identifiés. Les ressources de transport sont caractérisées par :

- une capacité unitaire. Les ressources multi-capacitaires sont souvent utilisées pour transporter un lot de pièces. Nous assimilons en fait un lot de pièces à un produit unique, indissociable au cours du processus ;
- des temps de déplacement en charge et à vide ne dépendant que de la localisation des machines origine et destination des mouvements. Nous ne traitons pas ici le cas

des temps de transport dépendant des produits ;
- les temps de chargement/déchargement des ressources de transport sont inclus dans les temps de déplacement. Nous ne définissons pas ici la topologie du réseau de transport de notre atelier générique, car nous ne traitons pas dans ce mémoire les problèmes de routage et les risques de collisions entre les moyens de manutention.

1.4 Bilan

L'atelier générique, dont nous venons de définir les caractéristiques, nous semble représentatif de la majorité des ateliers de production avec ressources de transport que nous pouvons rencontrer dans la littérature et dans l'industrie, moyennant les hypothèses simplificatrices que nous avons faites. Cet atelier nous servira de base dans la suite de ce mémoire. En particulier, au chapitre suivant, nous définissons le problème d'ordonnancement associé que nous cherchons à résoudre, après avoir proposé un état de l'art du domaine.

Chapitre 2

Ordonnancement des ateliers avec ressources de transport

2.1 Ordonnancement d'atelier

Nous décrivons dans cette section quelques concepts généraux concernant les problèmes d'ordonnancement dits classiques rencontrés dans la littérature.

2.1.1 Concepts généraux

Résoudre un problème d'ordonnancement consiste à programmer l'exécution d'un ensemble de tâches, en leur allouant les ressources requises et en fixant leurs dates d'exécution ([Carlier and Chrétienne, 1988]). De manière plus détaillée, un problème d'ordonnancement est défini par quatre principaux types d'éléments : les tâches à effectuer, les ressources utiles à leur réalisation, les contraintes du système reliant les éléments précédents, et enfin le ou les objectifs à atteindre.

Tâche : c'est l'entité élémentaire d'un travail. En effet, les tâches sont regroupées en travaux. Dans le cas des problèmes d'ordonnancement d'atelier classiques, les travaux sont les *jobs* ou lots de pièces à réaliser. A chaque travail est associé une gamme opératoire qui est une liste ordonnée de tâches (opérations). Un travail est également caractérisé par une date de disponibilité (début au plus tôt) et de livraison (fin au plus tard à respecter, de manière impérative ou non). Chaque tâche est caractérisée par une durée et utilise une ou plusieurs ressources pour son exécution.

Ressource : c'est un moyen technique requis pour la réalisation d'une tâche et disponible en quantité et capacité limitées. On distingue plusieurs types de ressources : les ressources renouvelables et les ressources consommables :

- les ressources renouvelables, redeviennent disponibles en même quantité après avoir été utilisées par une ou plusieurs tâches (machine, robot...) ;
- les ressources consommables (matière première, budget,...), dont la consommation

globale au cours du temps est limitée. Elles ne sont plus disponibles après leur utilisation.

Les ressources renouvelables peuvent être disjonctives (ou non partageables), quand elles ne peuvent exécuter qu'une tâche à la fois (capacité unitaire). Elles peuvent aussi être des ressources cumulatives (ou partageables) si elles peuvent être utilisées par plusieurs tâches simultanément (capacité multiple).

Contraintes : Elles sont relatives aux tâches et/ou aux ressources. Les contraintes les plus couramment rencontrées sont :

- les contraintes potentielles : de succession (ordre liant les opérations d'un même travail) et de localisation temporelle (respect des dates au plus tôt et au plus tard) ;
- les contraintes disjonctives : elles imposent la non réalisation simultanée de deux tâches ;
- les contraintes cumulatives : seul un nombre limité de tâches peut être réalisé simultanément.

Ces deux dernières contraintes sont souvent liées à la capacité des ressources (disjonctives ou cumulatives).

Dans des cas plus complexes, et suivant la nature des ateliers et de la production, d'autres contraintes sont aussi à prendre en compte :

- la préemption/non préemption : elle caractérise la possibilité/interdiction d'interrompre une tâche au cours de sa réalisation ;
- le transport : il faut intégrer les temps de transport non négligeables devant les durées opératoires. De plus, si plusieurs ressources de transport partagent une même zone de déplacement, des risques de collisions existent et induisent des contraintes dites spatiales ;
- les contraintes temporelles particulières, par exemple les écarts minimaux et/ou maximaux entre les tâches (*time lags*, voir [Fondrevelle, 2005]) ;
- etc.

Objectifs et critères d'évaluation

Un objectif s'exprime en termes de minimisation ou de maximisation d'un ou plusieurs critères d'évaluation. Ces critères permettent d'apprécier la qualité des solutions trouvées. Dans les problèmes d'ordonnancement d'atelier, divers critères existent, notamment ceux :

- liés au temps :
  - le temps total de traitement de l'ensemble des tâches. C'est le critère le plus couramment utilisé (makespan). On le note $Cmax$;
  - le retard maximum, moyen ou total par rapport aux dates de fin prévues ;
- liés aux ressources :
  - la quantité maximale ou moyenne de ressources nécessaires pour réaliser un ensemble de tâches ;

  - la charge totale ou moyenne de chaque ressource ;
- liés aux coûts de production, de transport, de stockage, etc.
- ...

Un problème d'ordonnancement peut être décomposé en plusieurs sous-problèmes :
- un sous-problème d'affectation : il consiste à affecter les tâches aux ressources si plusieurs ressources sont capables de traiter la même tâche ;
- un sous-problème de séquencement : sur chaque ressource, il faut ordonner les tâches qui lui sont affectées, et déterminer leurs dates de début ou de fin.

Résoudre un problème d'ordonnancement, c'est donc trouver la solution qui optimise un ou plusieurs critères, tout en satisfaisant l'ensemble des contraintes du système.

2.1.2 Classification des problèmes d'ordonnancement d'atelier

On distingue plusieurs types de problèmes d'ordonnancement dits d'atelier. La figure 2.1 trie les problèmes par ressources ([Esquirol et al., 1999]). La plupart des problèmes étudiés comporte des ressources multiples, renouvelables et disjonctives. En se basant sur la nature des travaux à réaliser, on distingue 4 types de problèmes classiques :
- à une machine : chaque gamme ne comporte qu'une seule opération réalisable sur une seule ressource de traitement ou sur l'une des machines en parallèle disponibles ;
- *flow shop* : le flux des produits sur l'atelier est unidirectionnel car les produits suivent le même ordre de visite des ressources de traitement ;
- *job shop* : le flux est fonction des gammes opératoires des *jobs*. Ainsi, tous les travaux n'ont pas le même ordre de visite des ressources ;
- *open shop* : contrairement aux deux cas précédents, les gammes des *jobs* sont non linéaires. Il n'y a donc pas de contrainte sur l'ordre d'exécution des opérations d'un même travail.

A partir de ces problèmes classiques, des variantes existent, en fonction de la nature des ressources et des tâches, qui induisent des contraintes additionnelles telles que celles décrites précédemment. Des exemples de ces variantes sont :
- le *flow shop* hybride : à un "étage", plusieurs machines sont capables de réaliser une même opération. Cela implique un problème d'affectation en plus du séquencement ;
- le *job shop* généralisé : avec machines polyvalentes ;
- le *flow shop/job shop* avec *time lags* ;
- le *flow shop/job shop* avec transport, etc.

Plusieurs notations associées aux problèmes d'ordonnancement ont été proposées. Celle la plus utilisée correspond à trois champs : $\alpha|\beta|\gamma$. α décrit le type de problème étudié, β décrit les caractéristiques du système physique, γ représente le critère à optimiser ([Graham et al., 1979]). [Lacomme et al., 2010] ont utilisé ce type de notation pour caractériser les problèmes qu'ils ont étudiés dans les FMS : $JR|t_{kl}, t'_{kl}|C_{max}$, qui

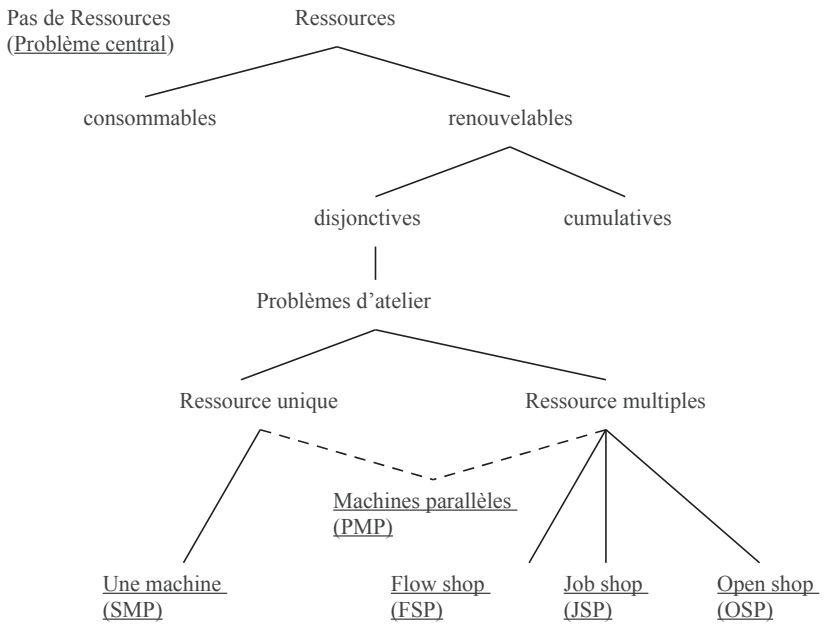

FIGURE 2.1 – Typologie des problèmes d'ordonnancement en fonction des ressources

veut dire que ces sont des problèmes du type *job shop* (J), avec nombre de véhicules limité (R), des temps de déplacement en charge (t_{kl}) indépendants des produits, et des temps de déplacement (t'_{kl}) à vide dépendants des machines. [Tacquard and Martineau, 2001] ont proposé une notation générique pour les problèmes associés aux FMS, basée sur une structure $\alpha|\beta|\gamma$. Elle est capable de représenter non seulement les éléments basiques mais aussi les structures complexes dans les systèmes de production flexibles. Mais ce type de notation n'est pas suffisant pour représenter toutes les contraintes rencontrées, par exemple dans les ateliers de traitement de surface (ATS). Par exemple, [Manier and Bloch, 2003] ont proposé une notation dédiée aux différents types de *Hoist Scheduling Problems* (HSPs) dans les ATSs, composée de quatre champs : $\alpha|\beta|\delta|\gamma$. Cette notation se présente sous la forme suivante : $XHSP|nl, ntransfer, synchro, (mh,-mt, ct)_{i=1tonl}/nc, circ, ret, empty/load-unload|nparts/nps, nop, clean, recrc|criterion$.

Concernant les caractéristiques des problèmes rencontrés dans les ateliers, une classification (inspirée de [Dawande et al., 2005]) et basée sur les trois champs : α, β et γ, est montrée dans la figure 2.2.

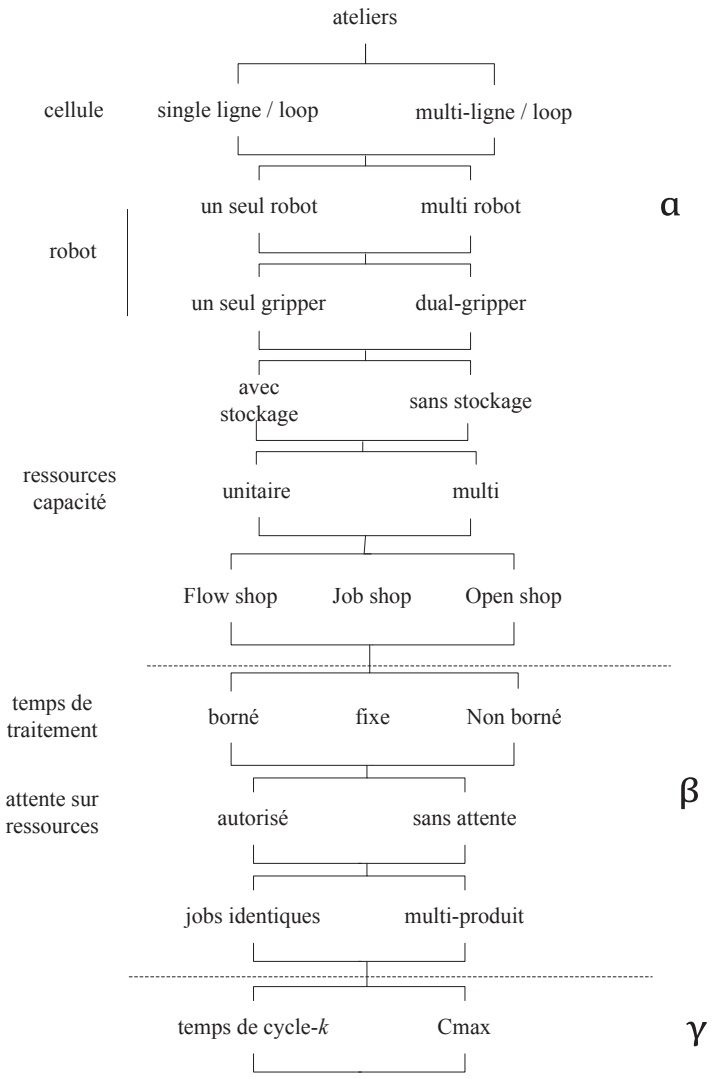

FIGURE 2.2 – Classification des problèmes rencontrés dans les ateliers (inspiré de [Dawande et al., 2005]).

2.2 État de l'art en ordonnancement des ateliers avec ressources de transport

La plupart des problèmes d'ordonnancement d'atelier dits classiques sont déjà NP-complets. Le plus souvent, leurs variantes le sont également. En particulier, l'ordonnancement des ateliers avec contraintes de transport est plus compliqué que l'ordonnancement classique (NP-complet, [Lacomme et al., 2010]), puisqu'il faut ordonnancer à la fois les tâches opératoires et les tâches de transport. Ceci explique que des méthodes exactes ne sont souvent mises en œuvre que pour résoudre des instances de petite taille ou des cas particuliers. On trouve plus souvent dans la littérature des algorithmes basés sur des méthodes approchées. Les problèmes d'ordonnancement peuvent être résolus de manière statique, dynamique ou une combinaison des deux. Nous traitons des problèmes hors lignes dans ce mémoire.

La suite de ce chapitre présente les travaux relatifs aux ateliers avec transport. Les publications sont rassemblées en fonction du type de problèmes étudiés. A l'intérieur de chacun des paragraphes ainsi constitués, les différentes approches proposées sont regroupées par auteurs (ou équipes de recherche) et présentées dans l'ordre chronologique. L'ensemble des descriptions des caractéristiques des problèmes pour chaque type d'atelier est présenté dans les figures à la fin de chaque partie.

Dans les ateliers de production, il y a principalement trois types de problèmes à résoudre : la conception du système, le problème de routage des véhicules (AGVs) et l'ordonnancement des ressources (de traitement et de transport). Les objectifs considérés sont souvent les suivants :

- maximiser le débit du système (par exemple, le nombre de charges de manutention par unité de temps)s ;
- minimiser le temps nécessaire pour traiter tous les travaux (c'est à dire le *makespan*) ;
- minimiser les temps de déplacement du véhicule (vide et/ou en charge) ;
- répartir la charge sur les ressources de transport ;
- minimiser les coûts totaux des mouvements ;
- minimiser les écarts entre le temps de réalisation du travail et sa date de fin au plus tard (c'est-à-dire le retard) ;
- minimiser le temps de déplacement maximal ou moyen des ressources de transport pour se déplacer vers un nouveau travail ;
- minimiser les temps d'attente au poste de chargement ;
- etc.

Les problèmes d'ordonnancement auxquels nous nous intéressons dans ce mémoire sont de type *job shop*, donc avec des gammes linéaires différentes par type de produit. Dans l'état de l'art qui suit, nous nous intéresserons particulièrement aux *job shops* statiques, même si nous évoquons les autres problèmes rencontrés dans les ateliers avec transport.

Ci-dessous nous avons classé les problèmes d'ordonnancement avec transport en deux

types principaux : avec stockage et sans stockage.

2.2.1 Problèmes avec stock

Ce type de problème existe principalement dans les systèmes de production classiques ou flexibles, ou certaines cellules robotisées ([Finke et al., 1996, Dawande et al., 2005]). La capacité de stockage est souvent supposée infinie. Cette classe peut être divisée en deux, suivant la présence ou non de problèmes d'affectation.

- sans affectation des tâches de traitement

Ce type de problème de *job shop* avec transport consiste à ordonnancer à la fois les machines et les ressources de transport. Il ressemble à *job shop* avec retard entre deux opérations successives ([Hurink and Knust, 2002]). Quand le nombre de ressources de transport est limité, les temps de retard ne sont plus fixes, d'où une complexité accrue de résolution de ce problème.

[Pundit and Palekar, 1990] ont appliqué une procédure de *branch and bound* et également une heuristique pour ordonnancer simultanément les machines et les ressources de manutention pour un cas de *job shop*. [Sabuncuoglu and Hommertzheim, 1992] ont traité un problème *job shop* avec transport en séquençant une seule tâche à la fois. Ces deux méthodes de résolution ne prennent pas en compte les contraintes de précédence entre *jobs*. [Bilge and Ulusoy, 1995] ont formulé le problème d'ordonnancement des machines et des AGVs comme une formulation MIP (*Mixed Integer Programming*). Ils ont proposé une heuristique pour la résolution. Elle résout itérativement deux problèmes d'ordonnancement (machines et AGV) jusqu'à ce qu'un résultat suffisamment bon soit obtenu. [Billaut et al., 1997] ont traité un cas spécial de FMS avec une topologie en une seule boucle. Il y a un nombre suffisant de véhicules entre deux machines successives, et les stockages en amont et en aval des machines sont supposés illimités. Le problème du *job shop* avec transport est alors transformé en flowshop hybride. Un ordre de visite de référence est défini sur l'ensemble des machines. Le cheminement de chaque *job* sur l'atelier est transformé en la réalisation d'une ou plusieurs fois de cet ordre de visite de référence, en ajoutant des pauses fictives sur les machines non physiquement visitées (le temps de traitement sur ces machines est de zéro). Dans ce cas, les *jobs* qui ont un ordre de visite différent de celui de référence peuvent être découpés en plusieurs sous-*jobs*. Dans ce nouvel ensemble de *jobs*, chacun a une séquence de traitements identique. Le problème devient donc un problème de *flow shop* avec transport. Cette transformation n'est valable que pour une topologie avec une seule boucle et au moins un véhicule entre deux machines successives, ce qui n'est pas toujours le cas réel. [Ulusoy et al., 1997] ont amélioré les solutions de ce type de problèmes en utilisant un algorithme génétique. Le codage du chromosome représente le séquencement des tâches opératoires et l'affectation des ressources de transport. Le séquencement des tâches de transport est déduit depuis le chromosome. Une

réparation simple est proposée si une solution non faisable est générée en échangeant les tâches opératoires qui violent la contrainte de précédence. Ce type de codage ne peut pas être étendu pour le problème d'affectation de ressource de traitement. [Anwar and Nagi, 1998] ont traité un problème d'ordonnancement dans des cellules de FMS en prenant en compte les transports entre différentes cellules et la contraintes de précédence entre *jobs* définies par les nomenclature. Les AGVs qui se déplacent entre les cellules sont considérés comme des machines supplémentaires. Un algorithme séquence à la fois les tâches opératoires et de transport par une propagation arrière (comme le MRP). La façon de déduire la date de disponibilité des AGVs dépend cependant de l'affectation des tâches opératoires qui doit être fixée à l'avance. De plus, la date de fin au plus tard pour le dernier nœud du graphe est égal à la longueur du plus long chemin du graphe initial où les disjonctions des ressources ne sont pas orientées. Cette date est une borne minimale du *makespan* avec pour hypothèse la capacité infinie des ressources, qui n'assure pas d'obtenir une solution toujours faisable. [Hurink and Knust, 2002] ont proposé deux méthodes de voisinage pour un problème de *job shop* avec ressources de transport identiques. Le problème est représenté par un graphe disjonctif avec des nœuds opératoires et des nœuds de transport. La construction du voisinage est basée sur la conception d'un bloc de tâches critiques proposée par [Brucker et al., 1994]. Le premier voisinage permet, soit de changer le séquencement des tâches opératoires, soit de changer l'affectation et le séquencement des tâches de transport à chaque étape. Le deuxième voisinage permet de faire tous les types de modifications. Une réduction des deux voisinages (inspirée de [Gambardella and Mastrolilli, 1996]) est appliquée. [Hurink and Knust, 2005] ont considéré le problème du *job shop* dans une cellule avec un seul robot. L'objectif est de minimiser le *makespan*. Le problème est présenté par un graphe disjonctif. Les nœuds représentent soit une tâche de traitement soit une tâche de transport. Les temps de transport en charge et à vide sont pris en compte. Une procédure de recherche locale est proposée. Une comparaison est montrée entre trois méthodes : une méthode en une phase qui consiste à changer une tâche de traitement ou de transport pour chaque itération ; une méthode en deux phases où, tout d'abord, une opération est changée ; ensuite, le séquence du robot est ensuite optimisée selon les nouvelles séquences sur les machines. La première méthode est efficace quand le temps d'exécution est limité. La troisième méthode est performante quand le temps de calcul est plus long. [Caumond et al., 2009] ont proposé une formulation mathématique pour trouver les solutions optimales des instances de [Bilge and Ulusoy, 1995] modifiées (avec un seul véhicule, une topologie de transport, et quatre ensembles de *jobs*). Cette formulation prend en compte le nombre maximum de *jobs* présents simultanément dans le système et les capacités de stockages en amont et en aval. Différentes heuristiques sont testées : FIFO, SPT, STT et MOQS. Pour résoudre les instances proposées par [Bilge and Ulusoy, 1995], [Deroussi et al., 2008] ont proposé une méthode hybride : le recuit simulé (*simulated annealing*) avec recherche locale itérative (*iterated local search*). La méthode de recherche locale itérative commence à partir d'une solution initiale générée par une

heuristique déterministe. À chaque itération, une solution est générée en appliquant trois mouvements d'échange à partir de la solution courante. Les voisins de cette solution sont explorés. L'acceptation de la meilleure solution pour remplacer la solution courante est déterminée par la procédure de recuit simulé. Le voisinage est construit par deux mouvements basiques : échange et insertion. Une réduction de voisinage est appliquée : une tâche de transport est d'abord sélectionnée, au lieu d'échanger avec n'importe quelle tâche de transport, la deuxième tâche est choisie parmi un ensemble de tâches qui sont définies dans l'intervalle des dates de fin des tâches de transport précédente et suivante. [Subbaiah et al., 2009] ont utilisé un algorithme évolutionnaire avec *sheep flock heredity* pour traiter les instances de [Bilge and Ulusoy, 1995]. Le codage du chromosome représente l'ordre total des tâches opératoires. Chaque chromosome est classé en plusieurs sous-chromosomes de gênes successifs. Les opérateurs de croisement et de mutation sont appliqués à deux niveaux : sur le chromosome entier ou sur un sous-chromosome. Une réparation est appliquée aux chromosomes non valides qui ont violé la contrainte de précédence. La performance de cette méthode est difficile à évaluer, parce qu'il y a trop de variations par rapport aux meilleurs résultats de la littérature. De plus, l'affectation des AGVs n'est pas détaillée. [Lacomme et al., 2010] ont proposé un algorithme mimétique pour résoudre les problèmes d'ordonnancement dans un FMS sans affectation des tâches opératoires et avec une ou plusieurs ressources de transport. Un graphe disjonctif est utilisé pour représenter le problème associé. Il contient des sommets associés à l'ensemble des tâches (opératoire et de transport) et des valeurs fixes sur les arcs. Le codage contient deux chaînes : une chaîne de sélection des ressources pour chaque tâche et une chaîne de séquencement des tâches de transport. La première chaîne est générée aléatoirement. La deuxième chaîne est générée par une heuristique proposée par Giffler and Thompson en 1960 ([Giffler and Thompson, 1960]), basée sur l'affectation définie par la première chaîne. Le croisement ne modifie que la première chaîne. La mutation se base sur une recherche locale en réaffectant une tâche de transport critique ou en permutant deux tâches critiques sur la même ressource de traitement ou de transport. Cette technique de construction du voisinage est inspirée de [Hurink and Knust, 2005]. Elle est efficace pour améliorer la solution courante par rapport au critère du *makespan*. Cependant, dans les cas où les temps de traitement sont bornés, la permutation des tâches peut générer une solution infaisable. [Larabi, 2010] ont considéré le problème de *job shop* avec plusieurs robots, de capacités unitaires ou non unitaires. Les problèmes sont représentés par un graphe disjonctif. Une méthode hybride, composée d'un algorithme mimétique intégré avec une recherche locale, est utilisée pour résoudre ces deux types de problèmes. [Elmi et al., 2011] ont traité un problème d'ordonnancement du j*ob shop* dans des cellules (JSCP) en y intégrant les transports entre cellules et le problème de ré-entrée. Les *jobs* sont classés en familles. Les *jobs* d'une même famille vont visiter sensiblement le même ensemble de machines. Chaque machine peut traiter différents jobs mais un seul à la fois. Chaque cellule consiste en un ensemble de machines. Certains *jobs* doivent visiter les autres cellules pour finir leur traitement. Les

transports entre les cellules sont pris en compte. Pour traiter chaque famille de jobs, le temps de préparation d'une cellule est indépendant du séquencement. Le problème dans une cellule est du type job shop avec des machines multifonctions. Une méthode de SA (simulated annealing) est développée avec une structure de voisinage basé sur les blocs sur le chemin critique. Ce type de problème est plus complexe que le problème job shop classique. Mais les ressources de transport sont supposées toujours suffisantes et les temps de transport entre machines dans une même cellule sont négligés.

- avec affectation des machines

Les problèmes où chaque tâche opératoire (ou certaines tâches opératoires) peut être réalisée par plusieurs machines, sont qualifiés de problèmes avec machines flexibles ou routage flexible. La plupart des publications traitent de problèmes de *job shop* flexible mais ne prennent pas en compte les contraintes de transport. Peu de recherches considèrent à la fois la flexibilité des machines et l'ordonnancement des transports. Ce type de problème est plus complexe que le problème précédent ainsi que le *job shop* flexible (FJSP) sans transport.

[Andrea and Gino, 2007] ont traité le problème FJSP dans les FMSs incluant les temps de préparation des machines et de transport. Le problème est représenté par un graphe disjonctif avec des nœuds associés aux tâches opératoires. Une heuristique de colonie de fourmis (*ant colony*) est utilisée pour optimiser le makespan. Le graphe est évalué par une règle de mise à jour locale (*local update rule*). La recherche locale est inspirée de l'algorithme de Nowicki et Smutnicki ([Nowicki and Smutnicki, 1996]). L'ordonnancement des tâches de transport n'est pas pris en compte parce que le nombre de ressources de transport est supposé illimité. [Deroussi and Norre, 2010] ont étendu les instances de FJSP en y ajoutant les transports. Chaque tâche opératoire est associée à un ensemble de machines (deux ressources) qui peuvent la réaliser. Il n'y pas de machine commune pour les tâches de traitements. Une méthode dynamique est proposée pour ordonnancer à la fois les machines et les AGVs. [Nonaka et al., 2012] ont considéré un problème FJSP avec séquence de traitement flexible. Chaque *job* a un ensemble de séquences de traitements différentes. Il n'y pas de tâche opératoire en commun parmi ces séquences et la longueur de chaque séquence peut varier. Chaque tâche opératoire est associée à un ensemble de machines qui sont capables de la traiter. Les temps de traitement des tâches sont dépendants de la machine. Une solution consiste donc à déterminer une séquence de traitement, une affectation des machines pour traiter les tâches et les dates de début pour chaque tâche. Le problème de séquencement du transport est négligé. L'approche proposée résout le problème en deux étapes. La première étape détermine des séquences sélectionnées pour chaque *job* et associe des machines aux séquences. Ce problème revient à équilibrer les charges des machines en minimisant la charge maximale des machines. Une solution optimale sera obtenue à l'aide d'un programme linéaire. Dans la deuxième étape, une solution initiale est générée par l'insertion des *jobs* un par un. Une recherche tabou

est ensuite appliquée pour améliorer la solution initiale. Ce type de problème existe dans les systèmes les plus flexibles. Il est plus difficile à résoudre que le *job shop* classique à cause du séquencement des traitements flexibles et des machines flexibles. Cependant, la contrainte de transport est ignorée, ce qui n'est pas souvent le cas dans les FMS. Il serait intéressant de généraliser notre modèle générique en y ajoutant les séquences flexibles.

2.2.2 Problèmes sans stock

Dans la littérature, les problèmes sans stock sont le plus souvent associés à d'autres contraintes, en particulier des temps bornés, voire des cas sans attente autorisée. Ce type de problèmes très contraint se rencontre dans l'industrie agro-alimentaire ou l'industrie du traitement de surface. Les problèmes associés sont connus sous la dénomination *Hoist Scheduling Problems* (HSPs). Ci-dessous nous fournissons les publications concernant les problèmes de HSP prédictif (PHSP). Il est à noter également que les HSPs les plus étudiés dans la littérature sont de type prédictif, mais cyclique (souvent du *flow shop*) ou de type dynamique. De même, l'affectation des machines est souvent connue a priori dans le cas statique, avec en majorité des problèmes avec ressource de transport unique.

[Song et al., 1993] ont proposé une méthode itérative nommée "procédure de date au plus tôt" dans le cas mono-produit. Quand la ligne de traitement n'est pas vide, la décision de rentrer un nouveau produit est importante. Cette procédure essaye de rentrer les produits un par un. Quand la contrainte sans attente est violée, le séquencement du dernier *job* rentré doit être réordonnancé en retardant la date d'entrée. La procédure garantit de générer une solution faisable. Cette méthode peut être étendue pour le cas multi-produits. Cependant, les temps de traitement bornés ne sont pas pris en compte. La décision de la date d'entrée du *job* est faite avec les temps de traitement minimaux. [Varnier and Baptiste, 1995] ont optimisé le temps transitoire pendant le changement de production. La méthode proposée consiste à combiner les mouvements des deux cycles optimaux pendant la phase transitoire. Un modèle linéaire et la programmation logique sous contraintes sont utilisés. Une procédure de branch-and-bound est dédiée à l'arbitrage des contraintes disjonctives. [Hertz et al., 1996] traite le HSP prédictif avec un seul robot (hoist). Tous les *jobs* sont identiques et l'objectif est de chercher des séquences identiques pour chaque produit en minimisant le *makespan*. Une hypothèse est ajoutée : les cuves sont multi-bacs, leur capacité est supérieure ou égale au nombre de *jobs*. Un algorithme itératif est utilisé. L'espace des solutions contient non seulement les solutions faisables pour le HSP, mais aussi les solutions qui ont violé les contraintes. En effet, il y a très peu de solutions faisables pour HSP par rapport aux solutions faisables pour les problèmes moins contraints. Le séquencement est construit petit à petit en vérifiant les règles (contraintes) définies. Ces règles aident à calculer les dates provisoires. À chaque étape, une tâche (de traitement ou de transport) qui respecte les contraintes est séquencée. La procédure continue jusqu'à ce qu'il soit impossible de choisir une tâche sans violer de contrainte. Cela ne veut pas dire tout de suite que la solution partielle n'est pas faisable.

Un graphe de contraintes et l'algorithme de Bellman-Ford sont utilisés pour vérifier la faisabilité du séquencement partiel ou total, ainsi que pour calculer le *makespan*. Quand une séquence partielle n'est pas faisable, certains arcs (liés à un *job*) sont retirés pour construire un graphe réduit. Cette procédure continue jusqu'à ce qu'on ne trouve plus de circuit de longueur positive dans le graphe réduit. [Lacomme, 1998] a utilisé un système multi-agents pour traiter le problème avec un seul *hoist* et différents types de *jobs*. Chaque cuve, chaque robot et la ligne sont considérés comme les agents. Un système nommé *triple coupling* est proposé.

Les références suivantes concernent des problèmes multi-produits, donc plutôt de type *job shop*. [Caux et al., 1992], [Caux et al., 1995] et [Fleury, 1995] ont utilisé un simulateur de système à événements discrets pour évaluer une solution. [Rossé-Bloch, 1999] a utilisé une procédure de *Shifting bottleneck* modifiée pour résoudre un HSP prédictif (PHSP). Un graphe disjonctif est proposé pour prendre en compte les temps de traitement bornés. Une hypothèse est ajoutée : il existe suffisamment de robots pour transporter les produits. Les temps de transport en charge sont ajoutés dans les valeurs sur les arcs. Ces temps ne sont pas les vraies valeurs des temps de transport en charge, mais reviennent à allonger le temps d'exécution des tâches opératoires. [Paul et al., 2007] ont proposé une heuristique nommée *adapted time windows* pour le HSP multi-produits avec un seul robot. Les chargements des *jobs* sont pris un par un selon leur date d'arrivée. Une fois qu'un *job* est totalement introduit, toutes les exécutions sont calculées par les fenêtres de temps. L'introduction du *job* suivant peut réduire les fenêtres de temps requis pour les *jobs* en cours sur la ligne. C'est-à-dire que toutes les activités en cours doivent être réordonnancées, mais sans réaffecter les ressources des tâches. Cette procédure insère les tâches une par une en vérifiant la faisabilité : si la tâche actuelle viole la contrainte sans attente, une procédure de *backtracking* est utilisée pour re-séquencer les anciennes tâches dans le même *job*. Elle essaie d'abord de re-séquencer cette tâche en retardant sa date au plus tôt (insérer les autres tâches avant et ensuite re-déterminer sa date au plus tôt). Si la modification est faisable, on continue ; sinon, on revient sur la tâche précédente. Le pire cas est de revenir jusqu'au début du *job* qui peut être toujours séquencé sans violer les contraintes. Cette procédure assure au moins une solution faisable mais pas forcément intéressante. La méthode proposée ici ordonnance un par un les produits. Elle peut être étendue pour les cas dynamiques. Cette méthode n'est pas valable quand les produits ne commencent pas nécessairement par le poste de chargement.

Une typologie de ces problèmes d'ordonnancement avec transport est présentée dans la figure 2.3. Dans cette figure, le paramètre recrc signifie recirculation. Il est relatif aux problèmes dits réentrants où un produit peut visiter plusieurs fois une même machine.

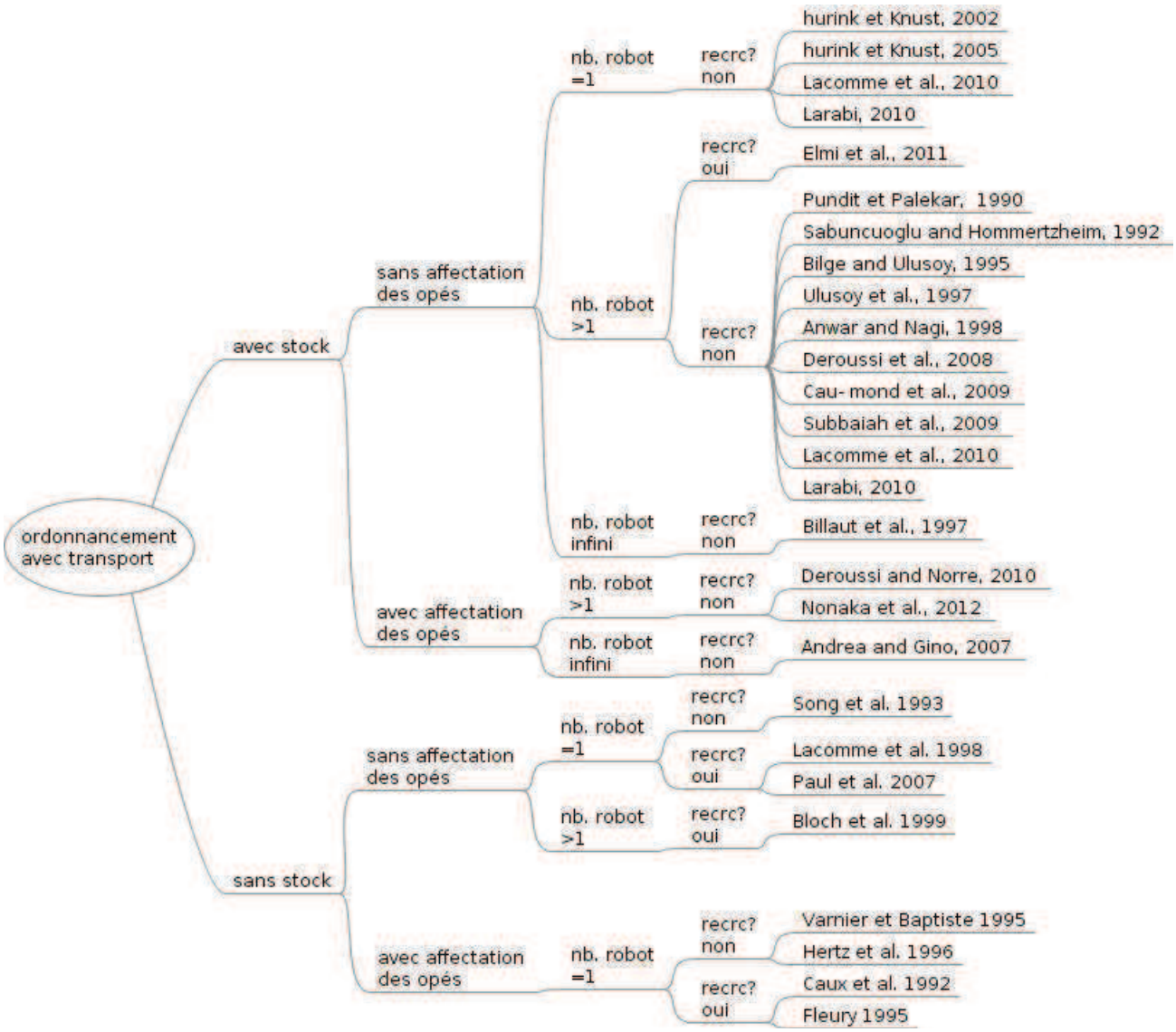

FIGURE 2.3 – Typologie de problèmes d'ordonnancement avec transport

2.2.3 Autres problèmes d'ordonnancement

Dans les systèmes automatisés, l'amélioration des performances conduit, si c'est possible, à la recherche d'ordonnancements cycliques dans lesquels une séquence des mouvements des robots est répétée indéfiniment. La productivité dépend à la fois du séquencement des mouvements des robots et de la séquence d'entrée des différents produits sur l'atelier [Agnetis, 2000]. L'ordonnancement cyclique peut être de degré n (ou n-périodique), si n *jobs* entrent dans le système de production à chaque période. Ce type de problème existe souvent dans les ateliers de traitement de surface et les cellules robotisées. [Brauner et al., 2005] ont prouvé que chercher une séquence cyclique des mouvements du robot avec temps de transport en charge, est NP-complet. La plupart des publications du domaine concernent les ateliers sans stockage. Quelques-unes concernent ceux avec stockage [Finke et al., 1996, Dawande et al., 2005]. Ici, nous ne considérons que les problèmes cycliques sans stockage. Il y a principalement trois types de problèmes rencontrés :

- **temps de traitement non bornés** : chaque type de produit a des temps de traitement définis et fixés, et une fois que son traitement est fini, le produit peut rester indéfiniment sur la machine en attendant la disponibilité du robot [Crama and Van de Klundert, 1997a, Hall et al., 1998, Sethi et al., 1992].
- **temps de traitement fixe, zero marge ou no-wait** : les temps de traitement sont fixés, mais une fois qu'un traitement est fini, le produit doit être déplacé immédiatement. C'est souvent le cas des cellules sans stockage ([Levner et al., 1998]).
- **temps de traitement bornés** : comme le cas précédent, le produit peut être déplacé dès que la borne minimale du temps de traitement est atteinte. Mais il doit avoir quitté la machine avant que la borne maximale du temps opératoire ne soit dépassée. C'est le cas pour le HSP.

Ici, nous présentons ces problèmes en deux catégories : les problèmes où les temps de traitement sont non bornés et ceux sans attente avec temps fixes ; les problèmes avec temps de traitement bornés.

*** Problèmes avec temps de traitement non bornés ou temps fixes sans attente**

[Sethi et al., 1992] ont traité le problème du flow shop dans les cellules robotisées sans stock et avec un nombre de machines limité (inférieur à trois). Ils ont prouvé que, dans une cellule robotisée avec deux machines traitant un seul type de produit, la solution optimale est un cycle de degré un.

[Logendran and Sriskandarajah, 1996] ont étendu ces résultats au cas multi-produits et avec des temps de transport variables. [Crama and Van de Klundert, 1997a], [Crama and Van de Klundert, 1997b] et [Crama et al., 2000] ont étudié un cas mono-produit et

mono-robot. Ils ont traité une extension du problème étudié dans [Sethi et al., 1992] au cas à m machines ($m > 3$). Une programmation dynamique est proposée pour résoudre le problème en temps $O(m^3)$. Ils ont prouvé que ce type de problème est fortement NP-complet. [Levner et al., 1997] ont développé un algorithme (complexité en $O(m^3 log(m))$) pour trouver le cycle optimal des mouvements de robot pour un problème sans attente et mono produit. [Hall et al., 1998] ont prouvé que l'ordonnancement ne se limite pas à l'obtention d'un cycle de degré un dans une cellule avec trois machines. Le problème de séquencement cyclique multi-produits à trois machines est NP-complet. [Agnetis, 2000] ont considéré le problème de *flow shop* dans une cellule à un seul robot et sans attente. Dans le cas de deux machines, ils ont montré que le problème peut être résolu en $O(nlog(n))$, en le réduisant au problème classique d'un *flow shop* sans attente à deux machines. Dans le cas mono-produit à trois machines, il est suffisant de considérer les cycles du robot dans lesquels toutes les machines sont visitées une ou deux fois. [Mangione et al., 2003c] ont prouvé qu'une solution cyclique de degré 1 ou 2 est optimale pour le problème cyclique mono-produit à trois machines et un seul robot, lorsque les temps de traitement sont fixes. [Mangione et al., 2003a], [Mangione et al., 2003b] ont traité le problème cyclique mono produit dans un atelier sans attente équilibré à quatre machines et un seul robot. Cet article a montré que pour trouver les cycles optimaux, on peut se limiter à trouver des cycles de production de moins de trois pièces, ce qui veut aussi dire que le degré des cycles optimaux est borné. Le degré des cycles dominants peut être borné par le nombre de machines moins un, ce qui simplifie la recherche des solutions optimales. [Che et al., 2003] ont cherché des ordonnancements cycliques de degré 2 pour des cellules robotisées mono-produit sans attente, où les temps de traitement sont fixes. Un algorithme polynomial de complexité $O(N^8 logN)$ a été proposé. Cette méthode peut-être étendue pour résoudre le problème avec deux types de jobs. [Dawande et al., 2005] ont étudié les problèmes rencontrés dans différents ateliers : système de traitement de surface, planification de grues (*crane scheduling*), usines textiles et ateliers de production de blocs moteur (*engine block manufacturing*). [Brauner, 2006] a proposé un état de l'art sur un problème d'ordonnancement cyclique de cellule robotisée de type *flow shop* sans stockage et avec des produits identiques. Le cas de trois machines avec temps de traitement non bornés est résolu complètement, mais pas le cas à quatre machines. [Leung and Levner, 2006] ont proposé un algorithme de complexité $O(n^2)$ pour déterminer un séquencement optimal du mouvement des robots pour un temps de cycle donné. Ils ont aussi proposé un algorithme pour déterminer le nombre minimal de robots requis pour un temps de cycle donné. Sa complexité est $O(n^5)$. Ces deux méthodes sont dédiées aux cas avec temps fixes. Si les temps de traitement sont bornés, le problème devient plus compliqué et leurs méthodes ne sont plus applicables. [Fiedler and Meyer, 2008] ont considéré le problème de blocage (*deadlock*) et de collision dans un *flow shop* robotisé de avec 25 machines (des cuves) et deux robots. [Kamalabadi et al., 2008] ont développé un nouveau modèle mathématique pour un problème cyclique multi-produits, avec trois machines et un seul robot. Ce mod-

èle est basé sur les réseaux de Petri. [Levner et al., 2007] ont fourni un bref aperçu de l'évolution des problèmes basiques d'ordonnancement cyclique et les approches possibles pour leur résolution. [Brucker and Kampmeyer, 2008] ont proposé un cadre général pour la modélisation et la résolution des problèmes d'ordonnancement cycliques.

Certaines publications traitent les problèmes avec des robots spéciaux dotés de deux pinces de préhension (*dual gripper*), qui permettent de décharger et charger simultanément une même machine. [Sethi et al., 2001] ont considéré le problème cyclique mono produit. La productivité est au plus doublée par rapport au robot classique. [Geismar et al., 2003] et [Geismar et al., 2006] ont traité ce problème avec des machines parallèles. [Sriskandarajah et al., 2004] ont étudié le cas multi-produits. [Drobouchevitch et al., 2006] ont traité le problème mono-produit en adaptant les méthodologies utilisées dans le cas d'un robot mono-préhenseur.

*** Problèmes avec temps de traitement bornés et sans stock**

Ce type de problème correspond aux problèmes cycliques du HSP (CHSP), rencontrés dans les ATS sans stock ([Bloch et al., 2008]). Le CHSP le plus simple, nommé en tant que "problème de base" ([Manier-Lacoste, 1994]), consiste à chercher la période minimale d'un cycle dans une seule ligne avec un robot et des ressources de traitement (cuves) de capacité unitaire et mono-fonction (problème non réentrant). Il est NP-complet même dans le cas le plus simple ([Lei and Wang, 1994]). Cela explique pourquoi la plupart des chercheurs ne résolvent que ce cas.

Pour le problème mono-produit avec un unique robot, [Phillips and Unger, 1976] ont proposé le premier modèle de type programmation linéaire entière mixte (Mixed integer Linear Programming). [Shapiro and Nuttle, 1988] ont utilisé une procédure arborescente de type séparation et évaluation pour résoudre ce problème avec des postes de chargement et déchargement dissociés. [Baptiste et al., 1993] et ([Manier-Lacoste, 1994]) ont appliqué la programmation logique sous contraintes (*Constraint Logic Programming*). [Mateo et al., 2000] ont proposé une formule mathématique pour définir le CHSP. [Mateo Doll, 2001] propose de chercher le chemin maximal dans un graphe cohérent (avec des chemins bornés). [Chen et al., 2002] utilisent deux procédures de type *branch and bound* pour trouver une séquence cyclique de degré 1. Ils utilisent aussi ce type de méthode exacte pour chercher des solutions n-cycliques ([Che et al., 2002]). [Liu et al., 2002] ont étendu au cas des cuves multifonctions et multi-bacs, le modèle de programmation entière mixte de [Phillips and Unger, 1976] pour le CHSP basique avec un seul robot. [Riera and Yorke-Smith, 2002] ont modélisé le problème de 1-cycle HSP avec un seul ou plusieurs robots, en prenant en compte la capacité des cuves. Au lieu de chercher directement la solution optimale des lignes de traitement de surface, [SUBAÏ et al., 2003] ont étudié l'adaptation d'une séquence cyclique mono-produit. Ils ont montré qu'une même séquence offre toute une plage de production possible en faisant varier le temps de cycle. Pour un même temps

de cycle (y compris l'optimum), il existe un ensemble de solutions donnant chacune des durées opératoires différentes.

Certains publications prennent en compte deux critères, non seulement le temps de cycle mais aussi un critère environnemental ([Xu and Huang, 2004], [Kuntay et al., 2006], [Subai et al., 2006]). Ce type de problème est nommé ECCHS (*environmently conscious cyclic hoist scheduling*) ou CHE (*Cyclic Hoist Scheduling for Environment*).

Dans le HSP, la présence de plusieurs robots engendre une complexité supplémentaire, due à la gestion des risques de collision. La méthode la plus souvent retenue dans les divers papiers consiste à partitionner le problème en sous-problèmes mono-robot, en découpant la ligne en plusieurs zones géographiques qui peuvent se chevaucher ou non (voir par exemple [Lei and Wang, 1991], [Manier et al., 1994], [Manier et al., 2000], [Liu and Jiang, 2005], [cht, 2010]). Enfin, certaines études traitent de l'optimisation du nombre de robots conjointement avec l'ordonnancement ([Armstrong et al., 1996], [Manier and Lamrous, 2006] et [Manier and Lamrous, 2008]).

Pour les problèmes d'ordonnancement cycliques multi-produits, il y a souvent peu de types de produits à cause de la complexité du problème. De plus ces types de produits sont souvent similaires, avec des gammes opératoires identiques dans lesquelles seules les durées peuvent varier. Cela revient à nouveau à un problème de *flow shop*. Pour ces problèmes, il est souvent plus intéressant de chercher des ordonnancements n-cycliques (voir par exemple [Collart Dutilleul and Denat, 1998], [**?**],[Mateo et al., 2002] et [Mateo and Companys, 2007],[Bonhomme et al., 2002], [El Amraoui et al., 2012]).

Les problèmes cycliques n'étant pas l'objet de notre travail, nous fournissons une synthèse des travaux sous forme des figures 2.4 et 2.5. Celles-ci fournisssent une typologie et des références que le lecteur intéressé pourra consulter. La figure 2.4 donne une typologie des problèmes cycliques rencontrés dans les cellules robotisés. La figure 2.5 est plus dédiée aux problèmes cycliques rencontrés dans les ATS.

2.2.4 Conception du système et routage des véhicules

La conception du système consiste notamment à implanter le système physique (avec dimensionnement des ressources), et à définir la structure du réseau de transport ainsi que son pilotage, en prenant en compte les risques de collision.

Il existe plusieurs méthodes pour concevoir le système d'AGVs et le routage. [Malmborg, 1990] ont proposé une combinaison de la simulation et d'un modèle analytique. [Bozer and Srinivasan, 1992] introduisent un algorithme basé sur une approche d'ensemble et de partitionnement (*set-partitioning*) pour décomposer un système sans chevauche-

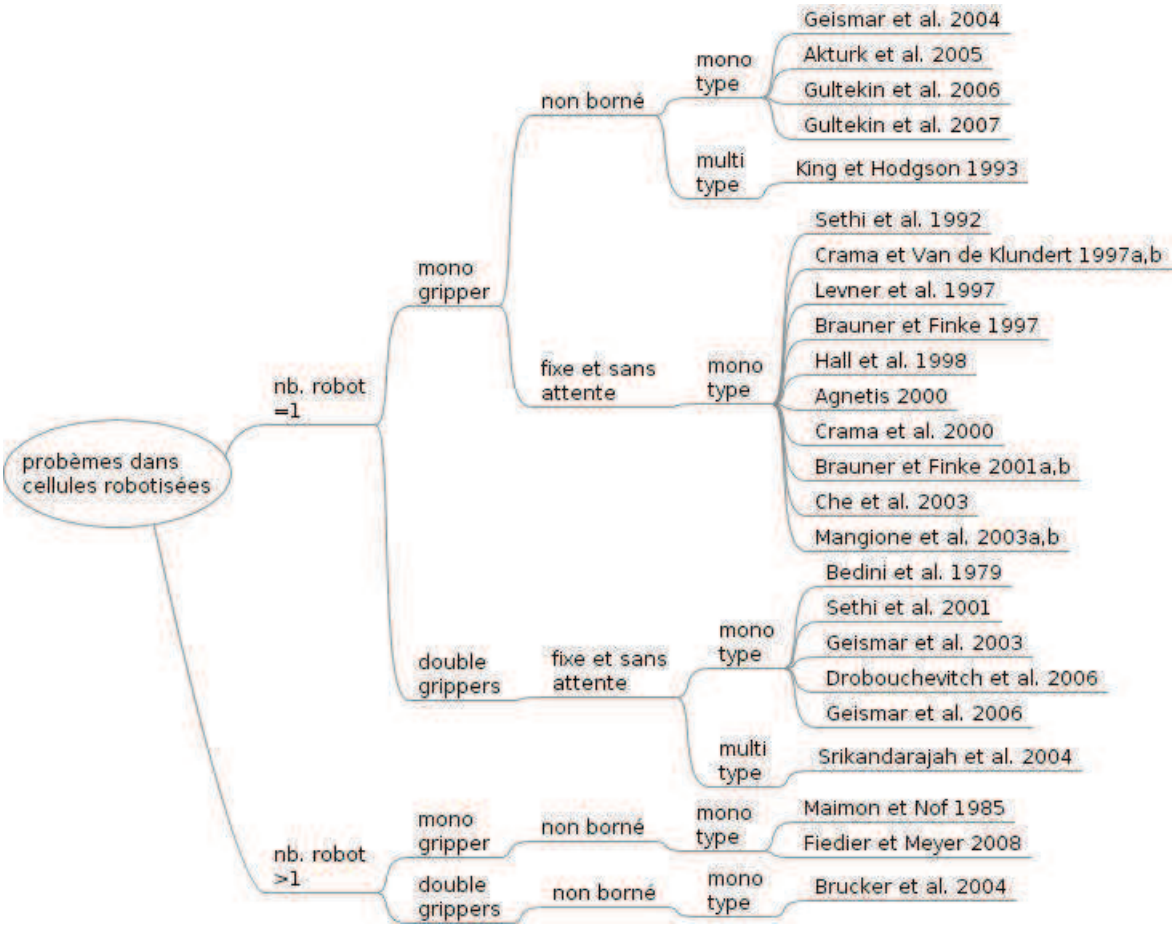

FIGURE 2.4 – Typologie des problèmes cycliques dans les cellules robotisées (*Robotic Scheduling* ou RS)

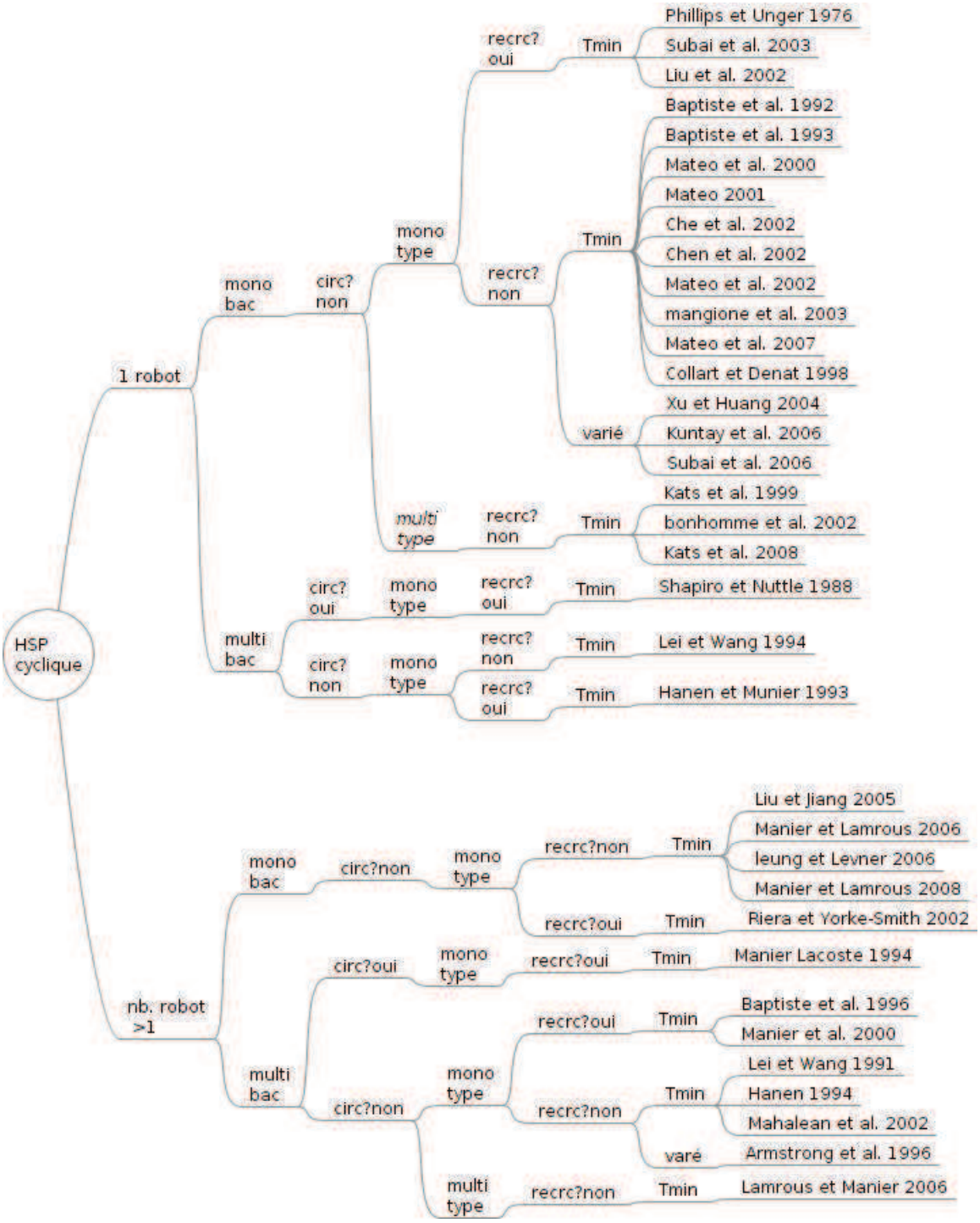

FIGURE 2.5 – Typologie des problèmes cycliques rencontrés dans les ATS (CHSP)

ment. [Johnson and Brandeau, 1993] ont proposé un modèle analytique pour un FMS avec plusieurs véhicules. [Bilge and Tanchoco, 1997] et [Van der Meer, 2000] ont étudié le cas avec multi-capacité des AGVs. L'usage de ce type d'AGVs peut réduire le nombre de véhicules nécessaires et augmenter le débit du système, notamment en cas de demande élevée de transports. [Savelsbergh and Sol, 1998] proposent un modèle de type PDPTW (*pickup and delivery problem with time windows*) en temps réel, en utilisant un algorithme de branch and price pour résoudre dynamiquement le routage et l'ordonnancement des AGVs. [Yang et al., 2004] ont proposé plusieurs politiques pour programmer les véhicules dynamiquement. L'objectif est de minimiser le coût total de traitement de toutes les demandes, qui est une combinaison du coût de la distance de voyage vide, de la pénalité de retard, et de perte de profit. [Corréa et al., 2007] ont proposé une méthode hybride pour résoudre un problème de répartition et de routage sans conflit de véhicules guidés automatisés (AGV) dans un système de fabrication flexible. Ce problème consiste à résoudre simultanément l'affectation, le séquencement et le routage sans conflit des véhicules. L'approche proposée consiste en une méthode de décomposition où le problème principal (ordonnancement) est modélisé avec la programmation par contraintes et le sous-problème (routage sans conflit) avec la programmation en nombres entiers mixte. La méthode hybride présentée a permis de résoudre des cas avec un maximum de six AGVs. [Pisinger and Ropke, 2007] ont proposé une heuristique générale pour résoudre cinq types de problèmes de routage des véhicules : VRPTW (*vehicule routing problem with time windows*), CVRP (*capacitated vehicule routing problem*), MDVRP (*multi-depot vehicule routing problem*), SDVRP (*site dependent vehicule routing problem*) et OVRP (*open vehicule routing problem*). Ces problèmes sont transformés en un modèle de *pick-up and delivery* et ils sont résolus par le *framework Adaptive Large Neighborhood Search* (ALNS). C'est une recherche de voisinage qui a un niveau adaptatif pour choisir une heuristique parmi un ensemble d'heuristiques d'insertion et de retrait.

2.2.5 Synthèse

L'examen de la littérature scientifique consacrée aux problèmes d'ordonnancement avec transport montre que la plupart des articles publiés traitent principalement deux types de problèmes : ceux avec temps de traitement fixes, et ceux avec temps de traitement bornés. Pour le premier cas, les problèmes sont souvent du type job shop avec ou sans stock. Le deuxième cas se rencontre dans les ATS ou RC sans stock. Mais la plupart des problèmes traités sont relatifs à la recherche d'ordonnancements de type cyclique dans le cas mono-produit ou multi-produits. En outre, la majorité des auteurs considèrent des ateliers "simples", ne comportant qu'un unique robot, avec des machines de capacité unitaire et mono-fonction.

On trouve une littérature conséquente consacrée aux problèmes propres à chaque type de système et de nombreux résultats y sont présentés. Mais ces problèmes sont souvent des cas particuliers. D'autre part, peu de liens entre les différents ateliers sont établis, de

même qu'avec les problèmes associés, et encore moins entre les méthodes et outils utilisés pour les résoudre. Ces liens ne font d'ailleurs pas toujours l'unanimité. Par exemple, [Crama et al., 2000] incluent le HSP dans la classe des robotic flowshop scheduling problems. A l'inverse, pour [Cavory et al., 2005], c'est le robotic flowshop scheduling problem qui est vu comme un cas particulier du HSP, lui-même considéré comme une classe de problèmes d'ordonnancement d'atelier cycliques avec contraintes de ressources. Toutefois, suivant le cas étudié, un HSP peut aussi ne correspondre à aucun des problèmes classiques d'ordonnancement d'atelier. En effet, dans la majorité des problèmes traités, il est rare de trouver réunies l'ensemble des caractéristiques du HSP : outre le risque de collision entre les ressources de transport (robots) et le fait que le problème est sans attente et sans préemption, les temps de manutention sont non négligeables devant les durées opératoires, celles-ci étant bornées non seulement par valeur inférieure mais aussi par valeur supérieure.

On constate finalement un manque de travaux liés à des problèmes généraux qui font l'objet de ce mémoire, et qui intègrent simultanément les contraintes relatives aux opérations de production et de transport, aux temps de traitement bornés avec ou sans stock, avec des ressources de traitement polyvalentes ou non, et des moyens de transport présents en exemplaire unique ou multiple.

Dans la section suivante, nous définissons le problème d'ordonnancement d'atelier de type *job shop* que nous étudions par la suite. Nous qualifions ce problème de générique dans le sens où il regroupe des problèmes issus de différents types d'ateliers avec contraintes de transport. Il servira de base pour l'élaboration de modèles et méthodes de résolution que nous décrivons dans les chapitres suivants.

2.3 Problème d'ordonnancement GFJSSP

Nous définissons ici un problème d'ordonnancement de type *job shop* qui peut être vu comme une concaténation des problèmes de *job shop* flexible avec ressources de transport rencontrés dans les divers ateliers de type systèmes flexibles de production (FMS), cellules robotisées (RC) et ateliers de traitement de surface (ATS). Nous appelons aussi ce problème *General Flexible Job Shop Scheduling Problem* (GFJSSP) ou problème générique, le terme "general" intégrant les aspects contraintes de transport et de stockage, ainsi que des durées opératoires bornées.

L'atelier générique considéré est constitué de trois principaux éléments physiques, comme nous l'avons décrit au chapitre précédent : les ressources de traitement (machines, cuves, etc.), les ressources de transport (AGVs, convoyeurs, *hoists*, robots, etc.) et les places de stockage (en amont et/ou en aval des ressources de traitement).

Les éléments du *job-shop* considéré ici sont : un ensemble de ressources de traitement (ou machines), un ensemble de ressources de transport (ou robots) et un ensemble de *jobs* indépendants J_1, J_2, ... J_n. Chaque *job*/travail a une date de début au plus tôt ou *release*

date, et une date de fin au plus tard ou *duedate*. Dans notre étude, nous supposerons que les *release dates* sont toutes égales à 0, ce qui veut dire que tous les travaux sont disponibles à la date de référence 0. Ceci nous place dans un contexte prédictif, où on cherche à déterminer une solution hors ligne. De même, nous n'imposons pas de date de fin des travaux, qui est donc fixée par défaut à une valeur très grande par rapport à l'horizon temporel considéré. Chaque travail doit également être réalisé suivant une gamme opératoire fixée (les tailles de gamme peuvent être différentes). Les opérations d'une gamme sont liées par des contraintes de précédence. Chaque opération doit être effectuée sur une seule des machines, choisie dans l'ensemble des machines qui peuvent traiter cette opération. Les opérations sont non préemptives, c'est à dire ne peuvent être interrompues une fois commencées. Le temps opératoire pour chaque opération est borné par valeurs inférieure et supérieure. La borne supérieure peut prendre différentes valeurs : soit elle est égale à la borne inférieure, ce qui revient à un temps fixe : soit elle est strictement supérieure à cette borne inférieure, auquel cas on est dans le cas d'une fenêtre de temps à respecter strictement ; soit enfin elle peut prendre une valeur infinie, ce qui signifie que le produit peut attendre sur la machine pendant un temps indéterminé. Une machine ne peut traiter qu'un seul job à la fois. A chaque ressource de traitement est associé un stock amont et un stock aval, dont la capacité varie entre 0 et $+\infty$ suivant les systèmes. Une capacité nulle est représentative d'un problème sans stock. Une capacité infinie signifie que le stock dispose toujours d'un nombre suffisant de places pour absorber la charge. Lorsqu'un traitement est terminé, le *job* associé doit être transporté à la machine suivante dans sa gamme opératoire. Cette tâche est réalisée par une ressource de transport. Les ressources de transport sont identiques et ne peuvent déplacer qu'un *job* à la fois (capacité unitaire). Tous les temps de transport en charge et à vide sont indépendants des *jobs* et de la ressource de transport utilisée, mais dépendants des machines (c'est à dire des points d'origine et de destination des mouvements).

Le problème étudié consiste à ordonnancer l'ensemble des tâches de traitement et de transport. L'objectif principal du problème est de minimiser le temps total de réalisation de l'ensemble des travaux, soit le *makespan*, tout en respectant l'ensemble des contraintes :

- de gamme (contraintes de précédence classiques pour des gammes linéaires),
- de durées opératoires bornées,
- de durées de transport (non négligeables devant les temps opératoires),
- de capacité des ressources de traitement, de transport et de stockage,
- d'affectation des ressources,
- de non préemption.

Par ailleurs, nous considérons pour notre étude les hypothèses suivantes :

- les pannes des ressources sont ignorées ;
- toutes les machines sont disponibles à la date de référence zéro ;
- tous les travaux sont disponibles à la date zéro ;
- les risques de collision entre les ressources de transport ne sont pas gérés ;

- par défaut, toutes les ressources de transport sont capables de réaliser n'importe quel déplacement de pièces ;
- il n'y a pas de transport entre une ressource de traitement et les stocks amont et aval qui lui sont associés.

Notons que l'objectif principal commun à tous les problèmes identifiés est de minimiser le *makespan*. Toutefois, nous pouvons aussi considérer un second objectif qui s'avère critique dans le cas de certains systèmes : il s'agit de minimiser le stockage (défini par le temps d'attente total de toutes les tâches opératoires avant ou après leur passage sur les ressources de traitement). Par exemple, dans les ATS, ce critère est associé à des exigences de qualité des produits traités. Dans ce cas, le problème devient alors multi-objectifs.

2.4 Bilan

En nous basant sur l'identification et l'analyse des similitudes et divergences des ateliers de production avec ressources de transport, nous avons défini un atelier et un problème d'ordonnancement génériques qui intègrent l'ensemble des caractéristiques que l'on peut trouver dans chacun des systèmes identifiés. Nous avons conscience de la complexité d'un tel type de problème. Notre objectif est d'élaborer une ou des méthodes adaptées à sa résolution et présentant de bonnes performances quel que soit le type d'instances traité (FMS, RC ou ATS). Notons que ce travail fait écho à l'une des problématiques à l'origine du groupe de recherche en ordonnancement "Bermudes : HSP/FMS/FSHP : similitudes, divergences, typologies, notations" (GdR MACS et GdR RO). Le développement de ces méthodes fait l'objet des chapitres 4 et 5. Au préalable, nous présentons au chapitre suivant deux modélisations du problème générique GFJSSP.

Chapitre 3

Modèles pour le General Flexible Job Shop Scheduling Problem

3.1 Introduction

Dans cette partie, deux modèles génériques sont proposés pour représenter le *General Flexible Job Shop Scheduling Problem* (GFJSSP), englobant tous les systèmes étudiés dans la partie précédente. Le premier est une formulation mathématique du problème, le second en est une représentation graphique. Ce sont des extensions de modèles de *job shop* flexible dans lesquels nous avons intégré des contraintes de transport et des temps opératoires bornés, sachant que le transport induit souvent des contraintes critiques dans les systèmes que nous étudions. Par ailleurs, le *job shop* flexible est une généralisation du *job shop*, qui permet de traiter une opération sur n'importe quelle machine d'un ensemble donné. En raison de ces caractéristiques, il est nécessaire de résoudre un problème d'affectation en plus du problème de séquencement.

3.2 Notation

Nous détaillons ici la notation utilisée dans les modèles que nous proposons dans la suite de ce chapitre.

Données

Jobs

J :	ensemble de travaux/*jobs* à traiter
n :	nombre de travaux dans J
J_i :	travail i ($i \in [1, n]$)
O_i :	nombre d'opérations de J_i ($i \in [1, n]$)
O :	nombre total d'opérations, $O = \sum_{i=1}^{n} O_i$
OP_{ij} :	$j$ème opération de J_i ($j \in [1, O_i], i \in [1, n]$)

P_{ij}^- : durée de traitement minimale pour OP_{ij}
P_{ij}^+ : durée de traitement maximale pour OP_{ij}
d_i : date de fin au plus tard de J_i (*due date*).
r_i : date de disponibilité du J_i (date de début au plus tôt, *release date*).
$U_{jj'}^i$: $= 1$, si OP_{ij} doit être réalisée avant $OP_{ij'}$, donc si $j < j'$
$= 0$, sinon.
ces paramètres expriment les contraintes de précédence entre deux opérations d'un même travail J_i

Traitement

M : ensemble des ressources de traitement (machines).
Il inclut les postes de chargement et déchargement, s'ils existent
m : nombre total de ressources de traitement
M_k : ressource de traitement k ($k \in [1, m]$)
MP_{ij} : ensemble des ressources de traitement qui peuvent réaliser OP_{ij} ($i \in [1, n], j \in [1, O_i]$)

Transport

R : ensemble des ressources de transport
r : nombre total de ressources de transport
R_h : ressource de transport h ($h \in [1, r]$).
TP_{ij} : tâche de transport entre OP_{ij} et OP_{ij+1} ($i \in [1, n], j \in [1, O_i - 1]$).
TR_{ij} : ensemble des ressources de transport qui peuvent transporter TP_{ij}
σ_{kl} : durée de déplacement à vide entre les machines M_k et M_l, $k, l \in [1, m]$.
τ_{kl} : durée de transport en charge entre les ressources M_k et M_l, $k, l \in [1, m]$.
T_{ij}^- : durée de transport minimale pour réaliser TP_{ij} ($i \in [1, n], j \in [1, O_i - 1]$)
$= \tau_{kl} / PJ_{ijk} = 1, PJ_{ij+1l} = 1$
T_{ij}^+ : durée de transport maximale pour réaliser TP_{ij} ($i \in [1, n], j \in [1, O_i - 1]$)

Stockage

S : ensemble des places de stockage
UST_{ij} : tâche de stockage en amont de la machine effectuant la tâche OP_{ij}, $i \in [1..n], j \in [1, O_i]$
DST_{ij} : tâche de stockage en aval de la machine effectuant la tâche OP_{ij}, $i \in [1..n], j \in [1, O_i]$
US_{ij}^- : durée minimale de UST_{ij}
US_{ij}^+ : durée maximale de UST_{ij}
DS_{ij}^- : durée minimale de DST_{ij}
DS_{ij}^+ : durée maximale de DST_{ij}

Variables

c_i : date de réalisation de J_i

p_{ij}^- : date au plus tôt pour commencer OP_{ij} ($i \in [1,n]$, $j \in [1,O_i]$)

p_{ij}^+ : date au plus tard pour commencer OP_{ij}($i \in [1,n]$, $j \in [1,O_i]$)

p_{ij} : date réelle de début de OP_{ij}($i \in [1,n]$, $j \in [1,O_i]$)

P_{ij} : durée réelle pour traiter OP_{ij}($i \in [1,n]$, $j \in [1,O_i]$)

PJ_{ijk} : $= 1$, si OP_{ij} est effectuée par M_k, $M_k \in MP_{ij}$
$= 0$, sinon.

$YP_{iji'j'k}$: $= 1$, si OP_{ij} est effectuée avant $OP_{i'j'}$ sur M_k ($PJ_{ijk} = 1$ $PJ_{i'j'k} = 1$) ;
$= 0$, sinon.

t_{ij}^- : date au plus tôt pour commencer TP_{ij} ($i \in [1,n]$, $j \in [1,O_i - 1]$)

t_{ij}^+ : date au plus tard pour commencer TP_{ij} ($i \in [1,n]$, $j \in [1,O_i - 1]$)

t_{ij} : date réelle de début du transport TP_{ij} ($i \in [1,n]$, $j \in [1,O_i - 1]$)

T_{ij} : durée réelle du transport TP_{ij} ($i \in [1,n]$, $j \in [1,O_i - 1]$)

TJ_{ijh} : $= 1$, si TP_{ij} est effectuée par R_h ($h \in [1,r]$) ;
$= 0$, sinon.

$YT_{iji'j'h}$: $= 1$, si TP_{ij} est effectuée avant $TP_{i'j'}$ sur R_h ($TJ_{ijh} = 1$, $TJ_{i'j'h=1}$) ;
$= 0$, sinon.

W_k^- : durée d'attente totale sur l'emplacement de stockage en amont M_k.

W_k^+ : durée d'attente totale sur l'emplacement de stockage en aval M_k.

us_{ij}^- : date au plus tôt de UST_{ij} ($i \in [1,n]$, $j \in [1,O_i]$)

us_{ij}^+ : date au plus tard de UST_{ij} ($i \in [1,n]$, $j \in [1,O_i]$)

us_{ij} : date réelle de UST_{ij} ($i \in [1,n]$, $j \in [1,O_i]$)

US_{ij} : durée réelle de UST_{ij} ($i \in [1,n]$, $j \in [1,O_i]$)

ds_{ij}^- : date au plus tôt de DST_{ij} ($i \in [1,n]$, $j \in [1,O_i]$)

ds_{ij}^+ : date au plus tard de DST_{ij} ($i \in [1,n]$, $j \in [1,O_i]$)

ds_{ij} : date réelle de DST_{ij} ($i \in [1,n]$, $j \in [1,O_i]$)

DS_{ij} : durée réelle de DST_{ij} ($i \in [1,n]$, $j \in [1,O_i]$)

Les temps de traitement sont bornés et non fixes, ce qui ne nous permet pas de calculer directement les dates réelles de début et de fin de chaque tâche de traitement et de transport. Il en va de même pour les temps d'attente aux places de stockage en amont ou en aval de chaque machine. Pour cette raison, nous avons introduit dans la notation des dates de début au plus tôt et au plus tard pour chaque tâche, que nous utiliserons ultérieurement.

Dans le modèle mathématique qui suit, nous supposons que n'importe quelle ressource de transport peut réaliser n'importe quelle tâche de transport. Donc $TR_{ij} = R$ ($i \in [1,n], j \in [1,O_i - 1]$). De plus, la date d'échéance est supposée être la même pour tous les travaux $d_i = d$ ($i \in [1,n]$).

3.3 Modélisation mathématique

Nous donnons ici une formulation mathématique de notre problème générique GFJSSP, ainsi qu'une évaluation de sa complexité en termes de nombre de variables et de contraintes. Le problème est ici traduit sous forme d'un modèle bi-objectif.

Un des objectifs, souvent rencontrés dans les problèmes d'ordonnancement de ces différents types d'ateliers, est la minimisation du *makespan* C_{max}. Or dans notre cas, un objectif supplémentaire nous a semblé important à prendre en compte, car il permet de réaliser une dichotomie de l'ensemble des problèmes traités. Il s'agit de la minimisation du temps de stockage. En effet, le stockage est autorisé sur certains systèmes, alors qu'il est strictement interdit pour d'autres. Nous considèrerons donc ces deux objectifs, qui sont exprimés comme suit :

Makespan :

$$C_{max} = max_{i\in[1,n]}\{c_i\} \tag{3.1}$$

où

$$\forall i \in [1,n], c_i = p_{iO_i} + P_{iO_i} \tag{3.2}$$

Stockage :

$$Stor = \sum_{i\in[1,n]} \sum_{j\in[2,O_i]} US_{ij} + \sum_{i\in[1,n]} \sum_{j\in[1,O_i-1]} DS_{ij} \tag{3.3}$$

où $\forall i \in [1,n]$, $\forall j \in [2,O_i]$, $US_{ij} = p_{ij} - t_{ij-1} - T_{ij-1}$, $\forall i \in [1,n]$, $\forall j \in [1,O_i-1]$, $DS_{ij} = t_{ij} - p_{ij} - P_{ij}$

Notre objectif est d'ordonnancer toutes les tâches de traitement et de transport afin de minimiser à la fois C_{max} et $Stor$, tout en respectant les contraintes suivantes :

- Contraintes de précédence entre deux tâches de traitement d'un même travail :
  $\forall i \in [1,n]$, $\forall j, j' \in [1,O_i], j \neq j' / U^i_{jj'} = 1$

$$p_{ij} + P_{ij} \leq p_{ij'} \tag{3.4}$$

- Chaque tâche de traitement peut être attribuée à une seule ressource de traitement :
  $\forall i \in [1,n]$, $\forall j \in [1,O_i]$,

$$\sum_{k\in MP_{ij}} PJ_{ijk} = 1 \tag{3.5}$$

- Chaque tâche de transport peut être attribuée à une seule ressource de transport :
  $\forall i \in [1,n]$, $\forall j \in [1,O_i-1]$,

$$\sum_{h\in[1,r]} TJ_{ijh} = 1 \tag{3.6}$$

- Contraintes de capacité pour chaque ressource de traitement :
  $\forall i, i' \in [1,n]$, $\forall j \in [1,O_i]$, $\forall j' \in [1,O_{i'}]$, $\forall k \in MP_{ij} \bigcap MP_{i'j'}$, et Mn est un très

grand nombre :

$$p_{ij} + P_{ij} \leq p_{i'j'} + (1 - YP_{iji'j'k}) \times Mn \tag{3.7}$$

$$p_{i'j'} + P_{i'j'} \leqslant p_{ij} + (YP_{iji'j'k} + 1 - PJ_{ijk}) \times Mn \tag{3.8}$$

– Relations entre les durées minimales, maximales et les durées réelles :
$\forall i \in [1, n], j \in [1, O_i]$

$$P_{ij}^{-} \leq P_{ij} \leq P_{ij}^{+} \tag{3.9}$$

$$US_{ij}^{-} \leq US_{ij} \leq US_{ij}^{+} \tag{3.10}$$

$$DS_{ij}^{-} \leq DS_{ij} \leq DS_{ij}^{+} \tag{3.11}$$

$\forall i \in [1, n], j \in [1, O_i - 1]$

$$T_{ij}^{-} \leq T_{ij} \leq T_{ij}^{+} \tag{3.12}$$

– Contraintes de capacité pour chaque ressource de transport :
$\forall i, i' \in [1, n]$, $\forall j \in [1, O_i - 1]$, $\forall j' \in [1, O_{i'} - 1]$, $\forall h \in TR_{ij} \bigcap TR_{i'j'}$, $\forall k, k_1, k', k'_1 \in [1, m]$, $PJ_{ijk} = 1, PJ_{ij+1k_1} = 1, PJ_{i'j'k'} = 1, PJ_{i'j'+1k'_1} = 1$, et Mn est un nombre assez grand :

$$t_{ij} + T_{ij} + \sigma_{k_1k'} \leq t_{i'j'} + (1 - YT_{iji'j'h}) \times Mn \tag{3.13}$$

$$t_{i'j'} + T_{i'j'} + \sigma_{k'_1k} \leq t_{ij} + (YT_{iji'j'h} + 1 - TJ_{ijh}) \times Mn \tag{3.14}$$

– Contraintes de déplacement (une ressource de transport doit avoir suffisamment de temps pour déplacer un travail entre deux opérations successives) :
$\forall i \in [1, n]$, $\forall j \in [1, O_i - 1]$,

$$p_{ij} + P_{ij} + DS_{ij} = t_{ij} \tag{3.15}$$

$$t_{ij} + T_{ij} + US_{ij+1} = p_{ij+1} \tag{3.16}$$

Il y a une contrainte supplémentaire pour les systèmes qui n'ont pas de stockage (par exemple : atelier de traitement de surface, chaîne d'approvisionnement alimentaire, etc.) : en effet, dans ce cas, une ressource de transport ne peut amener un produit sur une ressource encore occupée. Il faut d'abord décharger cette dernière par une ressource de transport, qui doit donc être disponible. Dans le cas mono robot, c'est impossible car la même ressource de transport ne peut amener un produit et en décharger un autre (sauf dans le cas de ressources spéciales, par exemple, un robot avec double préhenseur). Dans le cas multi-robots, vu l'encombrement de ces derniers, il faut laisser de la place au robot

qui décharge, avant qu'un autre amène la pièce suivante, d'où un léger laps de temps. Cette contrainte se traduit par la condition suivante, le temps de déplacement à vide ne peut être égal à zéro entre deux tâches de transport successives qui ne sont pas du même job. Cela revient à remplacer l'inégalité de l'équation 3.13 par une inégalité stricte quand $k_1 = k'$.

Ce modèle a la forme d'un programme linéaire mixte. Toutefois, son implémentation dans un logiciel de type CPLEX requiert l'introduction d'indices supplémentaires sur certaines variables. Cela est dû au problème d'affectation. Par exemple, un indice lié aux machines (respectivement aux robots) doit être ajouté aux variables liées aux tâches de traitement (respectivement aux tâches de transport). La nouvelle formulation résultante n'apporte pas d'information complémentaire à la compréhension du modèle, mais augmente de manière significative sa complexité en termes de nombre de variables et de contraintes. Cette formulation peut être trouvée à l'annexe A.

La complexité du modèle dépend du nombre de variables et du nombre de contraintes concernées. Le nombre total de variables du modèle dans sa forme initiale présentée ci-dessus est :
$(r + m) \times O^2 + (16 + m + r - 2 \times r \times n) \times O + r \times n^2 - r \times n + 2 \times m - 3 \times n$.

Le nombre de chaque type de contraintes est :

- pour l'Eq. 3.4 : $\sum_{i=[1..n]} O_i \times (O_i - 1)$
- pour l'Eq. 3.5 : O
- pour l'Eq. 3.6 : $(O - n)$
- pour l'Eq. 3.7 : $O^2 \times m$
- pour l'Eq. 3.8 : $O^2 \times m$
- pour l'Eq. 3.9 : O
- pour l'Eq. 3.10 : O
- pour l'Eq. 3.11 : O
- pour l'Eq. 3.12 : $O - n$
- pour l'Eq. 3.13 : $(O - n)^2 \times r$
- pour l'Eq. 3.14 : $(O - n)^2 \times r$
- pour l'Eq. 3.15 : $O - n$
- pour l'Eq. 3.16 : $O - n$

Le nombre total de contraintes est :
$\sum_{i=[1..n]} O_i \times (O_i - 1) + 2(m + r) \times O^2 + 4(2 - nr) \times O + 2r \times n^2 - 4n$.

Le tableau 3.1 donne quelques exemples numériques pour diverses tailles d'instances de benchmarks de la littérature que nous avons résolus dans les chapitres suivants. Les valeurs obtenues peuvent être considérées comme significatives. Elle correspondent à des problèmes sans affectation, et à des bornes inférieures pour les problèmes avec affectation.

TABLE 3.1 – Complexité du modèle en fonction de la taille des instances

nombre jobs n	nombre machines m	nombre robots r	nombre opérations O	nb. moyen d'opérations par job O_i	nombre de variables	nombre de contraintes
8	5	2	24	3	3 914	6 992
5	5	2	25	5	4 485	8 130
8	10	2	24	3	6 924	12 752
6	6	1	36	6	9 492	17 796
5	10	1	50	10	28 375	54 880
5	10	2	50	10	30 445	58 930
21	5	1	105	5	64 417	125 538
21	5	2	105	5	71 557	139 650
21	10	1	210	10	482 327	956 928
21	10	2	210	10	518 237	1 028 370
40	18	1	290 (280)	7	1 586 326	3 156 440

3.4 Modèle graphique

3.4.1 Graphe disjonctif

La représentation d'un problème d'ordonnancement sous forme de graphe fournit une bonne vision globale du problème. Elle permet notamment d'intégrer les relations entre les tâches et d'obtenir diverses informations temporelles. Un problème d'ordonnancement de type job shop est souvent modélisé par un graphe disjonctif $G = N \cup C \cup D$. De manière générale, celui-ci contient un ensemble de nœuds N, un ensemble d'arcs conjonctifs C et un ensemble d'arcs disjonctifs D. Les nœuds représentent les tâches, les arcs conjonctifs sont associés aux contraintes conjonctives du problème (telles que les contraintes de précédence liées aux gammes opératoires). Enfin les arcs disjonctifs symbolisent les contraintes disjonctives : par exemple, deux tâches affectées à la même ressource ne peuvent être réalisées simultanément. Ainsi, il faudra donner la priorité d'exécution à l'une des tâches, sans savoir a priori laquelle sera prioritaire. C'est le résultat du séquencement qui permettra de statuer. Lorsqu'une telle décision est prise, ladite contrainte devient conjonctive et l'arc associé devient orienté. On dit que la disjonction a été arbitrée.

En ce qui concerne la problématique qui nous intéresse ici, nous trouvons dans la littérature des exemples de *job shop* avec transport modélisés par des graphes disjonctifs. Hurink et Knust, Lacomme et al. ([Hurink and Knust, 2005], [Knust, 1999], [Lacomme et al., 2007], [Lacomme et al., 2010]) ont utilisé un graphe disjonctif pour représenter un problème de *job shop* avec une ou plusieurs ressources de transport, et en intégrant des nœuds associés aux traitements et des nœuds associés aux tâches de transport. Cependant, ce graphe est limité à des temps de traitement et de transport fixes. Dans le cas du HSP, Bloch ([Rossé-Bloch, 1999]) a proposé un graphe disjonctif qui contient des arcs négatifs

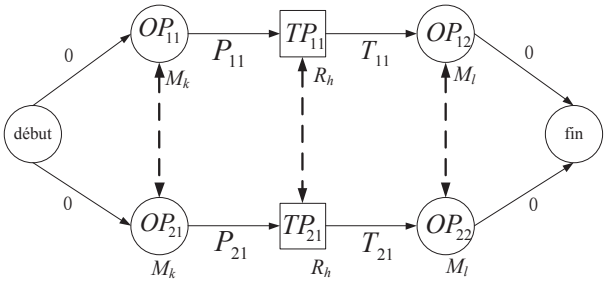

FIGURE 3.1 – Graphe disjonctif utilisé dans [Hurink and Knust, 2005] et [Lacomme et al., 2007]

pour modéliser les temps opératoires maximaux. Dans la suite, et pour le distinguer de notre graphe, on l'appellera le *graphe simple*. C'est un graphe qui contient uniquement des nœuds opératoires. Pour prendre en compte les bornes minimales et maximales des durées opératoires :

- les arcs positifs représentent les contraintes de succession liées aux travaux, avec des valeurs égales à la somme du temps opératoire minimal et du temps de transport en charge entre les deux nœuds opératoires.

- les arcs négatifs représentent les contraintes de durées maximales pour les temps de trempe. Leur valeur est égale à l'opposé de la somme de la durée maximale et du temps de déplacement en charge.

Bien qu'il n'y ait pas de nœud de transport dans ce graphe, le temps de transport en charge est pris en compte en l'ajoutant à la valeur entre deux sommets successifs, mais pas les temps de déplacement à vide. Cette représentation évite d'indiquer la durée réelle de la tâche, qui n'est pas connue *a priori*. Ce graphe représente efficacement le séquencement des ressources de traitement et la quantité suffisante de ressources de transport disponibles. Il ne représente cependant pas les problèmes de séquencement du robot (ressources de transport). En effet, il considère l'hypothèse suivante : il existe assez de robots pour transporter les produits quand on en a besoin. Par ailleurs, ce graphe ne peut modéliser d'autres problèmes que le HSP dans la mesure où les temps de stockage sont interdits. Les figures 3.1 et 3.2 montrent ces deux types de graphes disjonctifs pour un exemple à deux *jobs*.

De manière générale, les cas traités de *job shop* avec transport ne sont à chaque fois que des cas particuliers de notre problème générique. Les représentations graphiques associées sont alors insuffisantes pour modéliser l'ensemble des caractéristiques et contraintes qui nous intéressent. Pour intégrer tous ces éléments et contraintes, nous avons étendu les graphes précédents, en partant plus particulièrement du graphe simple de Bloch ([Rossé-Bloch, 1999]). Nous obtenons alors un graphe disjonctif modifié que nous appelons graphe disjonctif générique. Il intègre au niveau des noeuds les différentes tâches possibles que

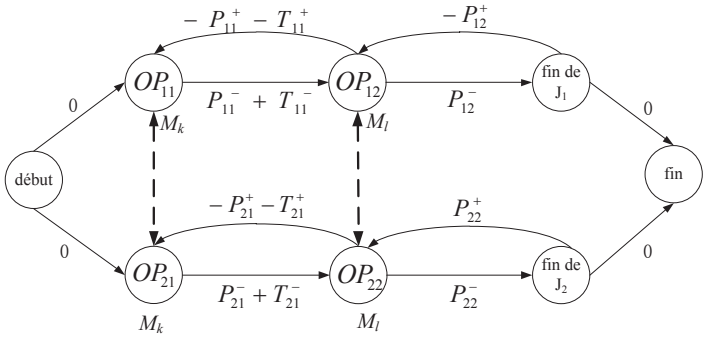

FIGURE 3.2 – Graphe disjonctif utilisé dans [Rossé-Bloch, 1999]

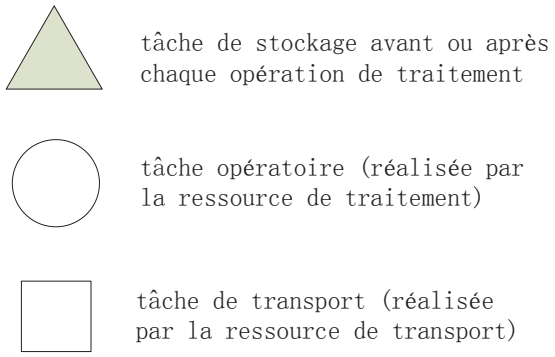

FIGURE 3.3 – Les différents types de noeuds de notre graphe générique

sont le traitement opératoire à proprement parler, le transport et le stockage. Les symboles utilisés pour ces différents types de noeuds sont présentés dans la figure 3.3. La figure 3.4 montre qu'un *job* donné est représenté dans le graphe par la succession de sous-ensembles de noeuds suivants : stockage entrée- traitement- stockage sortie - transport.

Le graphe doit permettre de représenter les caractéristiques des divers ateliers considérés : atelier flexible FMS, cellule robotisée RC et atelier de traitement de surface ATS. Ces caractéristiques sont renseignées par la valeur des paramètres US_{ij}^{+}, DS_{ij}^{+}, T_{ij}^{+} et P_{ij}^{+} indiqués sur les arcs (tableau 3.2). Ainsi dans les ATS et certains RC pour lesquels aucun stock n'est autorisé, $US_{ij}^{+} = DS_{ij}^{+} = 0$. Sinon, dans les FMS et autres RC, $US_{ij}^{+} = DS_{ij}^{+} = \infty$. Dans tous les cas, l'arc depuis un noeud de traitement vers le noeud de stock de sortie suivant est valué par P_{ij}^{-} qui correspond à la durée de traitement fixe ou minimale selon les cas. Par contre, l'arc depuis un noeud de stock de sortie vers le noeud de traitement précédent est valué par P_{ij}^{+} qui peut prendre une valeur fixe ou infinie suivant le cas. Lorsque le temps de transport en charge n'est pas borné, cela signifie qu'on autorise des pauses en charge pour les ressources de transport. Dans la plupart des

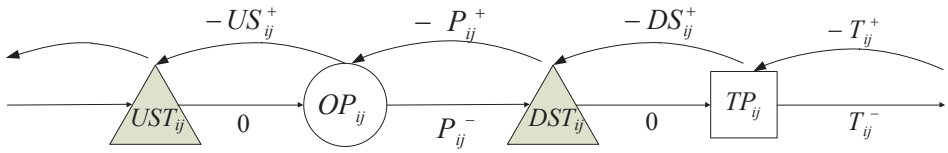

FIGURE 3.4 – Représentation d'un élément de la gamme d'un job par le graphe générique

TABLE 3.2 – Valeurs des paramètres suivant le type de problème

Caractéristiques	Contraintes	systèmes associés	US_{ij}^{+}	P_{ij}^{+}	DS_{ij}^{+}	T_{ij}^{+}
durées opératoires	bornées	certains RCs,ATS		$> P_{ij}^{-}$		
	fixes	FMS, certains RCs		$= P_{ij}^{-}$		
	non bornées	FMS,ATS,certains RCs		$= +\infty$		
stock	sans attente	certains RCs,ATS	$= 0$		$= 0$	
	avec stockage	FMS, certains RCs	$= +\infty$		$= +\infty$	
durée de transport en charge	fixe	FMS,RCs,ATS				$= T_{ij}^{-}$
	non bornée	FMS,certain RCs				$= +\infty$

cas, le temps de transport en charge est fixe.

Rappelons que dans notre étude, nous supposons que :

- la capacité des stocks, s'ils existent, est supposée illimitée ;
- les temps de préparation sont supposés inclus dans les durées opératoires ;
- les temps de manutention entre un stock (amont ou aval) et une ressource (de traitement ou de transport) sont inclus dans la durée opératoire minimale ou le temps de transport.

Ces divers éléments nous ont amenés à définir notre graphe disjonctif modifié $G = N \cup C \cup D$, tel que :

- l'ensemble de nœuds $N = NM \cup NR \cup NS$ est composé de :
  - l'ensemble des nœuds opératoires $NM = NM_O \cup N_f$. NM_O contient les nœuds $N_1, N_2, ..., N_O$ associés à chaque traitement ; l'ensemble N_f contient N^* et N^{**} qui représentent les évènements début et fin pour l'ensemble des travaux,
  - l'ensemble des nœuds de transport NR ($|NR| = O - n$),
  - l'ensemble des nœuds NS associés à chaque tâche d'attente aux places de stockage d'entrée et de sortie ($|NS| = 2O$) ;
- l'ensemble des arcs conjonctifs C qui représentent les contraintes de précédence induites par la gamme opératoire de chaque travail. Compte tenu de la succession des tâches : attente/traitement/attente/transport, associée à chaque opération de la gamme, ces contraintes se traduisent par des arcs entre chaque nœud de traitement ou transport et la place de stockage d'entrée ou sortie qui le suit. Les contraintes conjonctives exprimant le respect des durées opératoires minimales et maximales sont traduites par des arcs de valeur respectivement positive et négative, comme expliqué précédemment :

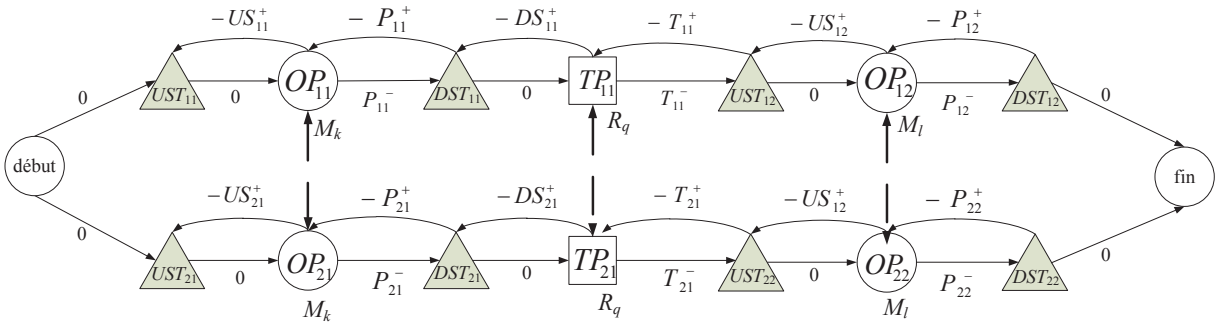

FIGURE 3.5 – Graphe disjonctif générique pour un exemple à deux jobs

- les arcs successifs $UST_{ij} \rightarrow OP_{ij} \rightarrow DST_{ij} \rightarrow TP_{ij} \rightarrow UST_{ij+1}$, $i \in [1,n], j \in [1,O_i]$, représentent les conjonctions associées à un même travail,
- $UST_{ij} \leftarrow OP_{ij} \leftarrow DST_{ij} \leftarrow TP_{ij} \leftarrow UST_{ij+1}$, $i \in [1,n], j \in [1,O_i]$ représente les durées maximales des tâches du travail,
- $N^* \rightarrow UST_{i1}$, $i \in [1,n]$: conjonction entre le départ et chaque travail,
- $DST_{iO_i}, i \in [1,n] \rightarrow N^{**}$, $i \in [1,n]$: conjonction entre chaque travail et la fin ;

– l'ensemble des arcs disjonctifs $D = DM \cup DR$ traduisant les contraintes disjonctives induites par le partage d'une même ressource par plusieurs tâches. Cet ensemble contient :
  - l'ensemble DM des disjonctions sur les ressources de traitement ; sur M_k, les tâches OP_{ij} et $OP_{i'j'}$ (avec $PJ_{ijk} = 1$ et $PJ_{i'j'k} = 1$) ne peuvent être réalisées simultanément,
  - l'ensemble DR des disjonctions sur les ressources de transport. Sur R_h, les tâches TP_{ij} et $TP_{i'j'}$ (avec $TJ_{ijh} = 1$ et $TJ_{i'j'h} = 1$) ne peuvent être transportées simultanément.

La figure 3.5 montre un graphe disjonctif générique pour un exemple à deux jobs.

3.4.2 Arbitrage des disjonctions

Dans notre problème, les disjonctions sont dues à la présence de ressources disjonctives (machines et ressources de transport), qui ne peuvent réaliser qu'une seule tâche à la fois.

Disjonction sur traitement

Traditionnellement, l'arbitrage de la disjonction consiste à choisir une orientation pour l'arc disjonctif, et à affecter à cet arc la durée de l'opération en amont (durée minimale dans notre cas). Or ici deux problèmes se posent. D'une part, le fait d'avoir des durées minimales et maximales implique que cet arbitrage impose la création de l'arc «retour» négatif correspondant à la durée maximale de l'opération exécutée en premier. D'autre

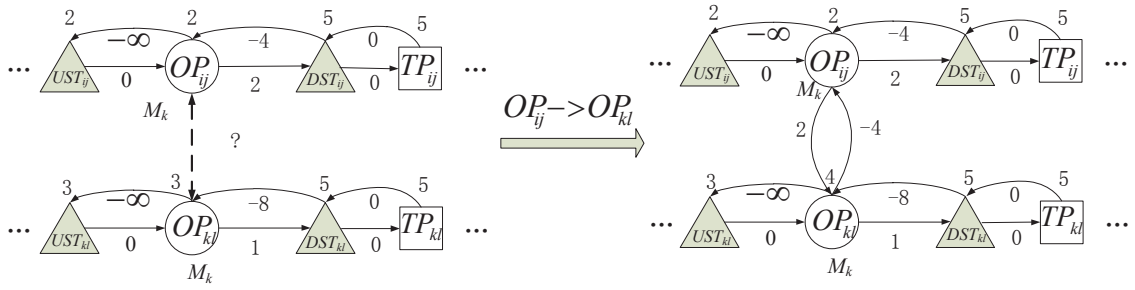

FIGURE 3.6 – Arc disjonctif entre deux nœuds de traitement et une possibilité d'arbitrage

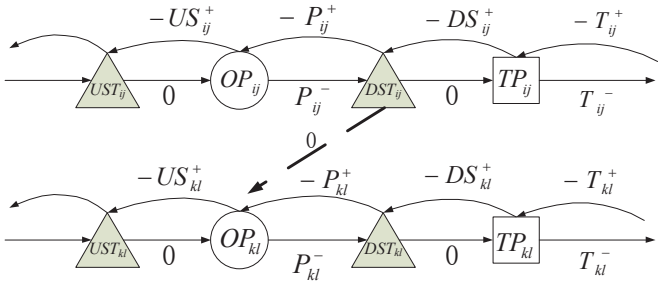

FIGURE 3.7 – Arbitrage de la disjonction entre deux tâches opératoires : OP_{ij} avant OP_{kl}

part, le fait d'avoir une fenêtre temporelle et non une durée fixe (max≥min) implique que dans le graphe, la valeur effective entre la fin d'une opération et sa mise en stock peut être différente de la valeur entre cette fin et le début de l'opération suivante sur la machine ! La figure 3.6 illustre un exemple de ce cas. La figure de gauche montre une partie du graphe générique avec les dates de début au plus tôt (notées en dessus de chaque nœud) obtenues par le graphe actuel. Une disjonction entre OP_{ij} et OP_{kl} y est arbitrée par $OP_{ij} \rightarrow OP_{kl}$ (figure de droite). Les temps opératoires minimal et maximal sont fixes. p^-_{ij} et p^-_{kl} sont les dates au plus tôt de OP_{ij} et OP_{kl} respectivement. Comme $p^-_{ij} + P^-_{ij} > p^-_{kl}$ $(2+2>3)$, p^-_{kl} doit être mise à jour par $p^-_{ij} + P^-_{ij} = 2+2 = 4$. Mais en réalité, la date de fin de OP_{ij} sur M_k correspond à la date d'entrée dans le stock aval DST_{ij}. Cela détermine le temps de traitement réel pour OP_{ij} qui est $ds_{ij} - p^-_{ij} = 5-2 = 3$, ce qui veut dire que OP_{kl} doit commencer au plus tard à la date $p^-_{ij} + 3 = 5$ (au lieu de 4).

Pour éviter ces deux inconvénients, nous avons opté pour une deuxième représentation de ce type d'arbitrage. Elle consiste à ajouter un arc entre la place de stockage aval de la première opération et la place correspondant à la 2ème opération. De ce fait, un seul arc suffit, de valeur nulle, indiquant le temps entre la fin d'une opération et le début de la suivante sur la machine considérée (figure 3.7).

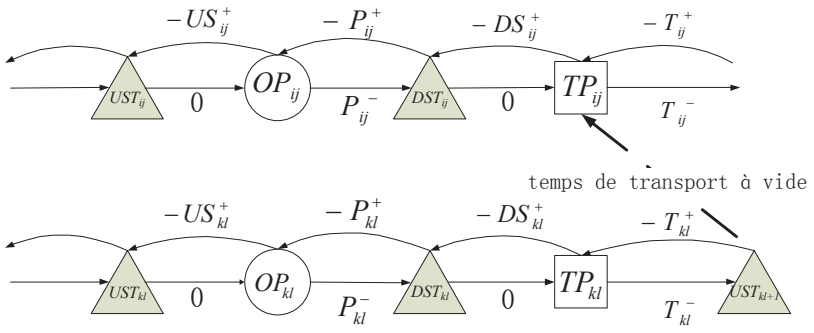

FIGURE 3.8 – Arbitrage de la disjonction entre deux tâches de transport

Disjonction sur transport

Dans la figure 3.5, il y a une disjonction entre deux tâches de transport (TP_{11} et TP_{21}) effectuées par la même ressource de transport (R_q). Nous supposons que l'arbitrage de la disjonction donne le résultat $TP_{kl} \rightarrow TP_{ij}$ (voir figure 3.8). Ici, nous avons fait le même choix de représentation que pour les arcs de traitement disjonctifs, par souci d'homogénéité, mais aussi pour le même type de problème que précédemment dans le cas d'un transport dit avec stock autorisé en charge.

Conséquence des arbitrages

Il est important de noter que l'arbitrage des disjonctions est dépendant des choix effectués au niveau des autres ressources. Les tâches d'un travail sont liées par les contraintes de gamme. Les arcs disjonctifs lient les différents travaux. De plus, les choix de l'ordre de passage définissent l'orientation des arcs disjonctifs après arbitrage. Ceci peut créer un ou plusieurs circuits de longueur positive. Pour tout problème de type *jobshop*, on sait bien qu'une solution n'est pas faisable quand il y a un circuit positif dans le graphe associé. Quand des arcs négatifs sont ajoutés dans un graphe, le risque d'avoir des circuits positifs est encore plus important. Il est donc d'autant plus difficile de les éviter. Cela veut aussi dire que les problèmes d'ordonnancement avec des contraintes de temps bornés sont plus difficiles à résoudre, car l'espace de recherche contient moins de solutions faisables.

En général, il existe deux cas possibles pour que ce phénomène arrive :

1. un interblocage causé par les choix d'arbitrage de la séquence sur différentes ressources ;
2. un circuit de longueur positive liée aux limites maximales de durées de traitement. Le non respect de la borne maximale affecte la faisabilité de la solution.

Ces deux cas sont présentés dans les figures 3.9 et 3.10. Dans la première figure, deux disjonctions de deux paires de tâches de traitement sont arbitrées par : $OP_{ij+1} \rightarrow OP_{kl} \rightarrow OP_{ij}$. Il y a un circuit qui viole les contraintes de précédence. Dans la seconde figure, les disjonctions entre les trois tâches de transport sont arbitrées par : $TP_{ij} \rightarrow TP_{kl} \rightarrow$

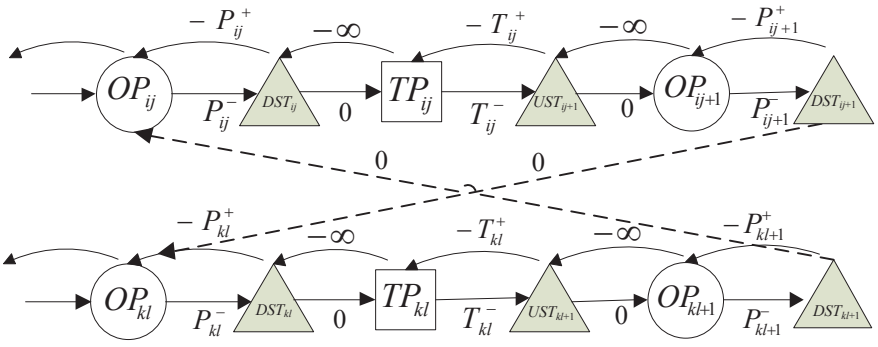

FIGURE 3.9 – Premier cas de circuit positif.

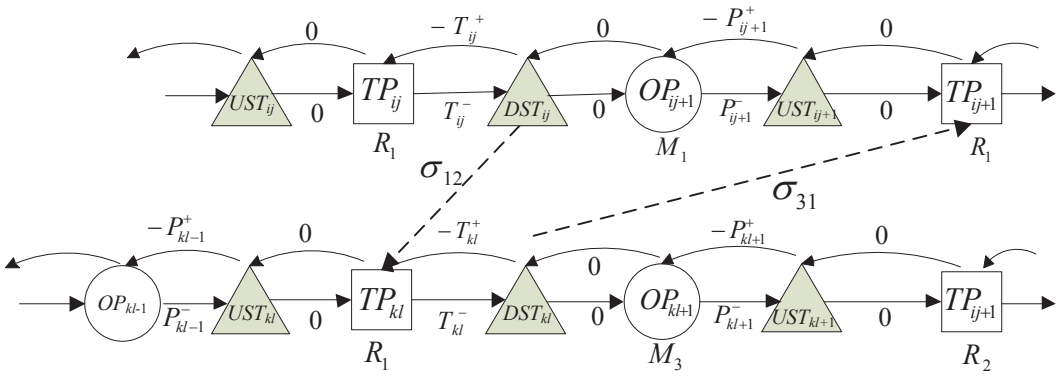

FIGURE 3.10 – Second cas de circuit positif.

TP_{ij+1}. Il y a un circuit : $UST_{ij+1} \rightarrow TP_{kl} \rightarrow UST_{kl+1} \rightarrow TP_{ij+1} \rightarrow DST_{ij+1} \rightarrow OP_{ij+1} \rightarrow UST_{ij+1}$. C'est un circuit positif si la condition suivante est satisfaite :

$$\sigma_{12} + T_{kl}^{-} + \sigma_{31} - P_{ij+1}^{+} > 0 \tag{3.17}$$

Dans cet exemple, le circuit contient les arcs obtenus après arbitrage d'une disjonction liée aux tâches de transport. Il est bien sûr possible d'avoir le même problème avec des tâches de traitement et/ou des tâches de transport. Notons que, quand les durées maximales sont assez courtes par rapport aux durées des tâches, il est plus facile d'avoir des circuits positifs. Quand les durées maximales sont infinies, il n'y aura plus de circuit de ce type et la solution devient faisable dans le cadre d'un *jobshop* classique avec transport.

Dans le chapitre suivant, nous détaillerons la manière dont nous remédions à ce prob-

lème de circuits positifs dans notre procédure de résolution.

3.4.3 Comparaison des graphes disjonctifs

Dans cette partie, nous comparons la capacité des divers graphes simple et générique pour représenter les solutions. L'avantage de séparer les sommets opératoires et les sommets de transport est d'offrir la possibilité de traiter le séquencement sur les ressources de transport. Dans le graphe générique, le fait de séparer ces nœuds autorise la représentation de toutes les informations relatives à l'ensemble des ressources. La représentation de l'ensemble des contraintes nous permet alors de résoudre l'ordonnancement des machines, puis l'ordonnancement des robots. Le graphe simple de Bloch présente ainsi moins de solutions que le graphe générique. Pour expliquer cela, nous utilisons un exemple de 3 jobs, 4 machines (M_1,M_2,M_3,M_4) et un poste de chargement M_0. Les données sont fournies dans le tableau 3.3. Les temps de déplacement en charge et à vide sont d'une unité de temps.

TABLE 3.3 – Données d'un exemple

job	temps opératoires			
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}
1	$M_0 : [0, \infty]$	$M_1 : [8, 15]$	$M_2 : [16, 23]$	$M_4 : [0, \infty]$
2	$M_0 : [0, \infty]$	$M_1 : [20, 28]$	$M_3 : [10, 21]$	$M_2 : [0, \infty]$
3	$M_0 : [0, \infty]$	$M_4 : [14, 32]$	$M_2 : [0, \infty]$	

Un ordonnancement faisable des ressources de traitement est :
$M_0 : OP_{11} \rightarrow OP_{21} \rightarrow OP_{31}$
$M_1 : OP_{12} \rightarrow OP_{22}$
$M_2 : OP_{13} \rightarrow OP_{24} \rightarrow OP_{33}$
$M_3 : OP_{23}$
$M_4 : OP_{32} \rightarrow OP_{14}$

Cet ordonnancement est représenté sous forme de trois graphes disjonctifs dans les figures 3.11 à 3.13 :

Graphe simple (figure 3.11) : dans ce graphe, il y a un circuit positif en traits pointillés épais. Cela veut dire qu'avec un algorithme qui peut évaluer et détecter le circuit positif dans un graphe avec les arcs négatifs (par exemple, l'algorithme de Dantzig ou de Bellman-ford, ici, nous avons choisie le premier, voir l'annexe B), cette solution sera éliminée en tant que solution infaisable. Les explications de ce phénomène sont :

- dans un graphe simple, il n'y pas de nœud de transport, ou on peut dire que la tâche de transport est réunie avec sa tâche opératoire précédente. L'arbitrage entre deux opérations de traitement se traduit par un arc qui part depuis la tâche suivante du premier nœud, mais en réalité, il part depuis la fin de la tâche de

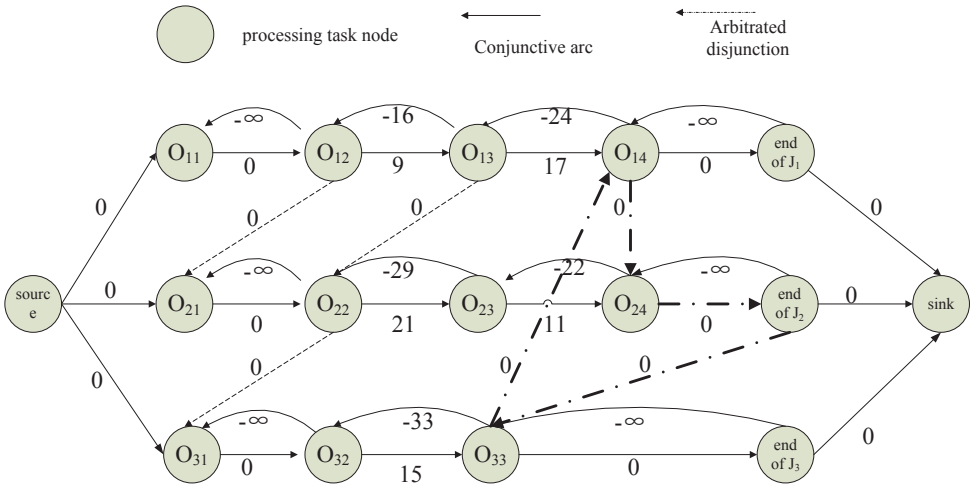

FIGURE 3.11 – Représentation d'un ordonnancement par le graphe simple utilisé par [Rossé-Bloch, 1999]

transport après la première tâche opératoire. La conséquence de cette représentation est une augmentation du risque de former des circuits positifs, alors que certaines des séquences associées seraient en fait faisables.

- le fait de pénaliser le temps de transport à vide en affectant la valeur zéro à ces arcs arbitrés ne correspond pas à une situation réelle. De plus, cela augmente le risque d'avoir un circuit positif.

Graphe classique avec nœuds de transport (figure 3.12) : il n'y pas de circuit, mais toutes les contraintes de bornes maximales ne sont pas prises en compte. De plus, les valeurs sur les arcs arbitrés ne peuvent pas être déterminés à priori.

Graphe générique (figure 3.13) : ce type de graphe sépare les tâches opératoires, les tâches de transport, et aussi les postes de stockage. La solution est présentée sans circuit positif. Les contraintes de temps opératoires maximales sont aussi représentées. Le départ d'un arbitrage est fait depuis le nœud de stockage aval associé. Il ne ferme pas directement un circuit grâce aux arcs négatifs entre les nœuds de stockage, de transport, et le noeud opératoire, ce qui diminue beaucoup le risque d'existence de circuit positif.

On voit que le graphe générique est capable de représenter plus de solutions que le graphe simple. Le graphe simple ne peut modéliser que les solutions sans stockage. Le graphe générique peut représenter les solutions avec ou sans stockage. Cela nous permet de déplacer un travail d'une machine (ou véhicule) vers un stock quand une tâche de traitement ou de transport est achevée, en attendant la disponibilité de la ressource suivante. Un autre avantage est de faire apparaître dans le graphe les fenêtres temporelles de chaque tâche, ainsi que les contraintes de stock. Le tableau 3.4 résume les différences entre les graphes disjonctifs.

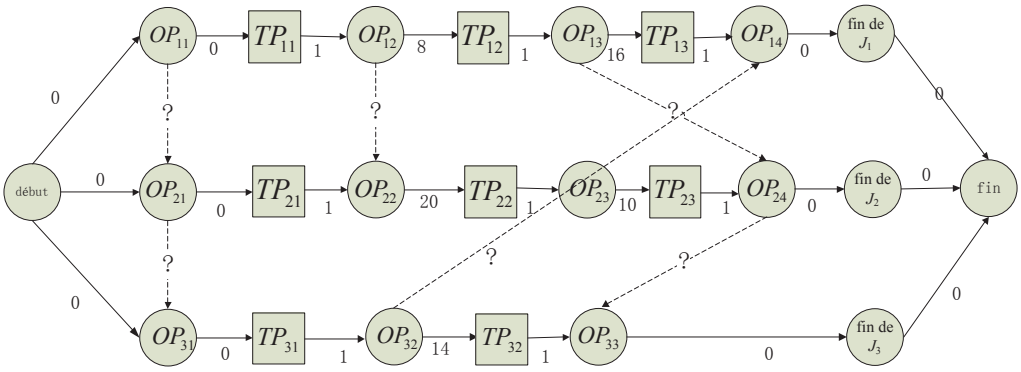

FIGURE 3.12 – Représentation d'un ordonnancement par le graphe utilisé par [Hurink and Knust, 2005]

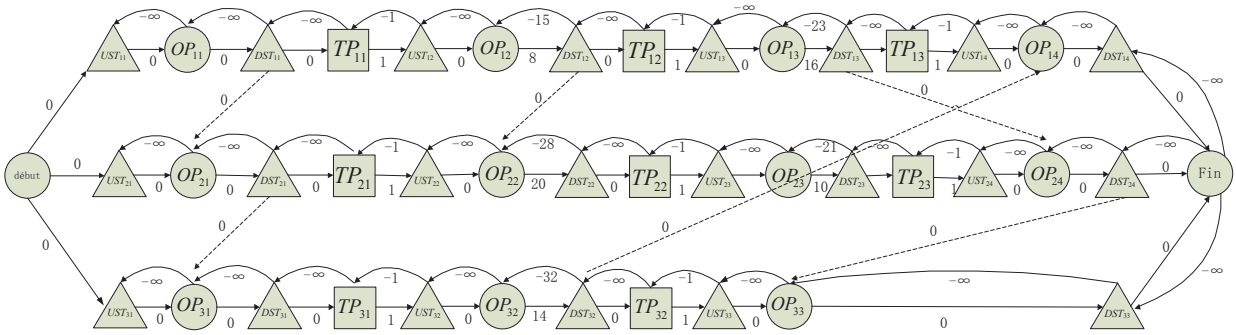

FIGURE 3.13 – Représentation d'un ordonnancement par notre graphe générique

TABLE 3.4 – Capacité de représentation des trois graphes

Graphe	type de solutions représent		temps de transport		Contrainte	Contrainte
	sans stock	avec stock	en charge	à vide	de stock	temps bornés
Bloch	oui (pas tout)	non	oui	non	non	oui
Hurink	pas assuré	oui	oui	oui	non	non
générique	oui	oui	oui	oui	oui	oui

3.5 Bilan

Dans ce chapitre, nous avons proposé deux modèles dits génériques pour le problème d'ordonnancement de type *job shop* flexible avec contraintes de transport et temps opératoires bornés, que nous avons appelé *General Flexible Job Shop Scheduling Problem* (GFJSSP). Nous avons montré que les différentes classes de systèmes identifiés et problèmes associés s'inscrivent dans ces deux modèles.

Le premier est un modèle mathématique qui exprime l'ensemble des caractéristiques, contraintes et objectifs de notre problème. Il est clair que la généralisation des contraintes n'induit pas une diminution de complexité par rapport aux problèmes spécifiques que le modèle intègre (par exemple, le temps de traitement devient borné au lieu de fixe, l'ensemble de ressources qui peut effectuer une tâche au lieu d'une ressource, ...). De ce fait, la résolution de ce modèle par une méthode exacte en un temps raisonnable est peu vraisemblable.

Le deuxième modèle est basé sur un graphe disjonctif fournissant un autre type de représentation de l'ensemble des caractéristiques et contraintes du problème générique considéré. Comparé aux autres types de graphes disjonctifs, ce graphe générique a plusieurs avantages : les contraintes de temps opératoire et de transport sont bien prises en compte, il est capable de représenter plus de solutions faisables et le niveau des stocks peut aussi être évalué. Cependant, le nombre de noeuds est aussi plus important, ce qui rend ce graphe plus complexe en termes d'évaluation. Nous nous appuierons en partie sur ce graphe dans notre procédure de résolution.

Cette analyse a guidé certains des choix apparaissant dans les chapitres suivants, en termes d'approches de résolution. En effet, compte tenu de la complexité du problème et du modèle mathématique associé, nous avons opté pour des méthodes approchées plutôt qu'exactes. Par ailleurs, pour chacun des types d'ateliers considérés, nous avons identifié dans la littérature des algorithmes efficaces. Dans ce contexte, et dans le cadre préliminaire d'une étude de faisabilité, nous nous sommes appuyés et inspirés de certaines de ces approches dédiées, en les adaptant pour la résolution de notre problème d'ordonnancement générique. La principale question est de vérifier si dans un nouveau contexte applicatif, ces approches conservent ou non leur efficacité.

Pour cela, nous avons procédé en deux étapes, en décomposant le problème en deux sous-problèmes classiques : d'abord celui du séquencement, que nous avons étudié en considérant que l'affectation avait été définie au préalable ; puis, nous avons ajouté la problématique de l'affectation des tâches aux ressources, classiquement rencontrée dans les systèmes dits flexibles. La résolution de ces deux sous-problèmes fait l'objet des deux chapitres suivants.

Chapitre 4

Séquencement

4.1 Introduction

Dans cette partie, nous considérons les problèmes de séquencement des ressources (ressources de traitement et de transport) en supposant qu'une affectation des ressources a déjà été réalisée au préalable pour toutes les tâches de traitement. Nous rappelons que les ressources de transport sont supposées être capables d'effectuer n'importe quelle tâche de déplacement de produit. L'approche décrite ici ne prend donc pas en compte le problème d'affectation, ou en tout cas pas pour les ressources de traitement. Ce problème sera abordé dans le chapitre suivant, où nous combinerons plusieurs approches pour résoudre à la fois les problèmes d'affectation et de séquencement de l'ensemble des tâches.

Dans le contexte d'un problème prédictif, le séquencement des machines impacte de manière significative le rendement du système. Le problème de *job shop* classique ne prend pas en compte le transport et ne s'intéresse qu'au séquencement sur les ressource de traitement. Dans notre problème générique, il faut déterminer non seulement le séquencement des machines, mais également celui des ressources de transport. En effet, si dans un HSP, ordonnancer les opérations revient à ordonnancer le transport du fait des contraintes de non attente et non stock, cela n'est pas vrai dans le cas général.

L'état de l'art du chapitre 1 a montré que la majorité des approches proposées résolvent des problèmes avec des temps opératoires fixés. Par ailleurs, la plupart des méthodes dédiées au HSP, qui considèrent le transport et les temps opératoires bornés, concernent des cas cycliques ou dynamiques. En outre, relativement peu d'instances de *job-shop* peuvent être trouvées, hormis quelques benchmarks bien connus.

Dans le cadre de notre démarche d'étude de faisabilité, nous avons identifié plusieurs méthodes récentes et efficaces utilisées pour les trois systèmes considérés.

La première est basée sur une recherche Tabou, utilisée par [Hurink and Knust, 2005] dans le cas d'un atelier avec un seul robot, et sans problème d'affectation. La définition du voisinage est basée sur l'identification de blocs-machine et blocs-robot. Il s'agit d'ensembles de tâches opératoires ou de transport successives exécutées par une même ressource

et appartenant au chemin critique. Les échanges réalisés pour générer les voisins sont effectués au sein d'un même bloc. Ils sont limités par le nombre de blocs et le nombre de tâches dans chaque bloc. Cette procédure s'est révélée efficace dans le cas d'un seul robot. Mais quand un problème d'affectation se pose, alors les tâches sont réparties sur les différentes ressources possibles. En conséquence, il existe moins de blocs et/ou moins de tâches dans chaque bloc, ce qui peut limiter l'amélioration de la solution courante. En s'inspirant en partie de la technique de construction du voisinage de [Hurink and Knust, 2005], [Lacomme et al., 2010] ont proposé un algorithme mémétique pour résoudre les problèmes d'ordonnancement dans un FMS sans affectation des tâches opératoires et avec une ou plusieurs ressources de transport. Ils utilisent un graphe disjonctif contenant des sommets associés à l'ensemble des tâches (opératoire et de transport) et des valeurs fixes sur les arcs. Cependant, ces méthodes n'intègrent pas les problèmes avec temps de traitement bornés et/ou sans stockage autorisé.

Le deuxième type de méthode sur lequel nous nous sommes basés est également de type recherche locale itérative. [Deroussi et al., 2008] ont proposé une méta-heuristique adaptée aux ateliers flexibles (FMS) avec plusieurs ressources de transport. Cette heuristique repose sur l'utilisation et le classement de différents mouvements pour générer un sous-ensemble de voisins de la solution courante. Elle permet d'améliorer certaines solutions avec temps d'attente.

Ces deux méthodes sont efficaces dans les cas pour lesquels elles ont été élaborées. Toutefois, elles ne peuvent pas s'appliquer directement aux cas plus généraux, par exemple aux temps de traitement bornés ou aux problèmes sans stock. [Rossé-Bloch, 1999] a appliqué une procédure de *shifting bottleneck* modifiée pour résoudre un HSP prédictif. Elle est combinée avec la représentation d'un graphe disjonctif qui intègre la contrainte de temps de traitement maximal. La faisabilité de la solution est vérifiée par l'algorithme de Dantzig. Une réparation simple est utilisée si la solution n'est pas faisable. Cette procédure est efficace quand le nombre de tâches sur une ressource est limité. Les temps de déplacement en charge sont pris en compte, mais pas les temps de déplacement à vide. De plus, les séquencements des ressources de transport ne sont pas représentés, donc le nombre de ressources de transport est supposé de fait illimité.

Pour résoudre le problème de séquencement de notre atelier générique, sous les contraintes, hypothèses, et objectifs énoncés au chapitre précédent, nous avons adapté ces méthodes dédiées à chacun des trois systèmes étudiés. Dans la suite de ce chapitre, nous proposons un algorithme de recherche Tabou dans lequel la définition de voisinage est inspirée de celle de [Deroussi et al., 2008] (section 4.2). Dans un deuxième temps, nous détaillons une seconde approche de séquencement qui est une procédure de shifting bottleneck modifiée et combinée avec des heuristiques de réparation (section 4.3). Nous analyserons l'efficacité de chacune de ces méthodes sur des benchmarks de la littérature.

4.2 Procédure de recherche locale itérative

4.2.1 Principes de recherche Tabou

Dans une première étape, nous avons développé un algorithme de recherche de type Tabou. La recherche tabou est une métaheuristique itérative d'optimisation, qualifiée de recherche locale au sens large, et proposée par Fred W. Glover ([Glover, 1990]). Le principe de cette approche consiste, à partir d'une position donnée dans l'espace des solutions, à en explorer le voisinage et à choisir la position dans ce voisinage qui minimise la fonction objectif. Dans l'espace de recherche, les solutions voisines d'une solution sont donc déterminées par un voisinage mais aussi par l'utilisation de structures de mémoire. Celles-ci servent à stocker des solutions ayant été recherchées récemment pour que l'algorithme ne les visite pas à plusieurs reprises. Les dernières positions explorées sont alors dites tabou (interdites) pendant un nombre donné d'itérations (égal à la taille de la liste les contenant). En théorie, la liste tabou doit conserver des positions complètes, ce qui dans certains types de problèmes, peut nécessiter l'archivage d'une grande quantité d'informations. Cette difficulté peut être contournée, comme nous le faisons ici, en ne gardant en mémoire que certaines paires de mouvements précédents. L'algorithme Tabou se déplace itérativement d'une solution s à une autre solution s' dans un voisinage défini de s, jusqu'à ce qu'un critère d'arrêt ait été satisfait. Normalement, celui-ci correspond à un nombre limité d'itérations. Quand ce nombre est atteint, la recherche s'arrête et renvoie la meilleure solution rencontrée au cours de l'exploration. Le principe d'un algorithme Tabou est décrit dans l'algorithme 1.

Algorithm 1 Recherche locale de type Tabou

```
Solution initiale s_0
s* ← s_0
while critère d'arrêt n'est pas atteint do
  RechercheLocale (s_0), trouver tous les voisins de s_0 en se basant sur le système de
  voisinage défini.
  évaluer les voisins (voir algorithme 4 dans la section 4.2.3)
  choisir la meilleure solution s' parmi tous les voisins qui ne sont pas dans la liste
  tabou
  s_0 ← s'
  if s_0 est meilleur que s* then
    s* ← s_0
    if la liste tabou est pleine then
      supprimer la première paire dans la liste tabou.
    end if
    mettre s* à la fin de la liste tabou.
  end if
end while
return s*
```

Nous utilisons cette procédure de recherche tabou pour améliorer une solution donnée. Les différentes étapes de notre procédure Tabou sont schématisées dans la figure 4.1. Ici la séquence initiale est associée à une affectation donnée de toutes les tâches aux ressources.

4.2.2 Génération de la solution initiale

Une solution de notre problème générique est représentée par l'ensemble des séquences de tâches sur chaque ressource (de traitement et de transport).

La solution initiale définit le point de départ de l'exploration de l'espace de recherche. Elle influence donc la vitesse d'obtention de la meilleure solution, de même que la capacité à atteindre cette meilleure solution. La façon de choisir une bonne solution de départ dépend souvent des caractéristiques du problème et n'est donc pas toujours évidente. Nous avons choisi ici de ne pas nous focaliser sur la recherche d'une bonne solution initiale, et générons donc celle-ci de manière aléatoire. Notons que, souvent, les articles relatifs aux méthodes de même type citées précédemment ne détaillent pas la manière dont les solutions initiales sont générées, ce qui rend difficile l'évaluation de l'impact de cette étape sur les performances des algorithmes.

Toutefois, une génération aléatoire présente le risque d'obtention d'une solution non faisable. En outre, la non faisabilité peut avoir plusieurs origines : non respect des contraintes de précédence, non respect de la borne supérieure des durées opératoires, ... Dans le premier cas, il est en effet possible qu'un blocage apparaisse entre différentes ressources. L'exemple de la figure 4.2 illustre ce type de situation de blocage entre deux ressources de transport R_1 et R_2. Les séquences des tâches sur ces ressources sont représentées par des noeuds de transport liés par des arcs horizontaux :
$R_1 : TP_{12} \rightarrow TP_{31} \rightarrow TP_{21}$
$R_2 : TP_{32} \rightarrow TP_{11} \rightarrow TP_{22}$
Les arcs entre ces deux séquences représentent les contraintes de précédence (dues aux gammes). On constate l'apparition d'un cycle sur ce schéma correspondant à la succession de noeuds : $TP_{12} \rightarrow TP_{31} \rightarrow TP_{32} \rightarrow TP_{11} \rightarrow TP_{12}$. Ce cycle induit les deux conjonctions contradictoires : $TP_{32} \rightarrow TP_{31}$ et $TP_{31} \rightarrow TP_{32}$. Il en va de même avec TP_{11} et TP_{12}.

Pour éviter de générer ce type de solution non faisable, nous appliquons une heuristique de réparation, constituée des étapes suivantes, et détaillée dans l'algorithme 2 :

- générer aléatoirement une séquence qui contient toutes les tâches de traitement (exemple : première ligne de la figure 4.3) ;
- réparer cette séquence jusqu'à ce qu'il n'y ait plus de contrainte de précédence violée. La réparation consiste à identifier tout couple d'opérations d'un même *job* ne respectant pas la contrainte de gamme de ce dernier, puis d'échanger la position des deux opérations du couple dans la séquence courante. Cette procédure est répétée tant qu'il reste des couples dans ce cas ;
- déduire l'ordonnancement associé à toutes les ressources de traitement et de transport.

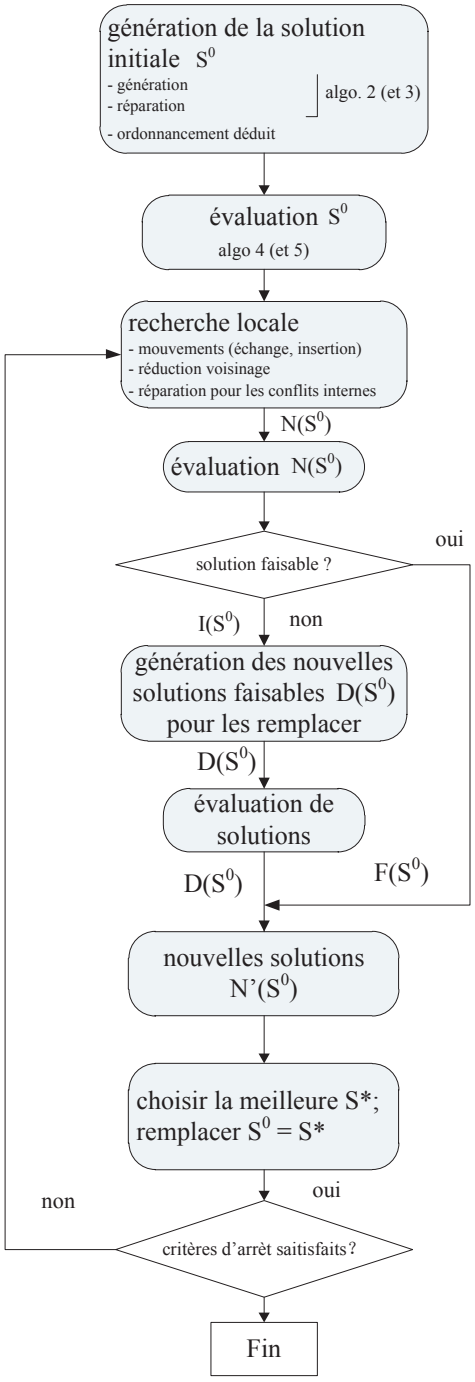

FIGURE 4.1 – Procédure Tabou

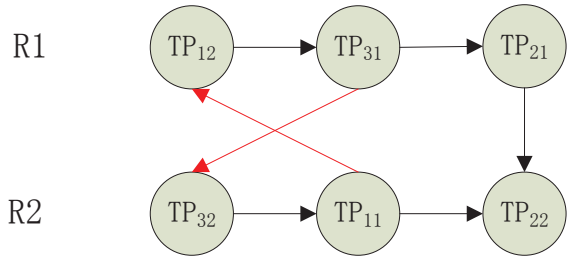

FIGURE 4.2 – Une séquence de transports infaisable.

Algorithm 2 Génération d'une solution faisable

générer aléatoirement une séquence *seq* qui contient toutes les tâches de traitement ;
réparer cette séquence en échangeant les paires de tâches qui violent la contrainte de précédence (algorithme 3) ;
//de l'affectation des opérations, déduire la séquence des tâches pour chaque ressource de traitement :
for $i \leftarrow 1$ **to** O **do**
  trouver l'opération OP_{jl} qui est la $i$ème opération dans la séquence *seqo*
  trouver la ressource M_k qui traite l'opération OP_{jl} $PJ_{jlk} = 1$;
  ajouter l'opération OP_{jl} à la fin de la séquence d'opérations de la ressource M_k ;
end for
//déduire la séquence de transport seqt qui contient toutes les tâches de transport :
for $i \leftarrow 1$ to O **do**
  trouver l'opération OP_{jl} qui est la $i$ème opération dans la séquence *seqo*, $OP_{jl} \leftarrow seqo[i]$;
  if OP_{jl} n'est pas la dernière tâche de son travail **then**
    trouver la tâche de transport TP_{jl} qui suit OP_{jl} ;
    ajouter TP_{jl} à la fin de la séquence de transport *seqt* ;
  end if
end for
//de l'affectation des déplacements en charge, déduire la séquence sur toutes les ressources de transport :
for $i \leftarrow 1$ **to** $O - n$ **do**
  trouver l'opération TP_{jl} qui est la $i$ème opération dans la séquence *seqt*, $TP_{jl} \leftarrow seqt[i]$;
  trouver la ressource R_h qui traite l'opération TP_{jl} $TJ_{jlh} = 1$;
  ajouter TP_{jl} à la fin de la séquence d'opérations de la ressource R_h ;
end for
return la solution.

Algorithm 3 Réparation d'une séquence *seq*.

```
for i ← 1 to size(seq) − 1 do
  for j ← i + 1 to size(seq) do
    if (deux tâches seq[i] et seq[j] sont dans le même travail) et (seq[j] doit être traité
    avant) then
      échanger seq[i] et seq[j] ;
    end if
  end for
end for
return seq
```

Pour illustrer cette heuristique, nous présentons un exemple simple composé de trois travaux $J = \{J_1, J_2, J_3\}$, quatre machines $M = \{M_1, M_2, M_3, M_4\}$ et deux ressources de transport R_1, R_2. Les informations sur chaque travail sont :
J1 : $OP_{11}\{M_1, M_2\} \rightarrow TP_{11}\{R_2\} \rightarrow OP_{12}\{M_3, M_4\} \rightarrow TP_{12}\{R_1\} \rightarrow OP_{13}\{M_2, M_3\}$
J2 : $OP_{21}\{M1, M2\} \rightarrow TP_{21}\{R_1\} \rightarrow OP_{22}\{M2, M3\} \rightarrow TP_{22}\{R_2\} \rightarrow OP_{23}\{M3, M4\}$
J3 : $OP_{31}\{M2, M4\} \rightarrow TP_{31}\{R_1\} \rightarrow OP_{32}\{M1, M3\} \rightarrow TP_{32}\{R_2\} \rightarrow OP_{33}\{M1, M4\}$
Dans ce qui précède, le formalisme $OP_{ij}\{M_u, M_v\}$ signifie que $\{M_u, M_v\}$ correspond à MP_{ij}, c'est à dire à l'ensemble des machines utilisables pour réaliser l'opération OP_{ij}. Ainsi, soit M_1, soit M_2 sera affecté à OP_{11}. Ici, les problèmes de séquencement sont étudiés en considérant que l'affectation est fixée a priori, ce qui veut dire qu'une ressource de traitement/de transport unique est attribuée au préalable à chaque tâche opératoire/de transport. Une affectation de ressource pour chaque tâche de traitement est par exemple :
J1 : $OP_{11}(M_1) \rightarrow TP_{11} \rightarrow OP_{12}(M_3) \rightarrow TP_{12} \rightarrow OP_{13}(M_2)$
J2 : $OP_{21}(M1) \rightarrow TP_{21} \rightarrow OP_{22}(M2) \rightarrow TP_{22} \rightarrow OP_{23}(M4)$
J3 : $OP_{31}(M2) \rightarrow TP_{31} \rightarrow OP_{32}(M3) \rightarrow TP_{32} \rightarrow OP_{33}(M4)$
D'après les données de cet exemple, il n'y a qu'un choix d'affectation possible pour chaque tâche de transport.

Une séquence opératoire totale est donc générée aléatoirement (première ligne de la figure 4.3). Elle contient toutes les tâches OP_{ij} à réaliser. Cette séquence initiale peut contenir plusieurs conflits liés aux contraintes de précédence, comme c'est le cas pour notre exemple : $OP_{32} \rightarrow OP_{31}$, $OP_{33} \rightarrow OP_{31}$, $OP_{23} \rightarrow OP_{22}$, $OP_{12} \rightarrow OP_{11}$. La figure 4.3 montre les échanges successivement réalisés avec notre procédure de réparation. Nous obtenons une séquence opératoire totale qui satisfait toutes les contraintes de gamme.

Basée sur cette séquence réparée, une séquence de transport est déduite en appliquant la règle suivante : la tâche de transport TP_{ij} qui a la priorité la plus grande est celle qui fait suite à la tâche de traitement OP_{ij} se terminant le plus tôt. Pour notre exemple, la séquence de transport déduite est représentée par la figure 4.4. En combinant l'affectation initiale et ces deux séquences (traitement + transport), nous avons pu obtenir la séquence des tâches sur chaque ressource. La figure 4.5 illustre la solution obtenue. Elle représente la solution suivante :

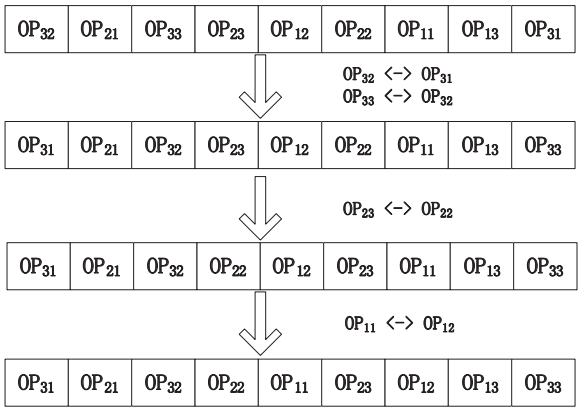

FIGURE 4.3 – Réparation d'une séquence de tâches de traitement

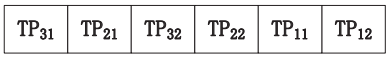

FIGURE 4.4 – La séquence de transport déduite à partir de la séquence opératoire associée

M1 : $OP_{21} \rightarrow OP_{11}$
M2 : $OP_{31} \rightarrow OP_{22} \rightarrow OP_{13}$
M3 : $OP_{32} \rightarrow OP_{12}$
M4 : $OP_{23} \rightarrow OP_{33}$
R1 : $TP_{31} \rightarrow TP_{21} \rightarrow TP_{12}$
R2 : $TP_{32} \rightarrow TP_{22} \rightarrow TP_{11}$.
On voit qu'il n'y a pas de blocage dans ces séquences, ce qui signifie qu'il s'agit d'une solution faisable du point de vue des contraintes. Cependant cela peut s'avérer insuffisant pour que la solution soit faisable par rapport à l'ensemble des contraintes, en particulier celles relatives aux durées de traitement maximales. Ce problème est géré dans la phase d'évaluation décrite ci-après.

4.2.3 Évaluation d'une solution

Comme les temps de traitement sont limités par des valeurs minimales et maximales, nous calculons les intervalles de temps compris entre les dates de début au plus tôt et au plus tard des tâches de traitement et de transport. Selon l'affectation et les séquences des ressources, les intervalles de temps peuvent être réduits. Par défaut, toutes les fenêtres de temps sont initialisées à $[0, d]$, où d représente la date de fin au plus tard de l'ensemble des *jobs*. Les dates de début au plus tôt et au plus tard sont ensuite mises à jour itérativement, en tenant compte de l'ordonnancement des ressources, tant qu'il reste des mises à jour possibles. Néanmoins, des solutions infaisables peuvent encore subsister. Pour ces cas, la mise à jour des intervalles de temps ne s'arrêtera jamais, car la procédure risque

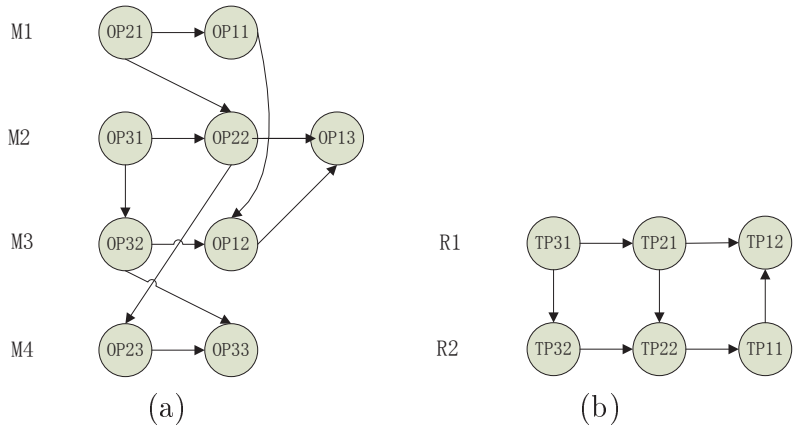

FIGURE 4.5 – (a) Séquencement des ressources de traitement ; (b) Séquencement des ressources de transport ;

de boucler indéfiniment. Nous définissons alors un certain nombre d'itérations maximum. Quand il est atteint, la mise à jour est arrêtée, et la faisabilité de cette solution est vérifiée en contrôlant tous les intervalles de temps. Ensuite, la solution est évaluée en se basant sur ces intervalles de temps. La date de l'opération la plus tardive est fixée à sa date au plus tôt. Les fenêtres de temps des autres tâches sont à nouveau réduites sur cette base. Le *makespan* et le stockage sont ainsi déterminés, selon les équations 4.1 et 4.3 rappelées ici :

$$C_{max} = max_{i\in[1,n]}\{c_i\} \tag{4.1}$$

où

$$c_i = p_{iO_i} + P_{iO_i} \tag{4.2}$$

$$Stor = \sum_{k\in[1,m]} W_k^- + W_k^+ \tag{4.3}$$

où $W_k^- = \sum_{i=1..n,j=2..O_i} US_{ij} \times PJ_{ijk}$, $W_k^+ = \sum_{i=1..n,j=1..O_i-1} DS_{ij} \times PJ_{ijk}$, $US_{ij} = p_{ij} - t_{ij-1} - T_{ij-1}$, et $DS_{ij} = t_{ij} - p_{ij} - P_{ij}$.

La procédure d'évaluation est décrite dans l'algorithme 4.

L'évaluation conjointe du *makespan* et du stockage a plusieurs avantages : elle traite les problèmes avec des temps de traitement bornés, alors que les méthodes classiques pour *job shop* ne peuvent pas. Elle nous permet de mieux comparer les solutions avec le même *makespan*, mais avec des besoins de stockage différents. Les intervalles de temps obtenus depuis la mise à jour précédente permettent de représenter plusieurs solutions possibles (en termes de dates de début au plus tôt pour les tâches de traitement et de transport, ainsi qu'en termes de stockage) pour la même valeur du makespan C_{max}. Notons

Algorithm 4 Évaluation d'une solution (calcul des intervalles de dates)

//Étape 1 :Initialiser les fenêtres de temps basé sur les séquences de traitement
for $i \leftarrow 1$ **to** n **do**
  //Initialiser les dates de début au plus tôt
  for $j \leftarrow 1$ **to** $O_i - 1$ **do**
    if $p^-_{ij} + P^-_{ij} > t^-_{ij}$ **then**
      $t^-_{ij} \leftarrow p^-_{ij} + P^-_{ij}$
    end if
    if $t^-_{ij} + \tau_{kl} > p^-_{ij+1}$ **then**
      $p^-_{ij+1} \leftarrow t^-_{ij} + \tau_{kl}$
    end if
  end for
  //Initialiser les dates de début au plus tard
  for $j \leftarrow O_i$ **to** 2 **do**
    if $p^+_{ij} - \tau_{kl} < t^+_{ij-1}$ **then**
      $t^+_{ij-1} = p^+_{ij} - \tau_{kl}$
    end if
    if $t^+_{ij-1} - P^-_{ij-1} < p^+_{ij-1}$ **then**
      $p^+_{ij-1} = t^+_{ij-1} - P^-_{ij-1}$
    end if
  end for
end for
//Étape 2 : mise à jour des dates
count $\leftarrow 0$
repeat
  Mettre à jour les dates en se basant sur les séquences des ressources (algorithme 5)
  count $++$
until Il n'y a plus de nouvelle mise à jour ou *count* $>$ un nombre défini
Calculer deux objectifs définis dans Eq. 4.1 et Eq. 4.3
for $i \leftarrow 1$**to**n **do**
  $p^+_{iO_i} \leftarrow C_max - P^-_{iO_i}$
  count $\leftarrow 0$
  repeat
    Mettre à jour les dates au plus tard en se basant sur les séquences des ressources (algorithme 5)
    count $++$
  until Il n'y a plus de nouvelle mise à jour ou *count* $>$ un nombre défini
end for

Algorithm 5 Mise à jour des dates en se basant sur les séquences des ressources

for $k \leftarrow 1$ **to** m **do**
  for $i \leftarrow 1$ **to** $size(M_k) - 1$ **do**
    //ième opération $OP_{i_1j_1}$ et son opération successive $OP_{i_2j_2}$ sur M_k, $i_1 \neq i_2$
    $OP_{i_1j_1} \leftarrow M_k[i]$, $OP_{i_2j_2} \leftarrow M_k[i+1]$
    if $p^-_{i_1j_1} + P^-_{i_1j_1} > p^-_{i_2j_2}$ **then**
      $p^-_{i_2j_2} \leftarrow p^-_{i_1j_1} + P^-_{i_1j_1}$
      mettre à jour les dates de début au plus tôt pour chaque tâche cc du même travail J_{i_2}, $j_2 < j < O_{i_2}$
    end if
  end for
  for $i \leftarrow size(M_k)$ **to** 2 **do**
    //ième opération $OP_{i_2j_2}$ et son opération précédente $OP_{i_1j_1}$ sur M_k
    $OP_{i_2j_2} \leftarrow M_k[i]$, $OP_{i_1j_1} \leftarrow M_k[i-1]$
    if $p^+_{i_2j_2} - P^-_{i_1j_1} < p^+_{i_1j_1}$ **then**
      $p^+_{i_1j_1} \leftarrow p^+_{i_2j_2} - P^-_{i_1j_1}$
      mettre à jour les dates de début au plus tard pour chaque tâche OP_{i_1j} du même travail J_{i_1}, $0 < j < j_1$
    end if
  end for
end for
for $h \leftarrow 1$ **to** r **do**
  for $i \leftarrow 1$ **to** $size(R_h) - 1$ **do**
    //ième tâche de transport $TP_{i_1j_1}$ et sa tâche successive sur R_h, $i_1 \neq i_2$
    $TP_{i_1j_1} \leftarrow R_h[i]$, $TP_{i_2j_2} \leftarrow R_h[i+1]$
    if $t^-_{i_1j_1} + \tau_{k_1l_1} + \sigma_{l_1k_2} > t^-_{i_2j_2}$ **then**
      $t^-_{i_2j_2} \leftarrow t^-_{i_1j_1} + \tau_{k_1l_1} + \sigma_{l_1k_2}$
      mettre à jour les dates de début au plus tôt pour chaque tâche TP_{i_2j} du même travail J_{i_2}, $j_2 < j < O_{i_2} - 1$
    end if
  end for
  for $i \leftarrow size(R_h)$ **to** 2 **do**
    //ième tâche $TP_{i_2j_2}$ et sa tâche précédente $TP_{i_1j_1}$ sur R_h
    $TP_{i_2j_2} \leftarrow R_h[i]$, $TP_{i_1j_1} \leftarrow R_h[i-1]$
    if $t^+_{i_2j_2} - \sigma_{l_1k_2} - \tau_{k_1l_1} < t^+_{i_1j_1}$ **then**
      $t^+_{i_1j_1} \leftarrow t^+_{i_2j_2} - \sigma_{l_1k_2} - \tau_{k_1l_1}$
      mettre à jour les dates de début au plus tard pour chaque tâche TP_{i_1j} du même travail, $0 < j < j_1$
    end if
  end for
end for

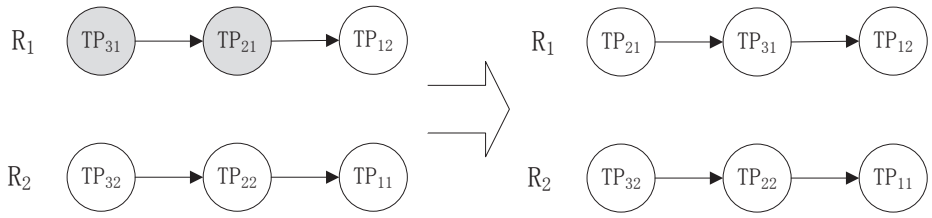

FIGURE 4.6 – Échange interne

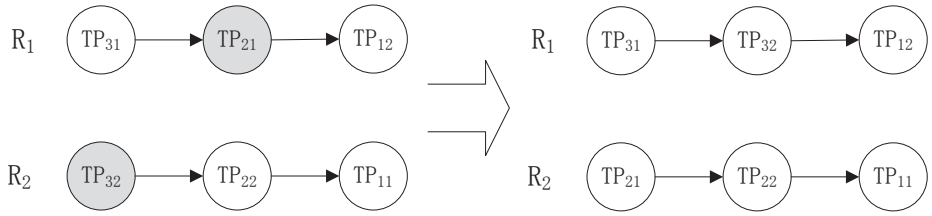

FIGURE 4.7 – Échange externe

que le stockage calculé dans chaque *buffer* est en fait une borne minimale du temps de stockage. Pour les systèmes où cela est strictement limité ou ceux où aucun moyen de stockage n'est autorisé (par exemple dans l'industrie chimique et l'industrie alimentaire), si le stockage minimum obtenu pour une solution est supérieur à zéro, ce n'est sûrement pas une solution faisable, en termes de qualité. Toutefois, les solutions faisables pour ces systèmes sont incluses parmi toutes les solutions avec une évaluation du stockage égale à zéro. Une autre procédure est proposée dans le chapitre suivant (voir l'algorithme 12), pour vérifier l'existence réelle de stockage dans une solution.

4.2.4 Construction et réduction du système de voisinage

Le système de voisinage que nous utilisons est composé de trois mouvements basiques :

- échange interne : échange de deux tâches traitées sur la même ressource de transport (voir Figure 4.6).
- échange externe : échange de deux tâches de transport affectées à des ressources différentes (voir Figure 4.7).
- insertion : suppression d'une tâche depuis sa ressource de transport prédéfinie, puis affectation à une autre ressource ; on peut aussi insérer cette tâche à une autre position de la séquence sur la même ressource (voir Figure 4.8).

Un mouvement d'échange interne modifie uniquement la séquence d'une seule ressource. La performance de ce type d'échange dépend donc strictement de l'affectation initiale de la solution concernée. L'échange externe et le mouvement d'insertion peuvent changer l'affectation des ressources de la solution initiale en échangeant deux tâches effectuées par différentes ressources ou en déplaçant une tâche vers une autre ressource possible. Notons

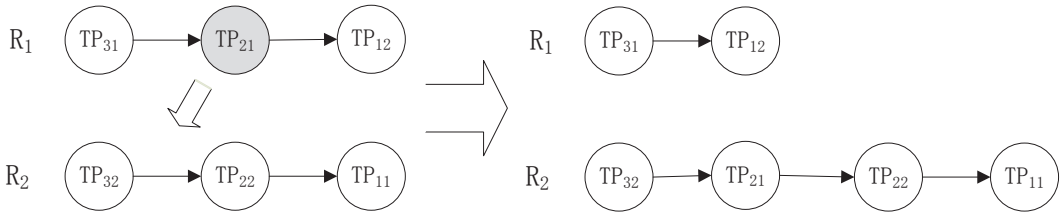

FIGURE 4.8 – Insertion

que nous avons appliqué ces mouvements aux ressources de transport. Mais indirectement, les séquencements des ressources de traitement peuvent aussi être modifiées, selon les nouvelles séquences de tâches de transport associées.

Le principe de base de la méthode de voisinage consiste à générer tous les voisins à partir des trois mouvements définis, puis de les évaluer pour retenir le meilleur voisin. Or, avec le premier et le second mouvements, au maximum $(O-n)(O-n-1)/2$ voisins sont générés pour les ressources de transport. Le troisième mouvement peut donner au maximum $(O-n) \times (O-n+r-2)$ voisins pour les ressources de transport. Suivant la taille du problème traité, il est alors souvent impossible d'explorer tous les voisins. De plus, parmi eux, seulement quelques-uns sont utiles pour améliorer la solution actuelle. Aussi nous avons ajouté différents critères de réduction destinés à limiter le nombre de voisins explorés dans l'espace de recherche. Pour garder les voisins potentiellement intéressants, trois types de critères de réduction sont appliqués en fonction des trois mouvements :

- pour l'échange interne, le critère de réduction est inspiré de Deroussi et al. ([Deroussi et al., 2008]) : une tâche de transport TP_{ij} quelconque est échangée avec une tâche parmi celles traitées dans l'intervalle $[p_{ij}+P_{ij}, p_{ij+1}]$ (entre la date de fin de OP_{ij} et le début de OP_{ij+1}). L'objectif est de réduire le *makespan* pour des solutions avec stockage. Mais ce n'est pas forcément très efficace pour des solutions avec peu ou pas d'attente aux places de stockage. Dans notre cas, compte tenu des contraintes de durées opératoires bornées, nous remplaçons l'intervalle de temps précédent par $[t^-_{ij}, p^-_{ij+1}]$.
- l'échange externe est appliqué à des couples de tâches dont les intervalles de dates d'exécution ont une intersection non vide. Ce critère de réduction limite le risque d'obtention de séquences non faisables.
- le troisième mouvement affecte une tâche TP_{ij} à une autre ressource de transport disponible au cours de l'intervalle $[p^-_{ij}+P^-_{ij}, t^-_{ij}]$. Cette ré-affectation a pour but d'anticiper la réalisation de cette tâche, si cela est possible. Cela est utile dans le cas où une ressource est surchargée par rapport aux autres. Ce critère contribue à équilibrer la charge sur l'ensemble des ressources de transport.

Malgré le voisinage défini et les critères de réduction adoptés, les trois mouvements de recherche locale peuvent conduire à la génération de solutions présentant des conflits. Ceux-ci peuvent être de deux types :

- conflits internes : ils sont dûs aux contraintes de précédence non respectées au niveau d'une même ressource ;
- conflits externes : ils proviennent du non respect de contraintes de précédence sur des ressources différentes (par exemple, le blocage de la figure 4.2).

Dans le cas de conflits internes, on répare la séquence jusquà ce qu"il n'y ait plus de violation de contrainte de précédence. Si la solution est toujours non faisable après cette réparation, elle sera remplacée par une autre solution faisable générée par l'algorithme 2.

4.3 Procédure *Shifting Bottleneck* améliorée

4.3.1 Description de la procédure *Shifting Bottleneck* (SBN) initiale

The Shifting Bottleneck procedure est une procédure souvent utilisée pour minimiser le *makespan* dans un atelier de type *job shop* pour lequel les ressources de traitement sont critiques. Cette heuristique est destinée à minimiser l'effet de goulet d'étranglement (*bottleneck*) en ordonnançant d'abord les ressources qui causent potentiellement le plus de retard. Elle a été proposée par [Adams et al., 1988] en 1988.

La procédure standard commence avec deux ensembles : l'ensemble des ressources de traitement qui sont déjà séquencées MS, et l'ensemble total des ressources de traitement M. A l'étape initiale, l'ensemble MS est vide. Un graphe disjonctif du problème associé est construit. A chaque itération, une ressource est choisie et ajoutée dans MS. Les opérations sur cette ressource sont séquencées et le graphe est mis à jour en intégrant l'arbitrage des disjonctions correspondantes sous forme d'arcs conjonctifs.

Le principe de la méthode consiste, à chaque itération, à déterminer la ressource qui cause le plus de blocage dans le graphe courant. Pour cela, il faut résoudre un problème à une machine de type $1|r_j|L_{max}$ pour chaque ressource M_j ($M_j \in M - MS$). Rappelons que selon la notation de Graham et al. ([Graham et al., 1979]), $1|r_j|L_{max}$ fait référence à un problème d'ordonnancement à une machine où les tâches sont soumises à des contraintes de dates de début au plus tôt r_j et où le critère à minimiser est le retard maximal L_{max}. Pour chaque opération de traitement, les dates de début au plus tôt (*release dates*) sont déterminées par le plus long chemin depuis le nœud de départ jusqu'au nœud opératoire associé. Les dates de fin au plus tard (*due dates*) sont calculées à partir du C_{max} correspondant au plus long chemin dans ce graphe. Ces dates permettent de définir les données des problèmes à une machine à résoudre à chaque itération. Pour la résolution, on utilise une procédure de type *branch and bound* décrite à l'annexe B, pour trouver le plus petit retard L_{max} pour chaque sous-problème à une machine associé à chaque ressource de traitement. Parmi les ressources non ordonnancées, la ressource M_k qui a la plus grande valeur de L_{max} sera choisie comme ressource goulet et les arcs conjonctifs associés à la séquence de cette ressource seront ajoutées au graphe partiel G. M_k est donc ajoutée dans

MS. De ce fait, le *makespan* courant est augmenté de L_{max}. L'étape suivante consiste à réordonnancer toutes les ressources de l'ensemble $MS - M_k$. La procédure est ensuite réitérée à partir du nouveau graphe tant qu'il reste des ressources à ordonnancer.

4.3.2 Une procédure *Shifting Bottleneck* modifiée

Nous proposons une nouvelle version modifiée pour le problème générique considéré (GFJSSP). Cette version est une extension de la procédure de [Rossé-Bloch, 1999], qui est déjà une première adaptation de la procédure initiale pour le *Hoist Scheduling Problem*. Elle utilise le graphe générique que nous avons présenté au chapitre précédent. Les adaptations effectuées ont pour objectif la prise en compte de l'ensemble des contraintes (temps de transport en charge et à vide, possibilité de stockage, temps opératoires bornés). Cette procédure modifiée est combinée avec une heuristique qui cherche les séquences faisables sur les ressources de transport. La figure 4.9 décrit cette procédure globale. L'algorithme 6 détaille cette procédure.

Condition initiale

Au départ, les ressources de traitement déjà ordonnancées sont enregistrées dans un ensemble MS. Cet ensemble est initialement vide. Notons que la procédure peut démarrer d'une situation initiale où les tâches sur certaines ressources sont déjà ordonnancées. Dans ce cas, $MS \neq \emptyset$. Par exemple, pour le HSP, tous les produits passent par le poste de chargement pour leur première opération. Donc l'ordre d'arrivée sur ce poste, s'il est connu, peut être considéré comme séquence d'entrée a priori. En fait, cet ordre d'entrée est souvent généré aléatoirement. Un graphe disjonctif générique G est construit. Il contient les arcs conjonctifs associés aux contraintes de gamme, et aux disjonctions arbitrées liées aux ressources de MS. Les arcs disjonctifs du graphe sont associés aux ressources dans $(M - MS) \cup R$ non encore séquencées (R étant l'ensemble des robots).

Le *makespan* courant C_{max} est égal au plus long chemin dans le graphe partiel G (graphe courant sans les arcs disjonctifs). L'algorithme de Dantzig (voir l'annexe B) est utilisé pour chercher le plus long chemin dans ce graphe. Il est aussi utilisé pour détecter l'existence de circuit de longueur positive qui signifie la non faisabilité du séquencement. Les dates de début au plus tôt de chaque tâche (opératoire, de transport et de stockage) sont déterminées par la longueur du plus long chemin depuis le nœud de départ jusqu'à chaque nœud associé. La date de fin au plus tard pour une tâche (OP_{ij}) est égale à la différence entre C_{max} et la valeur du plus long chemin entre la tâche considérée et le dernier noeud.

Procédure modifiée avec réparation

Cette procédure consiste à séquencer les tâches en appliquant un SBN, d'abord sur toutes les ressources de traitement, puis sur les ressources de transport. Elle est inspirée

Algorithm 6 Procédure SBN modifiée pour un graphe contenant des arcs négatifs.

//Recherche des séquences faisables sur les ressources de traitement
//Étape 1 : condition initiale
MS contient les ressources qui sont déjà séquencées.
Construction du graphe simple G qui contient tous les arcs de précédence et les arcs conjonctifs correspondant aux arbitrages des tâches sur chaque ressource dans MS *//en cas de circuit positif, l'algorithme de Dantzig retourne la valeur -1*
if Dantzig(G) $= -1$ **then**
  return -1 ;
end if
//Etape2. (Analyse des machines qui ne sont pas encore ordonnancées)
for chaque machine $M_i \in (M - MS)$ **do**
  minimiser L_{max} en utilisant la règle EDD pour une instance $1/r_j/Lmax$. $Lmax(i)$ est le $_{Lmax}$ minimal trouvé pour le sous-problème associé à M_i.
end for
//Etape3. (Sélection de la ressource goulet et de la séquence)
$L_{max}(k) \leftarrow max_{i/M_i \in M-MS} L_{max}(i)$
ajouter les arcs conjonctifs correspondant à la séquence sur M_k dans le graphe G.
if Dantzig(G) $= -1$ **then**
  enlever les arcs conjonctifs associés à la séquence de M_k
  Réparer la séquence de M_k (voir l'algorithme 7)
  if réparation non réussie **then**
    return -1
  else
    ajouter les arcs associés à la séquence réparée.
    $MS \leftarrow MS \bigcup M_k$
  end if
end if
//Etape4. (critère d'arrêt)
if $MS \neq M$ **then**
  goto Etape 2.
end if
Ordonnancer les ressources de transport (voir l'algorithme 9)
if ordonnancement réussi **then**
  return la solution
else
  return -1.
end if

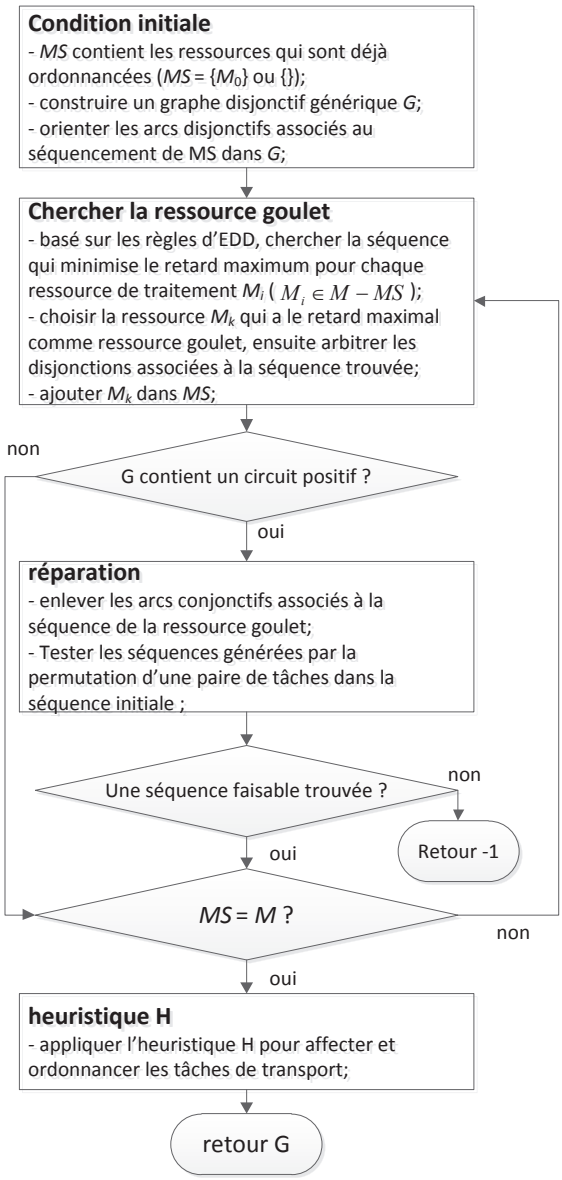

FIGURE 4.9 – Procédure SBN modifiée combinée avec l'heuristique H

de celle de [Rossé-Bloch, 1999], mais adaptée à notre graphe générique. A chaque itération, la ressource de traitement goulet est déterminée, et la séquence associée est ajoutée dans le graphe courant. Toutefois, à cause des valeurs négatives correspondant aux temps opératoires maximaux, la probabilité de générer une solution non faisable par la procédure SBN est plus grande. Pour limiter le risque de blocage, Bloch a ajouté une procédure de réparation. Elle consiste à modifier l'arbitrage d'une disjonction pour une paire de tâches associé à un circuit positif. Cette paire n'est pas connue a priori. On teste donc toutes les permutations des paires de tâches à partir de la séquence courante de la ressource

Algorithm 7 Réparation d'une séquence Seq d'une ressource de traitement dans un graphe G

for toutes les permutations de deux tâches de Seq qui ne sont pas dans le même travail **do**
  échanger cette paire pour obtenir une nouvelle séquence Seq'
  arbitrer les disjonctions associées à la séquence Seq' dans G
  if Dantzig(G) $\neq -1$ **then**
    return (Seq', G)
  end if
end for
return -1

goulet. On s'arrête dès qu'on obtient une paire dont la permutation permet d'éliminer le circuit positif (vérification de la nouvelle séquence par l'algorithme de Dantzig). Sinon, l'algorithme retourne la valeur -1. L'efficacité de cette réparation dépend notamment du nombre de tâches sur la ressource de traitement considérée. Plus ce nombre est grand, plus il est difficile de réparer la séquence obtenue par la procédure SBN avec un seul changement. Nous appliquons cette procédure au graphe générique pour rechercher les séquences faisables sur les ressources de traitement. La procédure continue jusqu'à ce que toutes les ressources de traitement soient ordonnancées.
Comme nous l'avons déjà expliqué au chapitre précédent, la version modifiée de Bloch avec réparation s'applique sous l'hypothèse qu'il y a suffisamment de ressources de transport pour l'ensemble des déplacement à réaliser. L'arbitrage des disjonctions sur les ressources de transport n'est donc pas un problème dans ce cas. Le fait de transposer cette méthode à notre problème générique et au graphe générique associé nous impose désormais de gérer les conflits de ressources de manutention puisque nous considérons que celles-ci sont des ressources disjonctives présentes en quantité limitée. Le tableau 4.1 compare le nombre de solutions obtenues par le SBN et le graphe simple de Bloch, avec notre SBN modifié utilisant notre graphe générique. Pour ce test, nous sommes partis d'un graphe générique intégrant les séquences trouvées par le graphe simple pour les ressources de traitement. Puis nous avons utilisé notre procédure SBN pour l'ensemble des ressources de transport. La réparation a été appliquée aux tâches de traitement et de transport. Les instances testées correspondent à 6 ateliers de traitement de surface générés par Mateo et Companys (2002), pour lesquels nous avons considéré 2 robots (nous avons utilisé ici les premières instances de chaque cas présentés en annexe F). Ces instances contiennent entre 5 et 10 machines (cuves ici), non compris les postes de chargement et déchargement, et 2 types de *jobs* y sont traités. Les *jobs* ont la même séquence de traitement (de 7 à 12 opérations par gamme, y compris les opérations de chargement et déchargement), mais avec différentes fenêtres temporelles sur les durées opératoires. Nous avons résolu des problèmes sans attente, sans stock et avec durées bornées pour 4, 6 et 10 *jobs*. Enfin, nous avons généré aléatoirement 200 000 affectations initiales pour les opérations de transport,

TABLE 4.1 – Comparaison des solutions obtenues avec le graphe simple de Bloch et le graphe générique

nb. de machines	nb. de jobs	nb. moyen de solutions obtenues avec graphe simple	nb. moyen de solutions obtenues avec graphe générique	rapport simple/générique
5	4	200000	28929	6,91
	6	144882	1884	76,9
	10	1145	1	1145
6	4	159807	14481	11,03
	6	50252	20937	2,4
	10	255	6	42,5
7	4	200000	182447	1,09
	6	169996	74637	2,27
	10	147188	2369	62,13
8	4	32982	29818	1,1
	6	9565	4081	2,34
	10	918	9	102
9	4	65563	18100	3,62
	6	9539	4	2384,75
10	4	65323	16298	4
	6	23430	13	1802

les opérations de traitement étant déjà préalablement affectées, les cuves étant toutes de capacité unitaire (mono-bac).

Les résultats sont des moyennes obtenues sur la base de dix exécutions pour chaque instance. L'interprétation du tableau est la suivante : par exemple, si on examine la deuxième ligne pour l'instance numéro 1 avec 6 jobs, on obtient un nombre moyen de 144882 séquencements possibles des ressources de traitement, sur 200000 tests effectués. Si on prend en compte les contraintes de transport, seulement 1884 séquences sont réalisables, soit 100 fois moins. Ce deuxième résultat est plus limitatif en nombres de solutions, mais il reflète mieux la réalité, et n'est pas surprenant dans le cas de problèmes de type HSP, correspondant à ce type de problème, qui sont très contraints, et où le transport est souvent l'activité critique. La dernière colonne du tableau 4.1 montre le rapport entre le nombre de solutions trouvées par les deux graphes (simple et générique). Ce rapport ne varie pas toujours de manière proportionnelle, ni par rapport au nombre de jobs, ni par rapport au nombre de machines. Ceal peut s'expliquer par le fait que la faisabilité dépend aussi fortement de la largeur des fenêtres temporelles des durées opératoires. Toutefois, on peut constater que, lorsque la taille du problème augmente, la valeur des rapports est malgré tout plus importante. Cela rend plus difficile la recherche d'un petit nombre de solutions faisables dans un espace de recherche très grand.

Le tableau 4.1 montre aussi que cette version de SBN étendue au graphe générique

est capable de résoudre des instances de HSP de petite taille. Mais quand cette taille augmente, il y a plus de tâches à ordonnancer par ressource, et la méthode actuelle ne parvient pas à trouver de solution faisable (par exemple en troisième ligne de résultat pour 5 cuves et 10 *jobs*). Les raisons sont :

- la réparation n'est faite que par le biais d'un seul échange d'une paire de tâches. Quand le nombre de tâches est grand, cet échange n'est souvent pas suffisant.
- le fait de séparer le séquencement des machines et des ressources de transport simplifie a priori le calcul. Mais cela limite également beaucoup les solutions possibles pour les moyens de manutention qui peuvent être les ressources critiques de l'atelier.

Compte tenu de ces observations, nous nous sommes intéressés à la façon d'intégrer l'arbitrage des disjonctions liées au transport au sein de notre procédure SBN modifiée. Dans la suite, nous présentons et comparons deux manières de gérer les conflits de transport.

Heuristiques dédiées au transport

Dans un premier temps, et compte tenu des réflexions précédentes, nous avons réfléchi exclusivement sur le problème de séquencement des tâches. Dans ce contexte, nous avons examiné diverses manières de compléter la procédure SBN, sachant que le verrou le plus fort provient principalement de la combinaison des bornes maximales des durées opératoires et de la contrainte de non stock. Parmi les différentes possibilités envisagées, nous avons élaboré une heuristique (nommée HA), combinée avec la procédure SBN, qui prend en compte une affectation initiale de toutes les tâches, si cette affectation existe (voir algorithme 8).

Dans ce cas, les ressources de transport sont considérées au même titre que les ressources de traitement. A chaque étape, la procédure SBN détermine une ressource goulet, qui peut être une ressource de traitement ou de transport. Toutefois, si la procédure classique *branch and bound* est efficace pour le job shop avec temps fixes et stockage autorisé, elle permet beaucoup plus rarement d'obtenir des séquences faisables dans le cas d'un job shop sans stock intermédiaire, comme nous avions pu l'observer avec les tests du tableau 4.1, en considérant le transport avec le traitement. Notons que dans un cas sans stock, le fait d'avoir des temps fixes est finalement la configuration la plus pénalisante puisqu'elle ne donne aucune marge de manoeuvre. Ceci dit, ce n'est pas le cas le plus traité. Le fait d'avoir des temps bornés donne une certaine flexibilité, mais non comparable à celle fournie par l'hypothèse d'un stock infini.

La démarche adoptée est donc la suivante, en considérant simultanément les ressources de traitement et de transport : nous déterminons de manière classique la séquence pour la ressource goulet. Si cette ressource est un robot, nous utilisons une heuristique appelée HA pour déterminer une seconde séquence de tâches, et nous retenons la meilleure solution trouvée. Cela permet de pallier le problème de non faisabilité dans les cas sans stock.

L'heuristique HA est la suivante : parmi l'ensemble des disjonctions (couples de tâches) sur la ressource de transport goulet, on sélectionne celles associant la tâche qui a la plus petite date de début au plus tôt. Dans ce sous-ensemble, on choisit aléatoirement une disjonction. L'arbitrage de la disjonction consiste à commencer par la tâche de plus petite date de début au plus tôt. On ajoute l'arc conjonctif associé, et le nouveau graphe générique est évalué. Si celui-ci contient un circuit positif, on considère l'ordre inverse pour cette disjonction. Si le graphe n'est toujours pas faisable, on arrête la procédure et l'algorithme retourne la valeur -1. S'il n'y a pas de circuit positif, l'arbitrage continue jusqu'à ce qu'il ne reste plus de disjonctions relatives aux tâches de transport de la ressource goulet. La procédure continue en cherchant le prochain goulet parmi les ressources de traitement et de transport non encore séquencés.

Algorithm 8 Heuristique HA : séquencer les tâches de transport affectées à la même ressource

appliquer l'algorithme de Dantzig pour évaluer le plus long chemin depuis le nœud de départ jusqu'à chaque nœud du graphe.
repeat
  classement des tâches de transport par ordre croissant de leurs dates de début au plus tôt (une date correspond à la longueur du chemin le plus long depuis le nœud de départ jusqu'à chaque nœud)
  if il n'y a plus de disjonction **then**
    return G
  end if
  parmi les disjonctions associées à la tâche ayant la plus petite date de début plus tôt, en choisir une aléatoirement et l'arbitrer (par exemple $TP_{ij} \rightarrow TP_{i'j'}$)
  if $us^{-}_{ij+1} + \sigma_{kl} > t^{-}_{i'j'}$ $(PJ_{ij+1k} = 1, PJ_{i'j'l} = 1)$ **then**
    appliquer l'algorithme de Dantzig
    if G contient un circuit positif (Dantzig $(G) = -1$) **then**
      enlever cet arc
      ajouter l'arc inverse
      if G contient un circuit positif (Dantzig $(G) = -1$) **then**
        return -1
      end if
    end if
    mettre à jour les dates de disponibilité des ressources de transport
    mettre à jour les dates de début au plus tôt de chaque tâche
  end if
until true

La prise en compte simultanée des opérations de traitement et de transport dans la procédure SBN présente l'avantage d'identifier la ressource critique parmi l'ensemble des ressources de l'atelier (machines ou robots). Toutefois, la qualité des solutions obtenues par l'heuristique HA dépend notamment de l'affectation initiale des opérations de transport, s'il existe plusieurs ressources de transport disponibles. Dans un deuxième temps, nous

avons donc élaboré une autre heuristique (nommée H), qui ne tient pas compte d'une éventuelle affectation préalable des ressources de manutention. Elle affecte et séquence petit à petit toutes les tâches de transport en prenant en compte leur date de début au plus tôt. En cela, elle ressemble à l'heuristique HA. Mais cette fois, l'heuristique est mise en œuvre après que le séquencement des tâches de traitement ait été réalisé par la procédure SBN modifiée puis réparé si besoin.

Le principe de l'heuristique H est donc le suivant : à chaque étape, parmi toutes les tâches de transport non affectées, on choisit la tâche de transport ayant la plus petite date de début au plus tôt. Cette tâche est affectée à la ressource de transport disponible et capable de la réaliser le plus tôt. Ensuite, parmi toutes les tâches de transport qui sont déjà affectées, on arbitre itérativement les disjonctions associées. Comme dans l'heuristique HA, on commence par arbitrer la disjonction de deux tâches pour laquelle la première tâche a la plus petite date de début au plus tôt. Après chaque arbitrage, on évalue le graphe résultant. La procédure continue à affecter puis arbitrer les tâches de transport jusqu'à ce qu'elles soient toutes affectées et qu'il n'y ait plus de disjonctions. Cette procédure est détaillée dans l'algorithme 9.

On remarque que nos heuristiques ré-évaluent le graphe par l'algorithme de Dantzig chaque fois qu'une nouvelle disjonction est arbitrée. Donc la complexité de l'évaluation dépend strictement de la taille du graphe générique. Notre graphe générique contient plus de nœuds que les autres graphes disjonctifs. Pour réduire cette complexité, chaque fois qu'une disjonction est arbitrée ($TP_{ij} \rightarrow TP_{i'j'}$), l'évaluation du nouveau graphe courant n'est effectivement réalisée que si l'arc associé nécessite de mettre à jour les dates de début au plus tôt de $TP_{i'j'}$. Cette condition s'exprime par la vérification de l'inégalité :
$us^{-}_{ij+1} + \sigma_{kl} > t^{-}_{i'j'}$.

Comparaison entre les deux heuristiques H et HA

Les deux heuristiques que nous proposons séquencent progressivement les tâches de transport en prenant en compte les dates de début au plus tôt dans un graphe courant. H est d'autant plus efficace que le nombre de ressources de transport est important. Nous avons comparé nos deux heuristiques sur la base d'instances de HSP proposées par [Mateo et al., 2002], correspondant donc à des problèmes sans stock, sans attente, avec fenêtres temporelles pour les durées opératoires. Ces instances ont été générées en faisant varier la vitesse du robot et la largeur des fenêtres temporelles. Dans leur forme initiale, elles considèrent 2 types de produits et 1 robot. Nous les avons modifiées, en les étendant au cas à 2 robots. Les tests ont porté sur un *job shop* avec 5 cuves et 21 *jobs* (10 de chaque type plus un du premier type de *job*). Le tableau 4.2 montre les résultats obtenus avec ces deux heuristiques, sur la base de dix exécutions par instance. Pour l'heuristique HA, nous avons généré aléatoirement des affectations initiales des opérations de transport. Le temps de résolution est environ dix minutes, l'heuristique H est légèrement plus long que HA (10%).

Algorithm 9 Heuristique H : affectation et séquencement des tâches de transport

appliquer l'algorithme de Dantzig(G) pour évaluer le plus long chemin depuis le nœud de départ jusqu'à chaque nœud dans le graphe.
calculer les dates de disponibilité pour chaque ressource de transport.
classer les tâches de transport par ordre croissant de leur date de début au plus tôt (la longueur du plus long chemin depuis le nœud de départ jusqu'à chaque nœud)
repeat
  for chaque tâche dans l'ordre croissant **do**
    if cette tâche n'est pas encore affectée **then**
      affecter la tâche sur la ressource de transport qui permet de la commencer au plus tôt (la date de disponibilité plus le temps de transport à vide)
      mettre à jour la date de disponibilité de la ressource de transport choisie
      break ;
    end if
  end for
  pour chaque ressource de transport, classer les tâches qui y sont affectées par ordre croissant
  for chaque ressource de transport **do**
    dans sa séquence (ordre croissant) de tâches affectées, trouver la première disjonction non orientée dont la première tâche a la plus petite date de début plus tôt
  end for
  if toutes les tâches sont affectées et il n'y a plus de disjonction **then**
    return G
  else
    parmi toutes les premières disjonctions de chaque ressource de transport, arbitrer la disjonction (par exemple $TP_{ij} \rightarrow TP_{i'j'}$) dont la première tâche a la plus petite date de début plus tôt
    if $us^{-}_{ij+1} + \sigma_{kl} > t^{-}_{i'j'}$ $(PJ_{ij+1k} = 1, PJ_{i'j'l} = 1)$ **then**
      appliquer l'algorithme de Dantzig
      if G contient un circuit positif (Dantzig $(G) = -1$) **then**
        enlever cet arc
        ajouter l'arc inverse
        if G contient un circuit positif (Dantzig $(G) = -1$) **then**
          return -1
        end if
      end if
      mettre à jour les dates de disponibilité des ressources de transport
      mettre à jour les dates de début au plus tôt de chaque tâche
    end if
  end if
until true

TABLE 4.2 – Comparaison des heuristiques H et HA pour les instances de [Mateo and Companys, 2007] avec 5 cuves, 21 jobs et 2 robots

Inst.	$\frac{C_{max}}{C_{HA}}$	$\frac{C_{max}}{C_H}$	$Gap\%$	Inst.	$\frac{C_{max}}{C_{HA}}$	$\frac{C_{max}}{C_H}$	$Gap\%$
51	2951	2900	-1,73	516	4164	3742	-10,13
52	2974	2736	-8,00	517	3292	2714	-17,56
53	4134	3535	-14,49	518	3462	3004	-13,23
54	3073	2453	-20,18	519	3606	3208	-11,04
55	3614	2598	-28,11	520	3440	3040	-11,63
56	3464	3285	-5,17	521	3305	2916	-11,77
57	4381	3782	-13,67	522	4631	3946	-14,79
58	2922	2704	-7,46	523	5038	4570	-9,29
59	3267	2922	-10,56	524	2890	2757	-4,60
510	2944	2921	-0,78	525	3962	3876	-2,17
511	4410	3901	-11,54	526	4221	3551	-15,87
512	2312	2312	0,00	527	3457	2999	-13,25
513	3235	2772	-14,31	528	2756	2586	-6,17
514	3621	2718	-24,94	529	3311	3156	-4,68
515	2852	2532	-11,22	530	5295	5072	-4,21

$Gap = (C_H - C_{HA})/C_{HA}$

Les résultats en termes de C_{max} montrent des performances de l'heuristique H supérieures à celles de HA, en moyenne de 10,75% (entre 0 et 28,11% suivant les cas). L'heuristique H semble donc effectivement meilleure que HA pour les cas avec plusieurs ressources de transport et les problèmes les plus contraints (sans stock et avec durées opératoires bornées). Finalement, nous avons opté pour l'heuristique H. Dans la suite du mémoire, lorsque nous ferons référence à notre procédure SBN, cela signifiera le SBN modifié avec réparation et l'heuristique H. Cela correspond au schéma général de la figure 4.9.

4.3.3 Cas particuliers pour les problèmes sans stockage

Gammes partiellement identiques

Certaines gammes peuvent être partiellement identiques en termes d'ordre de visite des ressources de traitement. Le problème d'ordonnancement dans ce groupe de ressources est alors similaire à un problème de type *flowshop*. Par ailleurs, s'il n'y a pas de flexibilité sur les machines (une seule ressource de traitement possible par tâche), alors les produits ne peuvent pas se doubler sur cette portion de l'atelier, et les séquences sur ces ressources sont les mêmes (cas d'un ordonnancement de permutation). Ce cas se rencontre par exemple dans les ateliers de traitement de surface ou dans certaines cellules robotisées de l'industrie agro-alimentaire.
La remarque précédente est vraie dans le cas de problèmes avec contraintes de non attente

et de capacité unitaire des ressources. Elle ne l'est plus si plusieurs machines peuvent être affectées à une même tâche, et en cas de fenêtres temporelles autorisant des temps effectifs différents. Cela autorise en effet un dépassement des produits. Dans le cas plus général d'un *job shop* avec stockage illimité, cette hypothèse n'est pas vérifiée non plus.

Quand la séquence d'une ressource est fixée, la propagation des arcs liés aux gammes partielles identiques, peut se faire de deux façons différentes :

- propagation 1 : on détermine les gammes partielles identiques associées à l'ensemble des travaux. On arbitre alors les disjonctions des *jobs* associés pour la ressource concernée ;
- propagation 2 : la séquence fixée pour une ressource doit être propagée aux autres ressources, chaque fois qu'un nouvel arc est ajouté au graphe. La propagation se fait en deux étapes : propagation en arrière et propagation en avant.

TABLE 4.3 – Exemple d'atelier à 3 travaux et 4 machines.

Travail 1			Travail 2			Travail 3		
Tâche	Machine	Bornes de durée	Tâche	Machine	Bornes de durée	Tâche	Machine	Bornes de durée
1	M_1	[2, 4]	1	M_1	[1,4]	1	M_2	[3,4]
2	M_2	[4, 5]	2	M_2	[2,3]	2	M_3	[2,4]
3	M_3	[4, 6]	3	M_3	[2,5]	3	M_4	[3,5]
4	M_4	[4, 6]						

Pour illustrer la méthode de propagation 1, on prend l'exemple du tableau 4.3, correspondant à un atelier à trois *jobs* et quatre machines, sur lequel aucun stock n'est autorisé. Tout d'abord, en comparant les gammes de fabrication de ces trois travaux, on trouve ci-dessous trois gammes partielles identiques communes à certains travaux :

- gamme partielle 1 : $M_1 \rightarrow M_2$ pour J_1 et J_2
- gamme partielle 2 : $M_2 \rightarrow M_3$ pour J_1, J_2 et J_3
- gamme partielle 3 : $M_3 \rightarrow M_4$ pour J_1 et J_3

On suppose que les tâches sur la machine M_2 sont réalisées dans l'ordre : $OP_{12} \rightarrow OP_{22} \rightarrow OP_{31}$. L'ordre de passage des travaux sur la machine M_2 est : $J_1 \rightarrow J_2 \rightarrow J_3$. Or M_2 existe dans les gammes partielles 1 et 2. On propage donc les ordres associés sur les ressources M_1 et M_3 : $OP_{11} \rightarrow OP_{21}$ pour M_1, $OP_{13} \rightarrow OP_{23} \rightarrow OP_{32}$ pour M_3. La figure 4.10 montre un graphe disjonctif générique sur lequel ces ordres ont été propagés. Notons que, sur ce graphe, toutes les disjonctions n'ont pas été arbitrées (arc $OP_{14} <-> OP_{33}$). En outre, ce graphe est simplifié par souci de lisibilité : nous n'avons pas reporté les arcs disjonctifs liés aux tâches de transport.

Pour la méthode de propagation 2, au lieu de propager tout d'un coup pour tous les arcs possibles, on vérifie les arcs un par un suivant la séquence de la ressource, et pour chaque arc, on applique la propagation en avant et en arrière tant que c'est possible (donc sur les tâches en amont et en aval). Cela nous évite d'analyser les gammes à l'avance. La procédure de propagation 2 est décrite dans l'algorithme 10.

Algorithm 10 Procédure de propagation (méthode 2) des arcs quand une séquence Seq sur la ressource est fixée

```
for chaque tâche OP_ij dans Seq do
  for toutes les tâches OP_kl après OP_ij dans Seq (OP_ij → OP_kl) do
    //propagation en avant
    if j < O_i et l < O_k then
      x = min{O_i, O_j}
      q = l + 1
      for p de j + 1 à x do
        if OP_ip et OP_kq sont traitées par la même ressource then
          arbitrer la disjonction de ces deux tâches : OP_ip → OP_kq
        else
          break ;
        end if
        ++q
      end for
    end if
    //propagation en arrière
    if j > 1 et l > 1 then
      q = l − 1
      x = min(j, l)
      for p de j − 1 à j − x + 1 do
        if OP_ip et OP_kq sont traitées par la même ressource then
          arbitrer la disjonction de ces deux tâches : OP_ip → OP_kq
        else
          break ;
        end if
        −−q
      end for
    end if
  end for
end for
```

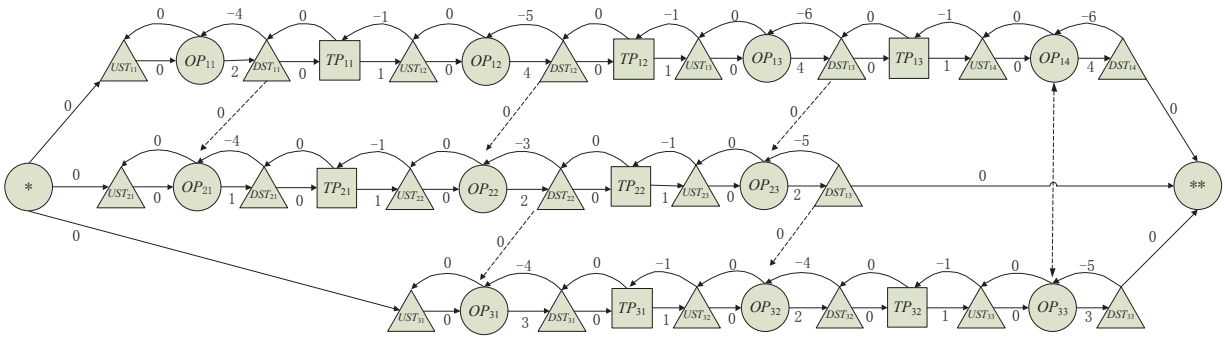

FIGURE 4.10 – Graphe disjonctif générique avec les arcs propagés pour l'exemple du tableau 4.3

Gammes identiques

Lorsque toutes les gammes, ou certaines gammes, sont totalement identiques, il est possible de généraliser la procédure de propagation précédente. Cela revient à ordonnancer :

- soit les tâches sur les ressources de transport, si le flux de produits est unidirectionnel (pas de ressources multifonctions) ;
- soit les tâches des ressources de transport et des ressources de traitement multifonctions (problème ré-entrant), dans le cas contraire.

Une autre exploitation de la présence de gammes identiques est de contrôler le nombre de travaux simultanés dans les ateliers. Comme les opérations suivent le même ordre de passage sur les ressources de traitement, les travaux entrant plus tôt sur l'atelier en seront sortis également plus tôt. L'ordre d'entrée sur la première ressource sera aussi l'ordre de sortie des produits. Le nombre maximal de travaux Js présents simultanément sur l'atelier ne dépasse donc pas le nombre de ressources de traitement $|M|$ moins 1. En effet, $Js \leq (|M|-1)$ signifie qu'il y a toujours au moins une ressource libre pour que les travaux puissent se déplacer et être traités. Dans ce cas, on peut ajouter au graphe générique des arcs additionnels qui traduisent cette hypothèse et limitent la combinatoire du problème. La figure 4.11 montre un exemple avec 4 machines, 4 *jobs* et 4 gammes identiques. L'ordre d'entrée des travaux est : $J_1 \rightarrow J_2 \rightarrow J_3 \rightarrow J_4$. D'autre part, on prend par exemple $Js = 2$. Donc J_3 ne peut commencer que quand J_1 est fini, J_4 ne peut commencer que quand J_2 est fini. Deux arcs supplémentaires peuvent donc être ajoutés : depuis la fin de J_1 vers le début de J_3 et depuis la fin de J_2 vers le début de J_4.

Il est possible qu'on ne puisse pas trouver de solution faisable avec Js travaux traités simultanément. Cela peut s'expliquer par un atelier trop chargé, ou quand certains *jobs* entrent trop tôt et/ou si trop peu de ressources de transport sont disponibles. Dans ce cas, une solution peut consister à réduire Js et chercher à nouveau les séquences faisables. Nous pouvons intégrer la contrainte Js dans la procédure SBN modifiée (cf. l'algorithme

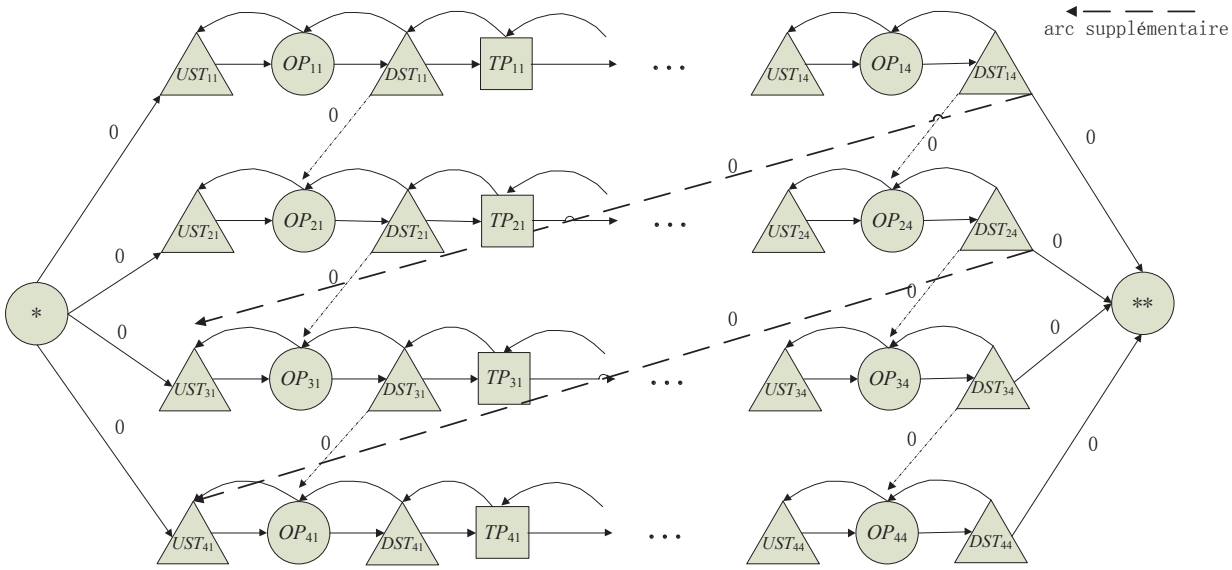

FIGURE 4.11 – Un graphe disjonctif orienté avec des arcs supplémentaires.

11).

Algorithm 11 Procédure SBN modifiée intégrant la contrainte Js

$Js \leftarrow |M| - 1$
repeat
  lancer la procédure SBN modifiée en ajoutant les arcs supplémentaires associés à Js dans le graphe générique G (l'algorithme 6)
  if une solution faisable trouvée **then**
    return cette solution.
  else
    $Js \leftarrow Js - 1$
  end if
until $Js = 0$

Cette procédure peut s'exécuter jusqu'à ce que le nombre de tâches simultanées soit égal à un. On est alors certain d'obtenir une solution faisable mais pas forcément performante. Les chances d'avoir une solution intéressante dépendent de :

- les données du problème : moins il y a de marge entre la date au plus tôt et la date au plus tard de la tâche, moins il y a de chances de trouver une séquence réalisable ;
- la taille du problème : le nombre de travaux et d'opérations de la gamme ;
- la performance des procédures heuristiques : la séquence proposée par la règle EDD ; l'efficacité de l'heuristique utilisée (algorithme H ou HA).

Ressources de traitement multi-fonctions

Une ressource de traitement est multi-fonctions si plusieurs opérations d'une même gamme utilisent cette ressource. On parle alors de problème d'ordonnancement ré-entrant. Les ressources mono-fonction peuvent être ordonnancées une par une avec l'algorithme 7. Chaque problème est de type $1|r_j|L_{max}$ avec au maximum n tâches de traitement sur la ressource associée. Pour les ressources multi-fonctions, ce sera plus compliqué et il y aura plus de chance d'obtenir une séquence non réalisable. Par exemple, dans un travail J_i, OP_{ij} et OP_{il} sont deux tâches qui doivent être traitées sur la même ressource M_k (avec j<l). Dans un autre travail J_p, une tâche OP_{pq} est traitée sur M_k. Il faut donc déterminer un ordre parmi ces trois tâches pour M_k. Supposons que la disjonction entre OP_{ij} et OP_{pq} est déjà fixée par propagation de : $OP_{ij} \rightarrow OP_{pq}$. Il reste donc l'arbitrage de la disjonction entre OP_{pq} et OP_{il}. La figure 4.12 illustre cet exemple.

Traiter OP_{pq} avant OP_{il} pourrait conduire à l'amélioration du *makespan*, si c'est faisable. La possibilité d'orienter l'arc $DST_{pq} \rightarrow OP_{il}$ dépend au moins de la longueur depuis OP_{il} jusqu'à OP_{ij}. Plus généralement, elle dépend du fait que le circuit associé ne soit pas positif. Pour cela, il faut vérifier la relation :
$-\sum_{h\in[j+1,l-1]}(P^+_{ih}+T^+_{ih})-T^+_{ij}+P^-_{pq}\leq 0$.
Il y a alors des chances que la solution soit réalisable. Sinon, elle n'est pas faisable.

Il est logique que, sur une ressource multi-fonctions, on ne puisse pas ajouter une tâche OP_{pq} qui dure trop longtemps entre deux tâches OP_{ij} et OP_{il}. En effet, cela induirait le non respect de la date limite pour OP_{il}. Il est à noter qu'ici, nous ne prenons pas en compte les risques d'avoir un circuit positif causé par les autres disjonctions orientées. Ce risque est certainement plus difficile à prévoir.

Une autre remarque est que dans le cas de problèmes avec stockage (et encore plus si ce stockage est infini), alors il n'y a pas de circuit positif, ce qui simplifie beaucoup le problème, entre autres dans l'évaluation des solutions. En effet, dans l'équation précédente, le stockage s'ajoute au terme de gauche ($\sum_{h\in[j+1,l-1]}(P^+_{ih}+T^+_{ih})$), et il suffit que sa valeur soit suffisante pour que l'inégalité soit toujours vérifiée.

Conclusion sur ces cas particuliers

Nous avons vu que le modèle générique de type graphe disjonctif que nous utilisons, est plus dense que les autres graphes utilisés dans la littérature, pour pouvoir représenter l'ensemble des contraintes considérées : plus d'arcs pour les temps maximaux, plus de nœuds (notamment on ajoute un nombre de nœuds égal à deux fois le nombre de tâches de traitement, pour pouvoir intégrer les contraintes de stockage nul ou infini). Cela induit une augmentation de la complexité et du temps de résolution, en particulier pour les instances les plus difficiles (sans attente, sans stock, avec durées bornées). Les cas particuliers de ces instances que nous avons évoqués ici, et pour lesquels nous avons proposé des procédures de traitement supplémentaires, ont principalement pour objectif d'accélérer la résolution, en intégrant dans le modèle les observations issues des caractéristiques du problème.

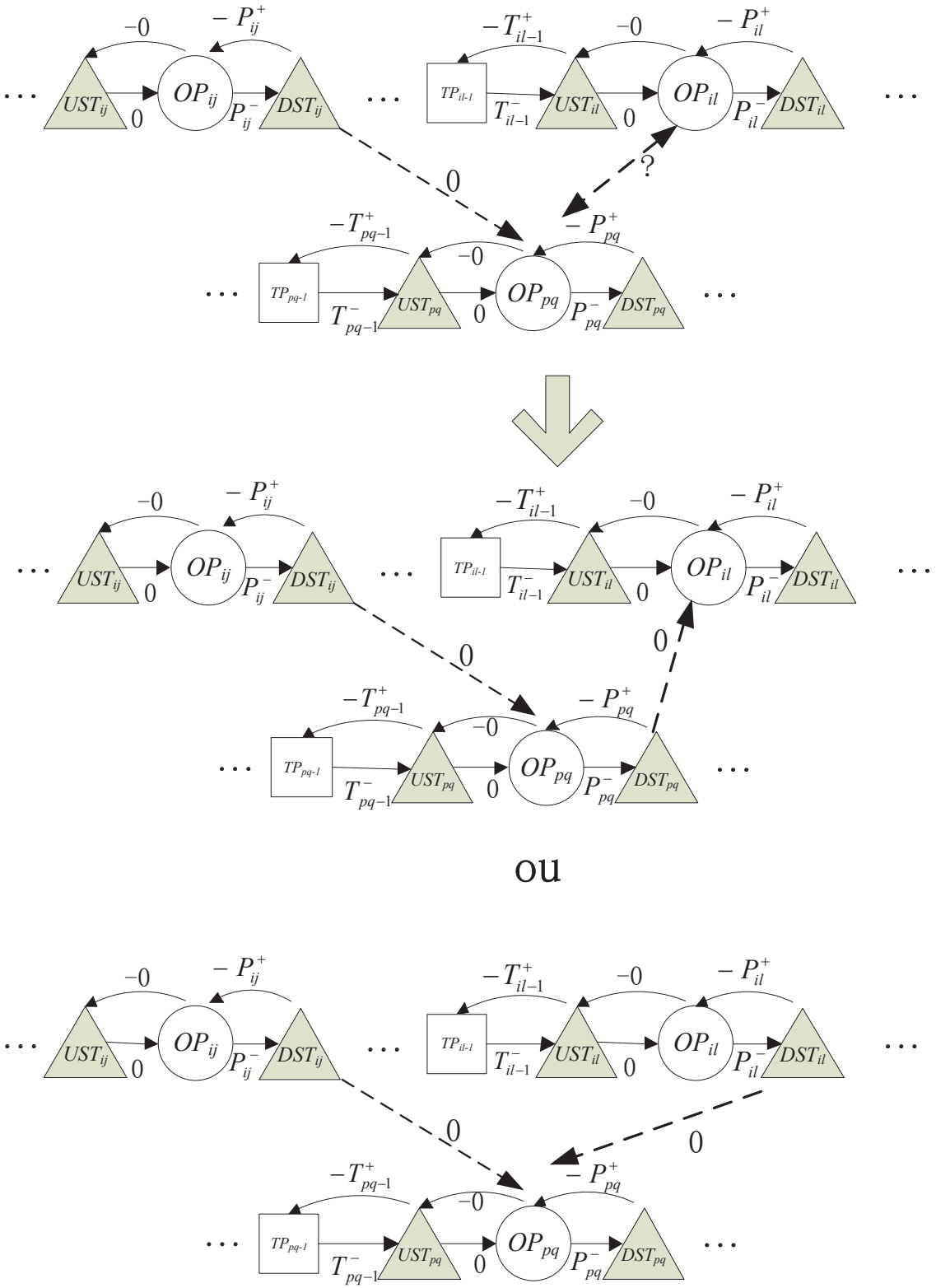

FIGURE 4.12 – Deux possibilités pour orienter une disjonction sur une ressource multi-fonctions.

Toutefois, ces cas nous ramènent finalement vers une spécialisation des algorithmes de résolution, et donc nous rapprochent de méthodes dédiées, ce qui n'est pas l'objet de notre travail. Pour satisfaire à notre but premier, il faudra donc trouver des pistes d'amélioration des méthodes proposées, tout en conservant leur caractère général.

4.4 Expérimentations

4.4.1 Description des tests

Les instances testées correspondent aux divers types d'ateliers identifiés avec contraintes de transport (FMS, RC et ATS), et aux problèmes le plus souvent associés. Nous avons défini quatre classes d'instances afin de valider notre modèle et les méthodes proposées :

- Classe 1 : elle comprend des instances de Hurink et Knust ([Hurink and Knust, 2002]). La classe 1 est caractérisée par : une seule machine disponible pour chaque tâche de traitement, une ressource de transport. Ainsi, il n'y a pas de problème d'affectation pour ces instances. Ces instances sont composées :
  - d'instances P01 de taille 6x6 (6 *jobs*, 6 machines, 6 tâches de traitement par *job*) contenant 66 tâches : 36 tâches de traitement et 30 tâches transport ;
  - d'instances P02 de plus grande taille 10x10, (10 *jobs*, 10 machines, 10 tâches de traitement par *job*) contenant 190 tâches : 100 tâches de traitement et 90 tâches de transport.
- Classe 2 : elle regroupe les instances de Bilge et Ulusoy pour les ateliers flexibles (FMS) ([Bilge and Ulusoy, 1995]), avec comme caractéristiques : une seule machine disponible pour chaque tâche de traitement, deux moyens de transport identiques, et des temps de traitement fixes. Le problème d'affectation se pose uniquement pour les tâches de transport. Les temps de déplacement des chariots sont donnés par des matrices selon 4 topologies de disposition des machines dans l'atelier. Ainsi la combinaison de 10 ensembles de *jobs* et des quatre topologies fournit 40 instances. Ces benchmarks sont nommés EX_{ij} , où i représente le numéro d'un groupe de *jobs* (i variant de 1 à 10), et j celui de la topologie utilisée pour les déplacements des chariots (j variant de 1 à 4). Par exemple EX103 représente le problème 10 où les temps de déplacement sont donnés par la topologie 3.
- Classe 3 : ce sont les instances proposées par Deroussi et al. pour un problème de *job shop* flexible (FJSP) ([Deroussi and Norre, 2010]. Elles sont basées sur des instances de [Bilge and Ulusoy, 1995], étendues à des problèmes d'affectation. Ainsi, deux machines sont disponibles pour chaque tâche de traitement ; il y a deux moyens de transport et les temps de traitement sont fixes. Il y a donc un problème d'affectation à la fois pour les tâches de traitement et de transport. Pour cette classe relativement réduite, nous avons testé nos méthodes pour deux critères : le *makespan*, mais aussi

la date de sortie du système. Dans ce deuxième cas, on cherche à minimiser la sortie du dernier produit, en prenant en compte la dernière opération de transport.

– Classe 4 : elle correspond aux instances de *Hoist Scheduling Problem* (HSP) de ([Mateo et al., 2002], [Paul et al., 2007]). Elle est caractérisée par : une seule machine disponible pour chaque tâche de traitement, une ou deux ressources de transport et des temps de traitement bornés.

Pour les trois premières classes d'instances, la capacité de stockage en amont et en aval des machines est supposée infinie. Egalement, les temps de traitement sont fixes. Dans notre modèle, cela se traduit par des bornes minimales et maximales égales à la durée du traitement associé. Seule la classe 4 correspond à des problèmes sans stock avec temps opératoires bornés. Les postes de chargement/déchargement (s'ils existent) sont considérés comme une ressource de traitement supplémentaire avec un intervalle de temps de traitement $[0, \infty[$. Enfin toutes les classes correspondent à des cas sans attente autorisée, ni sur les machines, ni sur les ressources de transport. Les temps de déplacement satisfont l'inégalité triangulaire. Les données de la classe 1 peuvent être trouvées à l'adresse : http ://www.mathematik.uni-osnabrueck.de/research/OR/robot/robot.html. Celles des autres classes sont fournies en annexe. Les caractéristiques de ces classes sont résumées dans le tableau 4.4.

TABLE 4.4 – Caractéristiques des instances

Classe	Temps de traitement	stock capa.	attente sur ress.	affectation tâches opé.	affectation tâches transp.	nb. de jobs	nb. de ress. robots	nb. de ress. machines	nb. de traitements	nb. inst.
1	Fixe	∞	Non	Non	Non	6/10	1	6/10	36/100	7/8
2	Fixe	∞	Non	Non	Oui	5 à 8	2	5*	13-21	40
3	Fixe	∞	Non	Oui	Oui	5 à 8	2	10*	13-21	10
4	Borné	0	Non	Non	Non/oui	5 à 40	1-2	5 à 18 *	35 à 290	209

* postes supplémentaires pour le chargement et déchargement à prendre en compte.

4.4.2 Résultats

Cette section présente les résultats que nous avons obtenus avec les deux méthodes de séquencement détaillées dans ce chapitre, et que nous nommons respectivement Tabou et SBN.

Les résultats correspondant à notre procédure SBN sont obtenus à partir de 1000 exécutions (10 fois un groupe d'exécutions de 100). Dans ce cas, si les instances correspondent à des ateliers flexibles, des affectations initiales des ressources de traitement sont générées aléatoirement. Les résultats de notre algorithme Tabou sont obtenus avec 1000 exécutions (10 fois un groupe d'exécutions de 100), chacune basée sur un maximum de 5000 itérations. La liste tabou contient 5 mouvements interdits. Si la longueur de cette liste est trop importante, cela limite le nombre de voisins possibles. Inversement, une taille trop

petite augmente la probabilité de boucler sur des solutions explorées au cours des itérations précédentes. Les affectations et séquences initiales sur les ressources sont générées aléatoirement. Notre programme est écrit en C++ sous linux, et il est testé sur un CPU de 2,8GHZ (AMD Phenom X2). Les temps d'exécution varient entre quelques minutes et une heure pour le Tabou, et jusqu'à trois heures pour le SBN, selon les tailles d'instances.

Le détail des résultats pour les quatre classes d'instances considérées est fourni dans les tableaux 4.5 à 4.9, en termes de *makespan* (noté C dans ces tableaux) et d'écart entre les C_{max} des solutions (noté Gap). La synthèse de ces résultats apparaît dans le tableau 4.12.

Dans ce dernier tableau, les deux méthodes Tabou et SBN montrent des performances moyennes similaires (écart relatif moyen entre 0,72% et 6%). Selon les classes, c'est l'une ou l'autre méthode qui se révèle meilleure. L'examen des résultats par classe nous permet de déduire les observations et analyses suivantes :

- classe 1 (tableau 4.5) : nos deux méthodes ne donnent pas de très bons résultats pour ces instances, alors qu'elles sont plus performantes pour les autres classes. Ceci s'explique en partie par le fait que les instances de la classe 1 ne contiennent aucun problème d'affectation. Ainsi, dans la procédure SBN, l'heuristique H n'est utilisée qu'à moitié. De plus notre SBN s'arrête dès qu'une solution faisable est trouvée, ce qui empêche toute amélioration si cette solution est peu intéressante en termes de C_{max}. En ce qui concerne l'autre méthode, notre Tabou est moins performant que celui de [Hurink and Knust, 2002]. D'une part, nous partons d'une solution initiale générée aléatoirement, ce qui peut être pénalisant. D'autre part, notre Tabou donne de meilleurs résultats pour les instances où un problème d'affectation existe. Il n'est pas sûr que ce soit le cas de [Hurink and Knust, 2002]. En effet, cette méthode est basée sur des permutations de tâches du chemin critique réalisées consécutivement par une même ressource. Le fait de dupliquer des ressources entraîne une répartition des tâches et donc moins de tâches affectées à chaque ressource. Donc cela peut diminuer drastiquement le nombre de voisins possibles.

- classe 2 (tableau 4.6) : c'est sur les instance de [Bilge and Ulusoy, 1995] que nos deux méthodes se révèlent globalement les plus performantes. En effet, les écarts par rapport aux meilleures solutions connues sont en moyenne d'environ 6 à 7%. De plus, les meilleures solutions connues sont retrouvées dans 10% des instances avec la méthode SBN, et dans 22,5% des cas avec l'algorithme Tabou. Au final, les deux méthodes présentent un écart relatif moyen de 0,78%. Nous pouvons les juger équivalentes pour cette classe.

- classe 3 : deux tableaux de résultats (tableaux 4.7 et 4.8) correspondent aux instances de *job shop* flexible avec transport proposées par [Deroussi and Norre, 2010]. Ils

TABLE 4.5 – Résultats pour les instances de classe 1

Inst.	Ref.	C_{SBN}	$Gap_1(\%)$	C_{Tabou}	$Gap_2(\%)$	$Gap_3(\%)$
P01D1d1	87	156	79,31	106	21,84	-32,05
P01D1t1	81	93	14,81	91	12,35	-2,15
P01T2t1	74	93	25,68	83	12,16	-10,75
P01T3t0	92	95	3,26	93	1,09	-2,11
P01D3d1	216	224	3,70	235	8,80	4,91
P01D2d1	148	158	6,76	169	14,19	6,96
P02D1d1	1013*	1454	43,67	1344	32,81	-7,57
P02D1t0	989	1430	44,59	1283	29,73	-10,28
P02D1t1	995*	1490	51,58	1382	40,59	-7,25
P02D2d1	1004	1576	56,97	1380	37,45	-12,44
P02D3d1	1078	1567	45,36	1512	40,26	-3,51
P02D5t2	1383*	1576	15,80	1683	23,66	6,79
P02T1t1	1022*	1376	40,69	1271	29,96	-7,63
P02T2t1	1053*	1394	40,38	1280	28,90	-8,18
P02T5t2	1090*	1413	38,26	1344	31,51	-4,88
moyenne			34,06%		24,35%	-6,01%

$Gap_1 = (C_{SBN} - Ref.)/Ref.$
$Gap_2 = (C_{Tabou} - Ref.)/Ref.$
$Gap_3 = (C_{Tabou} - C_{SBN})/C_{SBN}$
Ref : [Hurink and Knust, 2005]
Ref* : résultats de [Lacomme et al., 2010] meilleurs que ceux de [Hurink and Knust, 2005]

TABLE 4.6 – Résultats pour les instances de classe 2

Inst.	Ref.	C_{SBN}	$Gap_1(\%)$	écartype	C_{Tabou}	$Gap_2(\%)$	écartype	$Gap_3(\%)$
Ex11	96	97	1,04	0,42	97	1,04	2,27	0,00
Ex12	82	82	0	0,84	82	0	2,11	0,00
Ex13	84	88	4,76	0	86	2,38	0,67	-2,27
Ex14	103	110	6,8	0,32	108	4,85	2,32	-1,82
Ex21	100	109	9	0,67	107	7	2,22	-1,83
Ex22	76	80	5,26	0	80	5,26	2,49	0
Ex23	86	87	1,16	0	86	0	1,84	-1,15
Ex24	108	126	16,67	0,97	118	9,26	3,37	-6,35
Ex31	99	110	11,11	0	112	13,13	1,96	1,82
Ex32	85	87	2,35	0	85	0	2,91	-2,3
Ex33	86	89	3,49	0	93	8,14	1,64	4,49
Ex34	111	136	22,52	0	123	10,81	3,47	-9,56
Ex41	112	131	16,96	0	124	10,71	1,96	-5,34
Ex42	87	96	10,34	0	98	12,64	2,64	2,08
Ex43	89	99	11,24	0	101	13,48	1,87	2,02
Ex44	121	132	9,09	0,32	136	12,4	3,13	3,03
Ex51	87	90	3,45	0	90	3,45	1,96	0
Ex52	69	73	5,8	0	69	0	2,27	-5,48
Ex53	74	76	2,7	0	74	0	2,58	-2.36
Ex54	96	99	3,13	0	96	0	3,28	-3,03
Ex61	118	123	4,24	0	124	5,08	3,02	0,81
Ex62	98	104	6,12	0	105	7,14	1,81	0,96
Ex63	103	106	2,91	0	107	3,88	2,30	0,94
Ex64	120	140	16,67	0	134	11,67	3,03	-4,29
Ex71	111	122	9,91	0,85	124	11,71	2,20	1,64
Ex72	79	86	8,86	0	91	15,19	2,16	5,81
Ex73	83	91	9,64	0	96	15,66	1,79	5,49
Ex74	126	149	18,25	0,63	141	11,9	3,69	-5,37
Ex81	161	161	0	0	161	0	0,00	0
Ex82	151	151	0	0	151	0	0,00	0
Ex83	153	153	0	0	153	0	0,00	0
Ex84	163	172	5,52	0	167	2,45	2,01	-2,91
Ex91	116	123	6,03	0	121	4,31	3,88	-1,63
Ex92	102	107	4,9	0	109	6,86	1,65	1,87
Ex93	105	107	1,9	0,42	112	6,67	2,36	4,67
Ex94	120	125	4,17	0	131	9,17	3,89	4,8
Ex101	146	163	11,64	0	152	4,11	3,43	-6,75
Ex102	135	147	8,89	0	144	6,67	2,11	-2,04
Ex103	137	149	8,76	0,32	148	8,03	3,16	-0,67
Ex104	157	189	20,38	0,32	177	12,74	2,64	-6,35
moyenne			7,39%	0,15		6,45%	2,30	-0,78%

$Gap_1 = (C_{SBN} - Ref.)/Ref.$
$Gap_2 = (C_{Tabou} - Ref.)/Ref.$
$Gap_3 = (C_{Tabou} - C_{SBN})/C_{SBN}$
Ref : [Larabi, 2010]

TABLE 4.7 – Résultats pour les instances de classe 3 avec critère=date de sortie

Inst.	Ref.	C_{SBN}	$Gap_1(\%)$	écartype SBN	C_{Tabou}	$Gap_2(\%)$	écartype Tabou	$Gap_3(\%)$
fjsp1	160	176	10	1,26	188	17,5	3,41	6,82
fjsp2	138	138	0	1,27	154	11,59	3,71	11,59
fjsp3	142	152	7,04	0,32	164	15,49	4,60	7,89
fjsp4	138	148	7,25	0,95	160	15,94	2,95	8,11
fjsp5	112	116	3,57	1,89	122	8,93	1,35	5,17
fjsp6	158	166	5,06	1,35	178	12,66	5,56	7,23
fjsp7	150	166	10,67	0,84	174	16	4,83	4,82
fjsp8	197	204	3,55	1,08	208	5,58	4,84	1,96
fjsp9	166	176	6,02	0,67	180	8,43	3,98	2,27
fjsp10	188	206	9,57	1,91	216	14,89	4,88	4,85
moyenne			6,27%	1,15		12,70%	4,01	6,07%

$Gap_1 = (C_{SBN} - Ref.)/Ref.$
$Gap_2 = (C_{Tabou} - Ref.)/Ref.$
$Gap_3 = (C_{Tabou} - C_{SBN})/C_{SBN}$
Ref : [Deroussi and Norre, 2010]

fournissent les résultats respectivement en termes de date de sortie du système et de *makespan*. La date de sortie est obtenue en considérant le maximum entre les dates de fin des derniers traitements additionnés de leur temps de transport vers la sortie du système. Le *makespan* ne tient pas compte de ce dernier transport (date de fin de la dernière tâche de traitement). Pour le critère du *makespan*, nous n'avons pas les résultats de référence, ce qui explique que nous donnions seulement une comparaison relative des deux méthodes. Pour cette classe, où l'affectation doit être réalisée sur toutes les ressources, la procédure SBN fournit de meilleurs résultats que le Tabou. Ceci peut s'expliquer par le fait que l'affectation initiale est fixée aléatoirement pour toutes les ressources avec notre Tabou. Les modifications sont très limitées, notamment au niveau du transport. Par contre, avec notre SBN, cette affectation n'est imposée initialement que pour les ressources de traitement. Pour le transport, l'heuristique H est dédiée à la recherche d'une affectation plus intéressante des tâches.

- classe 4 (tableaux 4.9 à 4.11) : les premiers tableaux sont relatifs aux instances de [Mateo et al., 2002]. Le dernier concerne les instances de [Paul et al., 2007]. En fait assez peu de benchmarks sont disponibles dans la littérature sur le HSP. Ils sont le plus souvent dédiés aux problèmes cycliques. Ainsi, les instances de [Mateo et al., 2002] ont été initialement générées pour la résolution de problèmes 2-cycliques, avec 2 types de *jobs* entrant alternativement sur la ligne. Les solutions de référence fournies (Ref. T, voir en annexe H) correspondent aux solutions optimales, non pas en terme de C_{max}, mais en termes de période de la séquence 2-cyclique des mouvements de transport. Il s'agit du temps écoulé entre les dates d'entrée des kème et k+2ème produits arrivant sur

TABLE 4.8 – Résultats pour les instances de classe 3 avec critère=makespan

Instance	C_{SBN}	C_{Tabou}	$Gap(\%)$
fjsp1	156	160	2,56%
fjsp2	124	128	3,23%
fjsp3	140	162	15,71%
fjsp4	132	126	-4,55%
fjsp5	96	100	4,17%
fjsp6	148	152	2,70%
fjsp7	132	132	0,00%
fjsp8	191	188	-1,57%
fjsp9	154	162	5,19%
fjsp10	192	186	-3,13%
moyenne			2,43%

$Gap = (C_{Tabou} - C_{SBN})/C_{SBN}$

l'atelier. Nous ne pouvons donc pas comparer nos résultats directement avec ces périodes de référence. Toutefois, nous pouvons déduire un *makespan* de référence pour n *jobs* :

$$C_{max}ref = \lfloor n/2 \rfloor \times T + tt1.$$

$tt1$ est le temps passé sur la ligne par le dernier *job* (supposé de type 1) (voir figure 4.13). Il est égal à la somme des temps opératoires et des temps de transport de ce *job*. Mais [Mateo et al., 2002] ne fournissent pas les durées opératoires réelles des solutions. Néanmoins, à partir des fenêtres temporelles des temps de traitement, nous pouvons déduire des bornes minimale et maximale de $tt1$, donc de $C_{max}ref$ (respectivement LB et UB) :

$LB = \lfloor n/2 \rfloor \times T + \sum_{j\in[1,n_1]} P^-_{1j} + \sum_{j\in[1,n_1-1]} T^-_{1j}$,
$UB = \lfloor n/2 \rfloor \times T + \sum_{j\in[1,n_1]} P^+_{1j} + \sum_{j\in[1,n_1-1]} T^-_{1j}$.

Dans ce contexte, nous avons tout d'abord comparé les performances de nos deux méthodes pour des petites instances de [Mateo et al., 2002] : 5 cuves, 2 robots et 5 *jobs* (tableau 4.9). Cela fait 35 traitements et 30 tâches de transport à ordonnancer. Comparés aux bornes minimales (LB) de la référence pour des problèmes à 1 robot, nos résultats montrent des améliorations moyennes de 34,24% pour SBN et 32,15% pour Tabou. L'ajout d'une ressource de transport est donc ici toujours avantageux. De manière plus générale, les performances de nos deux méthodes sont sensiblement identiques, avec un écart en faveur de la procédure SBN pour ces instances (écart relatif moyen de 3,35%). Mais l'algorithme Tabou a trouvé ses limites, et n'a pas pu fournir de solution faisable pour des problèmes de plus grande taille : 5 à 10 cuves, 1 ou 2 robots avec 21 jobs pour [Mateo et al., 2002] (soit 147 à 252 tâches de traitement) ; 18 cuves, 1 robot et 40 jobs pour [Paul et al., 2007]) (pour environ 290 tâches de traitement). Par rapport aux autres tailles d'instances résolues dans d'autres classes, il est clair que les problèmes de type HSP sont plus

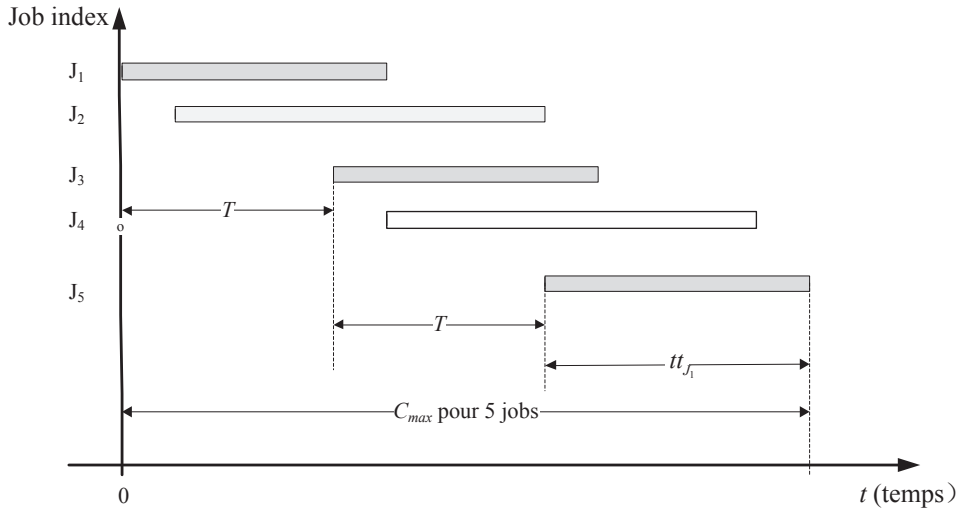

FIGURE 4.13 – Relation entre la période T et C_{max} pour les instances de Mateo et al. avec 5 *jobs*

contraints et que l'espace des solutions faisables est plus réduit. Il est donc très difficile dans ces conditions de trouver un bon voisin. Les tableaux 4.10 et 4.11 présentent donc les résultats uniquement de notre procédure SBN (qui est plus performante ici par ailleurs) pour l'ensemble des instances de HSP considérées pour cette classe. Ces résultats sont obtenus sur la base de dix exécutions.

Chaque ligne du tableau 4.10 correspond à la synthèse des résultats de toutes les instances associées au nombre de cuves donné, soit 30 instances obtenues pour chaque nombre de cuves, en faisant varier les fenêtres temporelles et la vitesse du robot. Ce tableau synthétise donc les tests réalisés sur 150 instances de HSP. Le C_{max} obtenu avec un seul robot y est comparé aux bornes du *makespan* de référence, LB et UB, telles ques définies précédemment. La valeur Gap_{LB} sur chaque ligne est la moyenne des écarts obtenus entre notre C_{max} et LB pour toutes les instances correspondant à un nombre de cuves donné. Notons que LB ne correspond pas forcément à une solution cyclique faisable, puisqu'il a été calculé artificiellement sur la base de toutes les durées opératoires minimales. En moyenne, on constate que LB$<C_{max}<$UB. Cela signifie que notre résultat est proche du *makespan* de référence réel (si ce n'est meilleur). Toutefois, le détail des résultats fourni en annexe H montre que notre C_{max} est effectivement inférieur à la borne minimale LB dans 19,3 % des cas (pour 6 à 10 cuves). En effet, la solution cyclique est optimale dans un fonctionnement en régime permanent. Mais dans le contexte de *job shop*, on part d'un atelier vide a priori. Cela veut dire que nous considérons les régimes transitoires en début et fin de production. Cela explique que le *makespan* de référence peut ne

TABLE 4.9 – Résultats pour les instances de classe 4 : 5 cuves, 2 robots, 5 *jobs*

Inst.	Ref. LB	C_{SBN}	$Gap_1(\%)$	C_{Tabou}	$Gap_2(\%)$	$Gap_3(\%)$
51	1106	795	-28,12	770	-30,38	-3,14
52	982	723	-26,37	754	-23,22	4,29
53	1294	714	-44,82	781	-39,64	9,38
54	989	647	-34,58	709	-28,31	9,58
55	1144	697	-39,07	801	-29,98	14,92
56	1085	727	-33,00	716	-34,01	-1,51
57	1248	708	-43,27	803	-35,66	13,42
58	961	612	-36,32	711	-26,01	16,18
59	971	642	-33,88	692	-28,73	7,79
510	1029	625	-39,26	642	-37,61	2,72
511	1246	765	-38,60	777	-37,64	1,57
512	862	684	-20,65	740	-14,15	8,19
513	1096	741	-32,39	769	-29,84	3,78
514	1110	774	-30,27	759	-31,62	-1,94
515	903	587	-34,99	624	-30,90	6,30
516	1274	756	-40,66	786	-38,30	3,97
517	1110	760	-31,53	713	-35,77	-6,18
518	1092	725	-33,61	754	-30,95	4,00
519	1224	825	-32,60	834	-31,86	1,09
520	903	556	-38,43	650	-28,02	16,91
521	1135	728	-35,86	757	-33,30	3,98
522	1710	1107	-35,26	1037	-39,36	-6,32
523	1559	929	-40,41	969	-37,84	4,31
524	992	701	-29,33	695	-29,94	-0,86
525	1554	958	-38,35	877	-43,56	-8,46
526	1486	969	-34,79	866	-41,72	-10,63
527	1127	805	-28,57	789	-29,99	-1,99
528	1008	692	-31,35	771	-23,51	11,42
529	1156	865	-25,17	840	-27,34	-2,89
530	1523	979	-35,72	985	-35,33	0,61
moyenne			-34,24%		-32,15%	3,35%

$Gap_1 = (C_{SBN} - Ref.LB)/Ref.LB$
$Gap_2 = (C_{Tabou} - Ref.LB)/Ref.LB$
$Gap_3 = (C_{Tabou} - C_{SBN})/C_{SBN}$
Ref : déduite de [Mateo et al., 2002]

TABLE 4.10 – Synthèse des résultats de classe 4 : 6 à 10 cuves, 21 *jobs*, 1 et 2 robots [Mateo et al., 2002]

nb. de cuves	1 robot (écart moyen) $Gap_{LB}\%$	$Gap_{UB}\%$	2 robots (moyenne) $Gap\%$
6	2,59	-4,74	-42,43
7	2,30	-5,28	-42,75
8	5,79	-1,68	-42,78
9	5,27	-1,91	-45,45
10	5,88	-1,39	-44,80
moyenne	4,37%	-3%	-43,64%

Gap_{LB} = moyenne des $(C_1 - LB)/LB$
Gap_{UB} = moyenne des $(C_1 - UB)/UB$
Gap = moyenne des $(C_2 - C_1)/C_1$
C_1 : makespan pour 1 robot
C_2 : makespan pour 2 robots

pas correspondre pas à un ordonnancement optimal dans un contexte non cyclique du *job shop*. En conséquence, il est effectivement possible d'avoir de meilleurs résultats que la borne inférieure associée LB. Comme précédemment, la résolution avec 2 robots améliore substantiellement les performances du système.

Enfin, notre SBN donne de meilleurs résultats que l'heuristique ATW de [Paul et al., 2007] dans 78,57% des cas (tableau 4.11), avec une amélioration moyenne allant de 6,45% jusqu'à 20%. Ici, l'environnement de production des instances est directement de type job shop, ce qui nous permet de valider directement l'efficacité de notre algorithme, quoique le temps de résolution étant moins intéressant (de l'ordre de l'heure, contre un temps de l'ordre de la seconde pour [Paul et al., 2007]).

4.5 Bilan

Dans ce chapitre, nous avons proposé deux méthodes de résolution approchées pour le problème de séquencement rencontré dans le *General Flexible Job Shop Scheduling Problem* (GFJSSP). La première est un algorithme Tabou. La seconde combine une procédure de Shifting Bottleneck s'appuyant sur notre graphe disjonctif générique, et sur une heuristique dédiée aux tâches de transport. Les deux méthodes considèrent une affectation prédéfinie des tâches de traitement. Par contre pour les tâches de transport, la première travaille à partir d'une affectation initialement donnée, tout en autorisant des modifications en cours de résolution. La seconde méthode ne tient pas compte d'une affectation préalable. Elle affecte et séquence simultanément les opérations de transport au fur et à mesure de la résolution. Les tests réalisés sur plusieurs classes d'instances de la littérature, qui correspondent à divers cas possibles de GFJSSP. Ils montrent que :

TABLE 4.11 – Résultats pour les instances de classe 4 : [Paul et al., 2007]

Inst.	Ref. C_{max} moyens		C_{max} avec SBN			
	C_{ATW}	écartype	meilleur	moyenneC_{SBN}	écartype	Gap_{ATW}(%)
1	28,69	1,72	22,56	23,71	1,23	-17,38
2	27,94	2,63	24,27	24,10	1,02	-13,75
3	27,76	2,63	21,10	23,11	1,12	-16,74
4	27,60	3,27	21,56	25,65	2,13	-7,08
5	27,91	3,23	20,60	22,21	1,34	-20,39
6	26,73	2,76	22,34	22,30	1,01	-16,55
7	27,37	3,03	22,79	25,31	1,29	-7,52
8	27,69	2,73	24,01	27,38	2,02	-1,11
9	26,88	2,83	22,04	24,22	1,87	-9,88
10	27,07	3,05	26,11	27,89	1,05	3,03
11	27,74	3,48	24,78	26,83	1,69	-3,30
12	26,49	3,00	23,35	25,95	1,83	-2,04
13	27,27	3,22	26,44	28,80	1,13	5,59
14	26,63	1,99	28,18	31,12	1,43	16,87
moyenne		2,83			1,44	-6,45%

$Gap_{ATW} = (C_{SBN} - C_{ATW})/C_{ATW}$
Ref : [Paul et al., 2007]

TABLE 4.12 – Synthèse des résultats

	écart		classe 1	classe 2	classe 3		classe 4	
					sortie	C_{max}	Mateo	Paul
SBN	moyen		34,06	7,39	6,27	non dispo	-34,24	-6,45
	min		3,26	0	0	non dispo	-44,82	-20,39
	max		79,31	22,52	10,67	non dispo	-20,65	16,87
Tabou	moyen		24,35	6,45	12,7	non dispo	-32,15	non dispo
	min		1,09	0	5,58	non dispo	-43,56	non dispo
	max		40,59	15,66	17,5	non dispo	-14,15	non dispo
comparaison	moyen	relatif	-6,01	-0,72	6,07	2,43	3,35	
	min	relatif	-32,05	-6,75	1,96	-4,55	-10,63	
	max	relatif	6,96	5,81	11,59	15,71	16,91	
meilleure	méthode		Tabou	Tabou	SBN	SBN	SBN	SBN

- pour les problèmes de petite taille, nos deux méthodes sont relativement équivalentes (écart relatif moyen identique), avec une dominance du Tabou pour les instances sans flexibilité de traitement et/ou avec temps fixes. Cette petite différence peut s'expliquer par le fait que le Tabou cherche à améliorer plusieurs fois la solution courante, alors que le SBN s'arrête dès qu'il a trouvé une solution faisable ;
- pour les problèmes de plus grande taille, avec flexibilité de traitement ou temps bornés et sans stock (par exemple pour le *Hoist Scheduling Problem*), le SBN est plus efficace, en particulier pour trouver des solutions faisables, même si son temps de résolution est plus long.

Les résultats obtenus ne peuvent pas concurrencer ceux des méthodes dédiées à chaque classe. Toutefois, ils ne sont pas si éloignés de certains des meilleurs résultats de la littérature. Par ailleurs, on remarque que plus les instances sont complexes, plus cet écart a tendance à diminuer, voire à s'annuler. L'amélioration de ces méthodes passe par la prise en compte de deux verrous identifiés ici : le premier vient du fait que l'affectation des tâches de traitement, d'une part est générée aléatoirement, d'autre part n'est pas remise en cause. Cela peut être particulièrement limitatif pour atteindre de bonnes solutions (voire la meilleure), dans le cas de problèmes de *job shop* flexibles. Le second verrou, en amont même de ce problème, concerne l'obtention de solutions faisables. Cette recherche peut en effet s'avérer critique pour les instances sans attente et sans stock. Le chapitre suivant s'intéresse à ces deux aspects, en étendant l'étude du problème à l'affectation à la fois des ressources de traitement et de transport.

Chapitre 5

Affectation et Séquencement

5.1 Introduction

Dans la partie précédente, nous avons introduit deux méthodes heuristiques de séquencement dans le cadre d'un problème statique. Ces méthodes fonctionnent sur la base d'une affectation des tâches préalablement fixée, totalement ou partiellement. Elles prennent en compte cette affectation et/ou la changent dynamiquement (au cours de la résolution), pour augmenter les chances de trouver de bonnes solutions faisables.

Leur performance dépend donc aussi de ces données de départ. On peut ainsi espérer qu'une bonne affectation initiale des tâches nous aide à trouver une solution correcte dans un bref délai. Mais la génération d'une affectation intéressante n'est pas si évidente. De plus, elle peut paradoxalement limiter l'exploration de l'espace de recherche à une zone éloignée de la solution optimale. Pour pallier ce problème, nous avons opté pour l'utilisation d'une approche par algorithme génétique dédiée à la problématique de l'affectation. Nous proposons tout d'abord une procédure combinée avec notre algorithme Tabou dédié au séquencement. Ensuite, notre procédure SBN modifiée est ajoutée dans cette procédure pour améliorer les performances de cette méthode hybride pour des instances plus complexes.

Ce chapitre est consacré à la description de nos contributions pour la prise en compte à la fois de l'affectation et du séquencement dans le GFJSSP. Nous donnons tout d'abord le principe de fonctionnement et la structure générale d'un algorithme génétique. Puis, nous définissons le codage du chromosome associé à chaque solution, et nous présentons les opérateurs et les différents éléments de notre algorithme génétique. Ensuite, la structure globale des algorithmes hybrides que nous proposons est détaillée. Enfin, les résultats des tests effectués sur les benchmarks de la littérature utilisés au chapitre précédent sont fournis et analysés.

5.2 Algorithme génétique

5.2.1 Principe

Les algorithmes génétiques (GA) ont été mis au point par John Holland dans les années 1960 ([Holland, 1992]). Il s'agit d'une heuristique de recherche dite à base de population, qui imite l'évolution biologique. Les algorithmes génétiques appartiennent à la classe des algorithmes évolutionnaires (AE), qui génèrent des solutions aux problèmes d'optimisation en utilisant des techniques inspirées par l'évolution naturelle.

Un algorithme génétique considère un ensemble d'individus, appelés chromosomes (ou génotype), et regroupés en population. Ces individus sont codés pour représenter des solutions d'un problème d'optimisation. Traditionnellement, les solutions sont représentées en binaire, mais d'autres codages sont également possibles. L'évolution commence généralement à partir d'une population d'individus générée aléatoirement. Puis la population est améliorée par des phases répétées d'évolution. A chaque génération, l'adaptation (*fitness*) de chaque individu dans la population est évaluée. Un opérateur de sélection est appliqué en fonction de la valeur de cette adaptation, afin de choisir certains individus appelés parents. L'étape suivante consiste à générer la population de la prochaine génération depuis les parents choisis, en appliquant des opérateurs de croisement (à des couples de solutions) et de mutation (à un parent à la fois). Les parents sont sélectionnés pour générer une ou plusieurs nouvelles solutions appelées enfants, jusqu'à atteindre la taille appropriée de population. Certaines recherches ([Eiben et al., 1994], [Ting, 2005], suggèrent que le choix de plus de deux parents peut produire un chromosome de bonne qualité. La nouvelle population est ensuite utilisée dans la prochaine itération de l'algorithme. L'algorithme se termine lorsqu'un critère donné est atteint, comme un nombre maximal de générations, l'obtention d'une solution avec une valeur d'adaptation suffisante, le temps d'exécution, etc. Un schéma de principe des algorithmes génétiques est donné dans la figure 5.1.

5.2.2 Codage du chromosome

Pour les algorithmes évolutionnaires, un des facteurs importants est la façon dont sont codées les solutions (ce que l'on a nommé ici les chromosomes), c'est-à-dire les structures de données qui représenteront les solutions. Le codage est la première étape avant d'appliquer l'algorithme génétique. Il détermine en grande partie l'efficacité de l'évolution génétique. En particulier, il conditionne en partie le choix des opérateurs de l'algorithme, par exemple le croisement et la mutation.

Il a été proposé à ce jour de nombreuses méthodes de codage différentes :

- le codage binaire : coder la solution comme une chaîne de bits, appelés gènes, qui prennent pour valeur 0 ou 1. Ce codage a été proposé par Holland en même temps qu'il a proposé les algorithmes génétiques. C'est la méthode de codage la plus souvent utilisée et la plus simple à appliquer (par exemple, le problème de sac à dos). Le

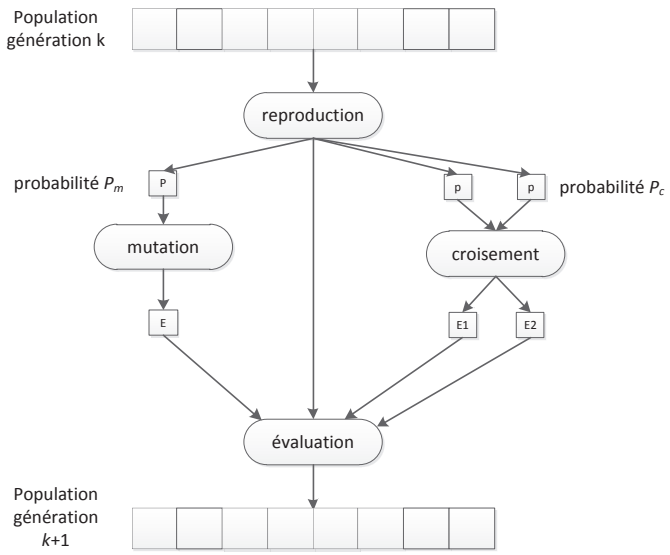

FIGURE 5.1 – Principe général des algorithmes génétiques

point négatif de ce codage est sa difficulté à s'adapter à certains types de problèmes ;

- le codage entier : la valeur de chaque gène est entière. Par exemple le codage du problème du voyageur de commerce (TSP) peut être une listes ordonnée des clients à visiter, avec un nombre entier associé à chaque client ;
- le codage réel : la valeur de chaque gène est un nombre réel, et la longueur du chromosome est égale au nombre de variables. Ce codage a tout d'abord été proposé par Michalewicz ([Michalewicz et al., 1992]). Il est plus adapté pour représenter certains types de solution dans le cas de problèmes réels, sans procédure spécifique de codage/décodage ;
- le codage symbolique : coder les chromosomes à l'aide de caractères multiples ([Choi et al., 2009]). Ce type de codage est plus adapté aux problèmes naturels. Il est donc aussi utilisé dans de nombreux cas ;
- le codage sous forme d'arbre : chaque chromosome est représenté par un arbre, chaque gêne correspond à un nœud de l'arbre, et chaque nœud sauvegarde ses parents et ses fils. Il est donc pratique pour les problèmes qui sont naturellement représentés sous forme d'arbre ou de graphe (souvent sous forme d'arbre binaire). Ce codage peut être construit avec une structure d'arbre qui simplifie le calcul des objectifs avec les opérateurs dédiés. Mais il n'assure pas qu'après avoir appliqué les opérateurs, les solutions associées aux arbres sont encore faisables. Pour les problèmes réels, il sera préférable de limiter la profondeur d'arbre afin de simplifier la complexité.

En plus des codages les plus utilisés, il y a aussi de nombreux autres codages : codage de Delta ([Whitley et al., 1991]), codage de Grey, codage des octets (bytes) dynamiques

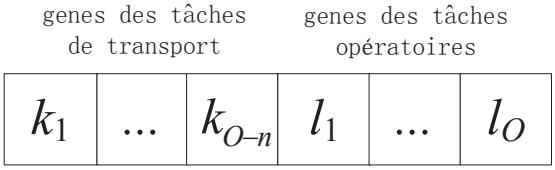

FIGURE 5.2 – Codage du chromosome.

([Mei et al., 2009], codage de chaîne, codage en matrice ([Gottlieb and Paulmann, 1998]), codage des "bytes quantum" ([Narayanan and Moore, 1996]), etc.

Pour intégrer l'aspect affectation à notre méthode de résolution du GFJSSP, nous avons utilisé un codage en nombres entiers. Chacun des individus participant à la procédure génétique est codé selon une représentation spécifique qui reflète l'affectation de toutes les tâches opératoires et de transport. Un chromosome est donc composé de deux parties distinctes, comme illustré par la figure 5.2. Chaque gène du chromosome correspond à une tâche de transport ou de traitement. Sa valeur représente le numéro de la ressource choisie parmi l'ensemble des ressources qui peuvent traiter cette tâche. Dans notre codage, les $O - n$ premiers gènes correspondent aux tâches de transport $(TP_{11}, ..., TP_{nO_n-1})$, et la valeur de chaque gène k_i $(1 \leq i \leq O-n)$ correspond au numéro de la ressource de transport choisie $(k_i \in [1, h])$; les O gènes restants correspondent à toutes les tâches opératoires $(OP_{11}, ..., OP_{nO_n})$, la valeur de chaque gène correspond au numéro de la ressource de traitement choisie parmi l'ensemble des ressources possibles. La population initiale d'individus est générée aléatoirement.

Les avantages du codage proposé sont :

- l'aptitude à représenter une solution : à l'aide de ce codage, un chromosome peut représenter un ensemble de solutions qui ont la même affectation des ressources. Si notre chromosome représentait une unique solution, donc à la fois l'affectation et le séquencement des ressources (par exemple voir [Larabi, 2010], cela serait certainement pénalisant dans le cas de problèmes sans stock que nous traitons (au même titre que ceux avec stock infini). En effet, dans le premier cas, comme nous avons pu le voir dans le tableau 4.1, nous courons le risque de générer une majorité d'individus non faisables. Avec notre codage, et donc pour une affectation donnée, nous nous offrons la possibilité de trouver au moins une séquence faisable associée ;
- des opérateurs classiques, non dédiés à un type d'instances ;
- un départ facile pour démarrer l'heuristique d'amélioration : pour un individu, les heuristiques peuvent commencer à partir des informations fixées par son chromosome, puis améliorer la solution.

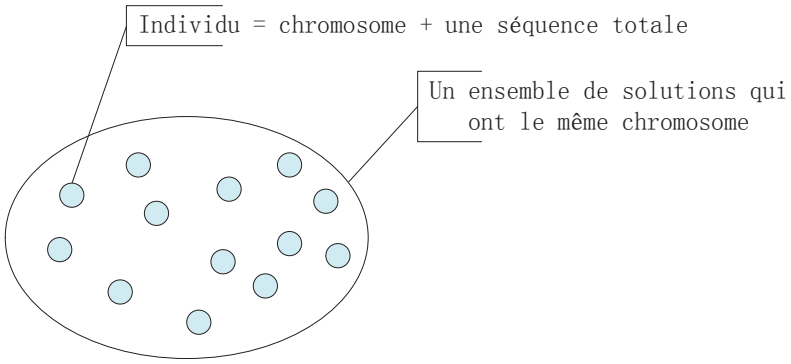

FIGURE 5.3 – Relation entre individu, chromosome et la séquence totale.

Comme nous venons de le dire, notre chromosome est insuffisant pour représenter complètement une solution de notre problème, puisqu'il peut correspondre à plusieurs ordonnancements. Toutefois, au cours de la procédure de résolution, quelle qu'elle soit, il faudra évaluer des solutions. Dans ce cas, l'évaluation nécessite d'associer à chaque chromosome, une séquence totale des tâches de traitement. Celle-ci permet de définir un ordre total des opérations pour chaque machine et chaque ressource de transport. Ainsi notre individu, et donc une solution du problème, est complètement défini par cette association du chromosome et de la séquence totale. La figure 5.3 montre la relation entre les individus, les solutions, les chromosomes et la séquence totale. Dans la procédure que nous proposons ici, la population initiale est générée aléatoirement. En fait, nous verrons par la suite que la séquence totale initiale de chaque individu peut être générée de différentes façons.

Il existe des points négatifs de ce codage simple basé sur les affectations des ressources :

- une diversité limitée : ce codage simple ne peut que représenter les différentes affectations des ressources pour toutes les tâches, ce qui veut dire que la diversité du chromosome est limitée par les tailles des ensembles de ressources pour chaque tâche. Pour les cas où il n'y pas de problème d'affectation, tous les chromosomes sont identiques.
- l'efficacité limitée des opérateurs : comme la diversité du chromosome est limitée, l'efficacité des opérateurs, qui fonctionnent sur les gènes des chromosomes, est aussi limitée. Dans les cas où il n'y pas de problème d'affectation, les opérateurs ne fonctionnent plus.

5.2.3 Opérateurs : sélection, croisement, mutation

Les algorithmes génétiques, afin de faire évoluer la population d'individus, se basent sur les opérateurs principaux suivants : sélection, croisement et mutation. La sélection

définit la technique de choix de couples de chromosomes parents. Le croisement est ensuite appliqué sur ces deux chromosomes pour construire un nouvel individu qui survivra dans la prochaine génération. La mutation est effectuée sur une petite portion d'individus pour avoir plus de diversité dans la nouvelle population.

Sélection

La sélection est appliquée afin de favoriser la reproduction des meilleurs individus avec l'opérateur de croisement. En général, il existe plusieurs techniques de sélection, par exemple :

- uniforme : chaque individu a la même probabilité $1/P$ d'être sélectionné, où P est le nombre total d'individus dans la population ;
- par roulette ([Goldberg, 1989]) : la probabilité de sélection est proportionnelle à la valeur d'adaptation : chaque individu I_i a une probabilité $\frac{F_i}{\sum_1^P F_i}$ d'être sélectionné, basée sur son adaptation F_i, où P est la taille de la population. Tous les individus sont placés depuis 0 à 1 sur une roulette construite selon leurs probabilités. Ensuite, un nombre est généré aléatoirement entre $[0, 1]$. L'individu qui correspond à cette valeur dans la roulette est donc choisi. Cette stratégie signifie que l'individu qui a la plus grande valeur d'adaptation a plus de chance d'être sélectionné, ce qui correspond plus au cas de la nature ;
- par rang : tous les individus sont d'abord classés par ordre croissant de la fitness. Puis on attribue un rang à chacun d'eux en fonction de son classement. Le plus mauvais individu aura le rang 1, etc. Chaque individu a une chance d'être choisi proportionnelle à son rang. Cette méthode peut entraîner un ralentissement de la convergence, car les meilleurs chromosomes ne diffèrent pas tellement des plus mauvais ;
- par tournoi à deux et trois : cela se fait en comparant des paires ou des triplets d'individus tirés aléatoirement dans la population. Le meilleur de la paire ou du triplet est sélectionné ;
- *steady state* : le principe de cette stratégie consiste à sélectionner aléatoirement des chromosomes parmi les k meilleurs individus, où $k <$ la taille de la population. Ce principe encourage la reproduction des meilleurs individus, et exclut les mauvais en ne leur donnant aucune chance de survivre. Un des inconvénients est que la diversité de la population diminue au cours des générations.

Croisement

Le croisement (ou recombinaison) est appliqué une fois que deux individus sont sélectionnés. Deux chromosomes s'échangent des parties de leurs chaînes, pour donner de nouveaux chromosomes pour la génération suivante. Le croisement peut être :

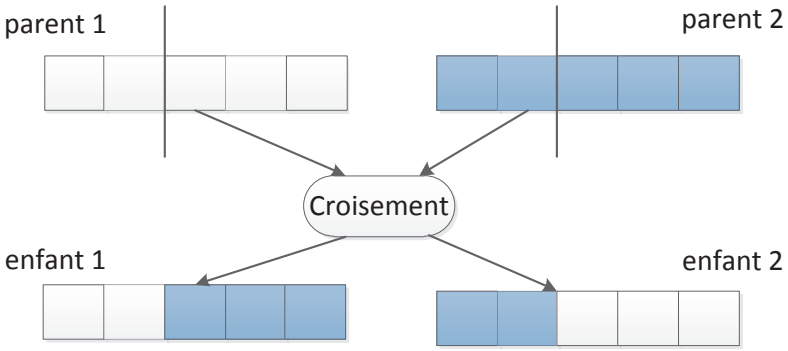

FIGURE 5.4 – Croisement en un point

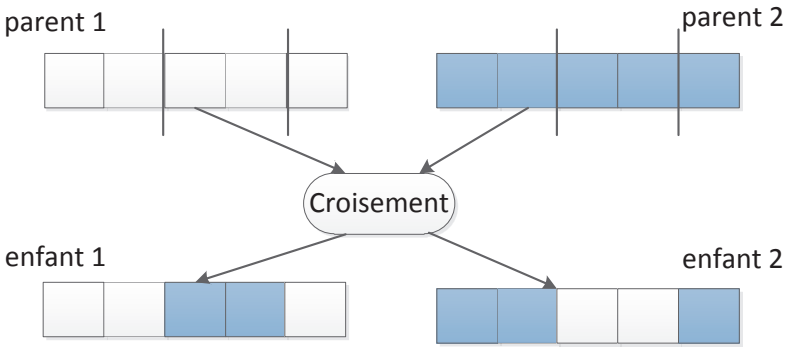

FIGURE 5.5 – Croisement en deux points

- simple : les deux chromosomes sont scindés en deux parties au niveau d'un point (position) choisi aléatoirement. Un nouveau chromosome peut être obtenu à partir de la première partie d'un parent, et de la deuxième partie de l'autre parent. Cela permet en fait de générer un ou deux nouveaux individus (voir la figure 5.4) ;
- multiple : il y a plusieurs points de croisement (voir la figure 5.5).

Nous avons appliqué le croisement simple qui est le plus souvent utilisé dans les algorithmes génétiques. La probabilité P_c que deux chromosomes sélectionnés soient effectivement croisés, est un paramètre de l'algorithme génétique qui dépend du problème. La probabilité d'un croisement est comprise entre 0 et 1 strictement.

Il est possible de faire cette opération en prenant X individus à croiser. On peut donc potentiellement obtenir X nouveaux individus. Notez qu'il n'est pas nécessaire et surtout

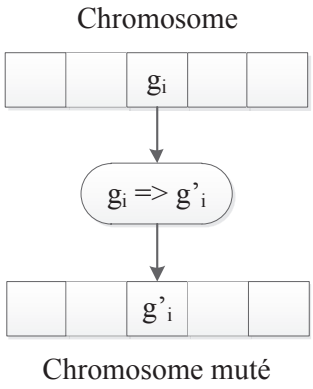

FIGURE 5.6 – Principe de l'opérateur de mutation

pas recommandé de croiser tous les individus d'une population, car rien ne garantit que le résultat d'un croisement sera meilleur que les individus parents.

Mutation

La mutation est une création de nouveaux individus en modifiant ceux déjà existant. Elle apporte aux algorithmes génétiques la propriété d'atteindre tous les points de l'espace d'état, sans pour autant les parcourir tous dans le processus de résolution [Fogel et al., 1996]. L'opérateur de mutation consiste généralement à tirer aléatoirement un gène dans le chromosome et à le remplacer par une valeur aléatoire (voir la figure 5.6). De la même manière que pour le croisement, on définit ici un taux de mutation P_m qui est généralement compris entre $0,001$ et $0,01$. Pour P_m, il est nécessaire de choisir une valeur relativement faible pour ne pas tomber dans une recherche aléatoire et conserver le principe de sélection et d'évolution. Rien ne dit que l'individu muté sera meilleur, mais il apportera des possibilités supplémentaires qui pourraient bien être utiles pour :

- éviter la convergence prématurée vers une solution locale ;
- créer de bonnes solutions. Généralement, on ne modifie qu'un gène pour passer d'une solution à une autre solution de forme similaire mais qui peut avoir une valeur d'adaptation totalement différente.

Élitisme

Lors de la création de la nouvelle population par sélection, croisement et mutation, nous avons une grande chance de perdre le meilleur chromosome. Cela ne garantit pas la convergence de l'algorithme. Pour résoudre ce problème, une stratégie élitiste est appliquée. Premièrement, elle copie le meilleur individu (avec le meilleur *makespan*) dans

T_{11}	T_{12}	T_{21}	T_{22}	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}
R_1	R_1	R_1	R_1	M_1	M_3	M_2	M_1	M_2	M_4

FIGURE 5.7 – Un chromosome possible pour un exemple simple.

la nouvelle population. Les autres sont sélectionnés par la procédure classique : sélection, croisement, mutation. Cela nous permet de garder le meilleur chromosome trouvé et de ne pas dégrader la performance de l'algorithme génétique.

Choix des opérateurs

Les opérateurs que nous avons utilisés sont :
- une sélection par roulette ;
- un croisement en un point, avec une probabilité P_c égale à 0,8 ;
- une mutation par changement d'affectation de ressource, avec une probabilité P_m égale à 0,01 ;
- une stratégie élitiste qui conserve deux individus à chaque génération : le meilleur pour le *makespan* et le meilleur pour le stockage. Ainsi, dans les problèmes sans stock, la meilleure solution en termes de *makespan* peut être choisie parmi les individus qui ont un stockage à zéro.

Nous utilisons l'exemple simple suivant pour illustrer les opérateurs de notre algorithme génétique. Il y a 2 jobs J_1 et J_2, 4 machines $M = M_1, M_2, M_3, M_4$ et un robot. Pour chaque job, on a la succession de tâches suivante :
J1 : O11(M1, M2) → T11 → O12(M3, M4) → T12 → O13(M2, M3)
J2 : O21 (M1, M2) → T21 → O22 (M2, M3) → T22 → O23 (M3, M4)

Un chromosome possible est donné dans la figure 5.7. Dans cette figure, et les suivantes, on indique R_i ou M_i au lieu de la valeur i pour préciser s'il s'agit d'une ressource de transport ou d'une machine.

Le croisement et la mutation sont illustrés par les figures 5.8 et 5.9.

5.2.4 Les fonctions objectifs

Dans notre modèle générique, un objectif est le *makespan* qui est le plus souvent demandé dans les systèmes de production. Pour s'adapter aux divers cas traités, on ajoute un deuxième objectif : le stockage total, qui est le temps d'attente total pour l'ensemble des tâches opératoires, en amont et en aval des ressources de traitement.

D'après les méthodes de séquencement que nous avons proposées, différentes procédures peuvent être utilisées :

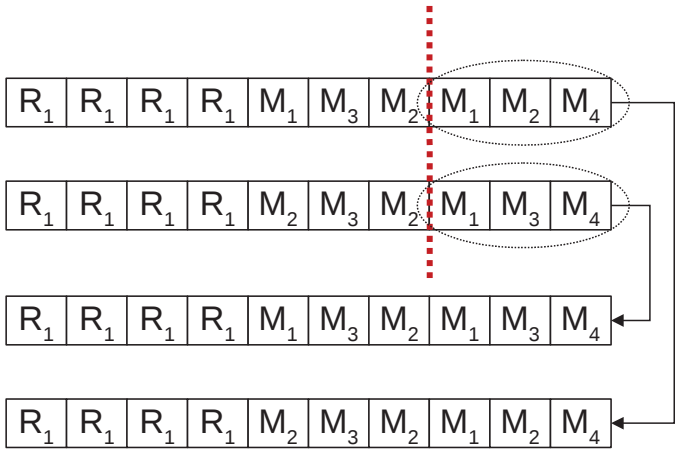

FIGURE 5.8 – Croisement de deux individus.

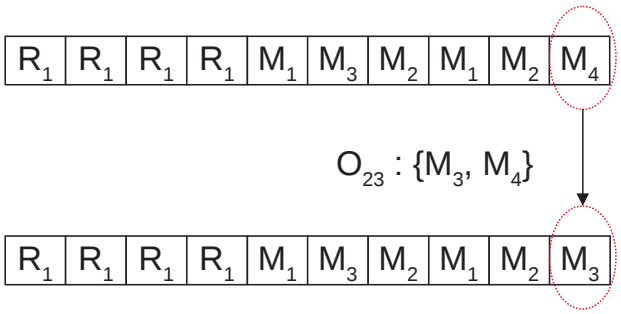

FIGURE 5.9 – Mutation d'un individu.

- si la méthode de séquencement est la recherche Tabou, nous pouvons réduire les fenêtres de temps pour chaque tâche (voir l'algorithme 4 au chapitre 4) ; le *makespan* et le stockage minimal peuvent être déduits ;
- si l'intégration de la méthode de séquencement est la procédure SBN avec réparation, l'évaluation est faite sur la base du graphe générique : l'algorithme de Dantzig permet d'évaluer le premier critère (*makespan*) ainsi que les dates au plus tôt de chaque tâche. Le deuxième critère (stockage) en est déduit (algorithme 12).

Algorithm 12 Évaluation du stockage pour une solution représentée par le graphe générique G

if G n'est pas encore évalué **then**
  applique Dantzig(G)
end if
stockage total $stor = 0$
for chaque nœud d'opération OP_{ij} **do**
  if OP_{ij} n'est pas une opération de chargement ou déchargement **then**
    calculer temps d'attente w_{ij}^{-} entre les dates de début des tâches UST_{ij} et OP_{ij}
    calculer temps d'attente w_{ij}^{+} entre les dates de début des tâches UST_{ij} et OP_{ij}
    $stor = stor + w_{ij}^{-} + w_{ij}^{-}$
  end if
end for
return *stor*

Pour résoudre ce problème multi-objectif (PMO), une approche Pareto peut être utilisée pour classer les individus dans la population. Nous rappelons ici quelques notions et définitions associés à ce type de méthode ([ghazali Talbi, 1999]).

Définition : PMO

Un problème multi-objectif (PMO) est défini par :

$$minF(x) = (f_1(x), f_2(x), ..., f_n(x)), s.c.x \in C \tag{5.1}$$

où $n \geq 2$ est le nombre d'objectifs. x est le vecteur des variables de décision, C représente l'ensemble des solutions faisables et $F(x) = (f_1(x), f_2(x), ..., f_n(x))$ est le vecteur des critères à minimiser.

Définition : relation de dominance

La relation d'ordre partiel sur l'ensemble de points réalisables dans l'espace des critères est appelée relation de dominance. Une solution $y = (y_1, ..., y_n)$ domine une solution $z = (z_1, ..., z_n)$ ssi $\forall i \in [1..n] y_i \leq z_i et \exists i \in [1..n] / y_i < z_i$.

Définition : solution Pareto optimale Une solution $x* \in C$ est Pareto optimale (non dominée) ssi il n'existe pas une solution $x \in C$ telle que $F(x)$ domine $F(x*)$.

Dans une approche Pareto, on classe les individus d'une population en fonction de leur degré de dominance. Les individus qui ont le même classement ont la même valeur d'adaptation (fitness). Cette technique nous permet de conserver à la fois les meilleurs individus pour le *makespan* et les meilleurs individus pour le stockage. [Goldberg, 1989] a utilisé en premier ce type de méthode pour distribuer la même probabilité de reproduction pour les solutions Pareto optimales. Il y a normalement deux étapes :

- Étape 1 : classer les individus dans la population en leur affectant un rang (algorithme 13). On attribue le rang 1 aux solutions non dominées ;
- Étape 2 : distribuer la probabilité de reproduction basée sur les rangs.

Algorithm 13 Procédure de classement des individus dans une population.

$k = 1$.
while Il y a encore des individus qui ne sont pas classés **do**
  Parmi les individus non classés, trouver tous les individus non dominés et leur attribuer le rang k
  $++k$.
end while

La probabilité de reproduction pour un individu I_i est donc distribuée comme suit :

$$p_i = q - (k-1) \times r \tag{5.2}$$

où k est le numéro de classement pour I_i, q est la probabilité du meilleur individu, et r est calculé comme ci-dessous :

$$r = \frac{q - q_0}{K - 1} \tag{5.3}$$

où q_0 est la probabilité du pire individu, et K est le nombre total de rangs dans le classement.

5.3 Procédure globale GATS

5.3.1 Principe

La plupart des publications traitant du *job shop* avec transport considère une affectation des tâches totalement ou partiellement fixe. Peu de recherches traitent à la fois les problèmes d'affectation et de séquencement. Pour cela, nous proposons une méthode hybride en intégrant un algorithme génétique et les heuristiques proposées. Dans ce cadre, l'algorithme génétique fonctionne comme une méthode de niveau supérieur pour gérer le problème d'affectation et faire évoluer la population des solutions, tandis que les heuristiques améliorent le séquencement. L'idée est d'exploiter les avantages des différentes méthodes et de résoudre le problème générique, de même que les cas spécifiques.

Dans cette section, nous couplons dans un premier temps l'algorithme génétique avec la procédure Tabou proposée au chapitre précédent (figure 5.10). Dans la suite, nous désignons cette procédure globale par le nom GATS (pour *Genetic Algorithm and Tabu Search*). Dans celle-ci, les chromosomes (donc les affectations) sont générés aléatoirement parmi les ensembles de ressources possibles. A partir d'un chromosome, le séquencement initial de chaque individu est généré de la manière décrite dans la section 4.2.2. L'étape d'évaluation des solutions se fait par l'algorithme 4. À partir de ces informations initiales, la procédure tabou améliore les séquences, et améliore aussi certaines affectations pour le transport, en échangeant deux tâches sur deux ressources différentes ou en insérant une tâche sur une autre ressource. Ces mouvements peuvent être assimilés à un type de mutation avec la plus haute probabilité d'apparition. Pour éviter de trop ralentir l'évolution globale, cette heuristique intégrée est limitée à un nombre d'itérations de recherche locale. Le meilleur voisin (meilleur *makespan*) exploré à chaque itération remplace la solution courante pour l'étape suivante. Pour les ateliers sans stock, ce voisin est choisi parmi l'ensemble des solutions explorées sans stock, si elles existent. Sinon la solution avec le meilleur *makespan* sera sélectionnée. Pour éviter la convergence locale, nous générons de nouvelles solutions aléatoires pour remplacer les individus qui n'ont pas changé pendant un certain nombre de générations. Il y aura donc plus de diversité dans la population. Mais la meilleure solution trouvée sera toujours gardée par la règle élitiste adoptée.

5.3.2 Tests et résultats

Nos tests sont effectués sur l'ensemble des 4 classes d'instances utilisées au chapitre précédent. Nous y ajoutons une 5ème classe. Ce sont des extensions des instances de la classe 3 (FJSP de [Deroussi et al., 2008]) aux problèmes avec temps bornés. Les temps de traitement fixes de la classe 3 deviennent les temps minimaux de la classe 5. Les temps opératoires maximaux sont de 20% plus longs que les bornes minimales : $P_{ij}^{+} = 1.2 \times P_{ij}^{-}$. Les instances de la classe 5 sont sans doute les plus proches de notre problème de *job shop* générique GFJSSP, c'est à dire avec : une flexibilité des machines et des ressources de transport, des temps opératoires bornés, des emplacements de stockage de capacité infinie. La conjonction du stockage infini et des temps bornés peut avoir des applications pratiques : par exemple elle peut traduire la possibilité d'autoriser une attente limitée sur les machines, dans le but d'éviter une surcharge sur les stocks (ou d'en réduire la capacité) ; également, dans le cas de problèmes de type HSP, on peut envisager des cas avec des processus opératoires chimiques nécessitant des temps bornés, mais avec de l'attente autorisée entre les cuves, notamment si ce type d'attente n'entraîne pas de dégradation des produits (cas de produits pour lesquels il n'y a pas de risque d'oxydation à l'air par exemple).

Le tableau 5.1 est un rappel des caractéristiques des 4 premières classes d'instances du tableau 4.4, que nous avons complété avec la classe 5.

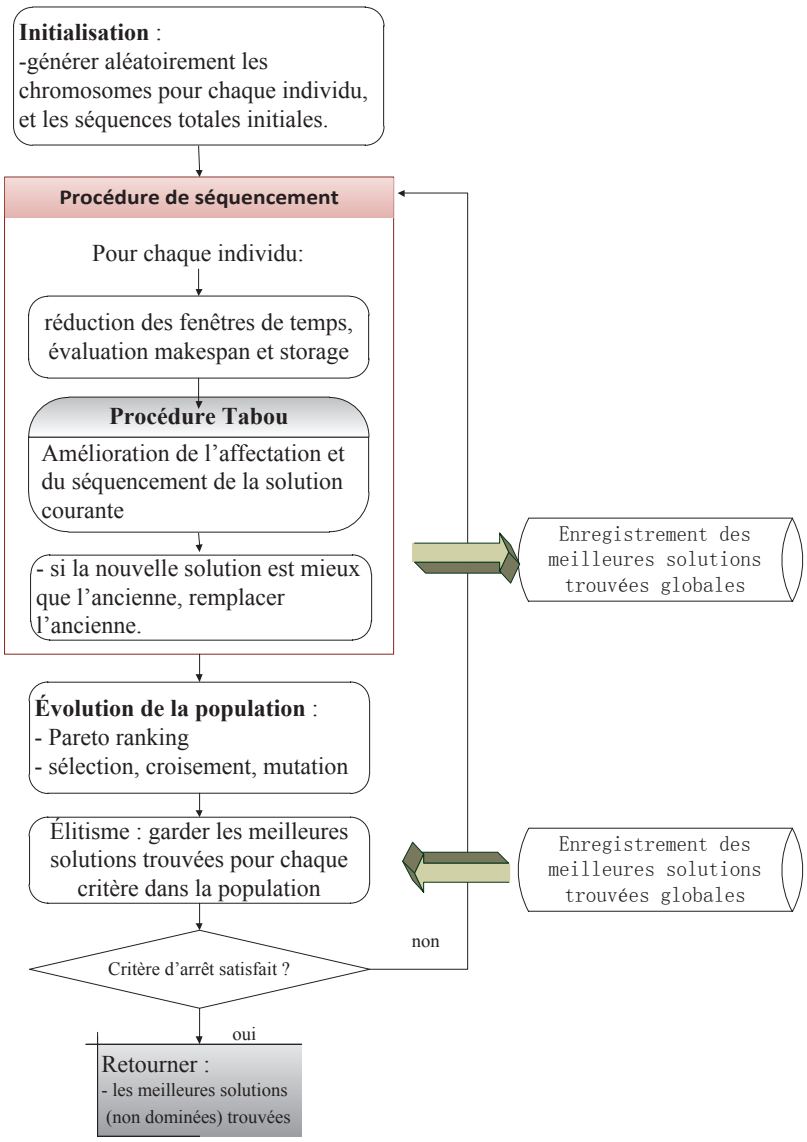

FIGURE 5.10 – Procédure globale proposée avec GA et Tabou.

TABLE 5.1 – Caractéristiques des 5 classes d'instances

Classe	Temps de traitement	stock capa.	attente sur ress.	affectation tâches opé.	affectation tâches transp.	nb. de jobs	nb. de ress. robots	nb. de ress. machines	nb. de traitements	nb. inst.
1	Fixes	∞	Non	Non	Non	6/10	1	6/10	36/100	7/8
2	Fixes	∞	Non	Non	Oui	5 à 8	2	5*	13-21	40
3	Fixes	∞	Non	Oui	Oui	5 à 8	2	10*	13-21	10
4	Bornés	0	Non	Non	Non/oui	5 à 40	1-2	5 à 18 *	35 à 290	209
5	Bornés	∞	Non	Oui	Oui	5 à 8	2	10*	13-21	10

* postes supplémentaires pour le chargement et déchargement à prendre en compte.

Nous donnons les résultats dans les tableaux suivants (tableaux 5.2 à 5.7). Ces résultats sont obtenus pour vingt exécutions de notre algorithme GATS pour chaque instance. Le nombre maximal de générations de l'algorithme génétique est de 2000 avec une taille de population égale à 50 individus. Le nombre maximum d'itérations pour la procédure de recherche tabou est 30. Dans ces tableaux, nous donnons non seulement la meilleure solution pour le critère du *makespan*, mais aussi la meilleure solution pour le temps de stockage minimal (min stor.).

Le tableau 5.2 correspond aux instances P01 de la première classe. Les résultats Ref C^* sont ceux déjà fournis dans le tableau 4.5. Notre algorithme GATS trouve le meilleur résultat pour une instance (colonne Gap1). En moyenne, l'écart avec le *makespan* des solutions de référence est de 4,36% (au lieu de 11,74% pour le Tabou). Nos résultats ne sont pas si mauvais si on considère que nous utilisons une méthode qui n'est pas dédiée à cette classe de problèmes et où seule une partie de l'algorithme est réellement utilisé ici (puisqu'il n'y a aucun problème d'affectation et les temps de traitement sont fixes). Malgré cela, pour toutes les instances de type P01, GATS améliore les résultats de notre Tabou seul (C_{Tab}), avec un gain moyen de 6,46% (Gap2). Pour les instances P02, le *makespan* de GATS est foruni dans le tableau 5.8. L'écart avec la référence est plus important que pour P01, mais le meme type de conclusion peut être fait. Pour l'ensemble des instances de classe 1, l'écart avec les meilleurs résultats est de 22,32% (contre 24,35% pour notre Tabou).

Enfin, la colonne Gap3 donne l'écart entre les meilleurs *makespans* obtenus par GATS, sous les hypothèses avec et sans stock. L'écart varie ici de 5 à 16,87% selon les instances, avec une moyenne de 10,53%. Le fait de ne pas avoir le même C_{max} dans les deux cas n'est pas surprenant. En effet, dans le problème sans attente et sans stock, les opérations de traitement et de transport doivent s'enchaîner de manière continue. Il est possible d'améliorer l'ordonnancement ainsi obtenu si on autorise le stockage, en anticipant notamment certains transports, puisque les produits peuvent attendre dans les *buffers*.

Pour la deuxième catégorie d'instances (tableau 5.3), notre algorithme génétique est utilisé uniquement pour les affectations des tâches de transport, ce qui génère de nombreux chromosomes semblables. Nous comparons les C_{max} obtenus par notre méthode hybride GATS aux meilleurs de la littérature obtenus dans [Abdelmaguid et al., 2004], [Deroussi et al., 2008] et [Larabi, 2010], dont les auteurs ont proposé respectivement une approche par algorithme génétique (GAA), méta-heuristique hybride (SALS), et mémétique (LA). Nous comparons également les performances de GATS et de notre Tabou.
Tout d'abord, nous pouvons observer que GATS améliore les performances du Tabou seul de 4.27% en moyenne sur les 40 instances. Par rapport aux résultats de référence, nous retrouvons le même *makespan* que GAA (respectivement SALS et LA) pour 50% d'instances (respectivement 60% par rapport à SALS et LA). Comparés à GAA, nous

TABLE 5.2 – Résultats pour les instances de classe 1

Inst.	Ref. C^*	Tabou C_{Tab}	GATS pour C_{max} $C_{max}1$	min stor.	Gap1 %	Gap2 %	$C_{max}2$ GATS pour min stor.=0	Gap3 %
P01T3t0	92	93	92	22	0	-1,08	99	7,61
P01D3d1	216	235	220	65	1,85	-6,38	231	5
P01D1t1	81	91	83	50	2,47	-8,79	97	16,87
P01D2d1	148	169	155	76	4,73	-8,28	167	7,74
P01T2t1	74	83	79	55	6,76	-4,82	88	11,39
P01D1d1	87	106	96	57	10,34	-9,43	110	14,58
moyenne					4,36%	-6,46%		10,53%

$Gap1 = (C_{max}1 - C^*)/C^*$.
$Gap2 = (C_{max}1 - C_{Tab})/C_{Tab}$.
$Gap3 = (C_{max}2 - C_{max}1)/C_{max}1$.

avons obtenu de meilleurs résultats pour 15% des cas (amélioration de moins de 2%). Globalement, nous sommes en moyenne respectivement à 1,06%, 1,71% et 1,78% des meilleurs résultats de référence. Ces résultats sont d'autant plus satisfaisants que nous comparons à nouveau notre algorithme générique avec des méthodes plus dédiées. Par ailleurs, pour les méthodes de [Hurink and Knust, 2002] et de [Deroussi et al., 2008], les résultats dépendent en partie des solutions initiales dont la génération n'est pas toujours détaillée dans la littérature. Ainsi l'analyse des différences obtenues est difficile à effectuer.

Le tableau 5.4 compare les résultats de notre méthode GATS pour cette même classe 2, dans les hypothèses initiales de ces instances (avec stock infini), et dans un cas sans stock autorisé. En fait, le problème est résolu pour un stockage infini : la colonne C_1 fournit les meilleurs C_{max} trouvés (déjà donnés dans le tableau précédent), en y ajoutant la valeur du stockage minimal associée. Toutefois, parmi les solutions explorées par GATS, on ne retient que celles ayant un stockage minimal égal à zéro (C_2 pour min stock 0). La dégradation du *makespan* est en moyenne de 3,54%, avec dans certains cas des résultats identiques.

La figure 5.11 représente, pour l'instance Ex14, une population de 200 individus après 200 générations. Les solutions non dominées se trouvent sur la frontière Pareto. Elle contient en particulier la meilleure solution pour chaque critère : une solution meilleure pour le C_{max} mais avec stockage ($C_{max} = 103$ et stockage minimal = 18), et une solution sans stock ($C_{max} = 108$ et stockage minimal = 0).

Rappelons que les résultats ont été obtenus pour un problème avec capacité de stockage infinie. La figure 5.11 montre qu'avec la méthode proposée, et en une seule exécution, nous pouvons effectivement modéliser et résoudre différents types de problèmes, en particulier avec stock infini ou nul, ce qui correspond à notre objectif de prise en compte de plusieurs classes de problèmes. En effet, le graphe des solutions et le front de Pareto obtenu mettent en évidence les types de problèmes associés, par exemple les problèmes

TABLE 5.3 – Résultats pour les instances de classe 2

Inst.	Ref.1	Ref.2		Tabou	GATS	Gaps%			
Inst.		SALS	GAA						
	C_{la}	C_{sa}	C_{ga}	C_{tab}	C_1	g_{la}	g_{sa}	g_{ga}	g_{tab}
Ex11	96	96	96	97	96	0,00	0,00	0,00	-1,03
Ex12	82	82	82	82	82	0,00	0,00	0,00	0,00
Ex13	84	84	84	86	84	0,00	0,00	0,00	-2,33
Ex14	103	103	103	108	103	0,00	0,00	0,00	-4,63
Ex21	100	100	102	107	104	4,00	4,00	1,96	-2,80
Ex22	76	76	76	80	76	0,00	0,00	0,00	-5,00
Ex23	86	86	86	86	86	0,00	0,00	0,00	0,00
Ex24	108	108	108	118	108	0,00	0,00	0,00	-8,47
Ex31	99	99	99	112	99	0,00	0,00	0,00	-11,61
Ex32	85	85	85	85	85	0,00	0,00	0,00	0,00
Ex33	86	86	86	93	86	0,00	0,00	0,00	-7,53
Ex34	111	111	111	123	114	2,70	2,70	2,70	-7,32
Ex41	112	112	112	124	116	3,57	3,57	3,57	-6,45
Ex42	87	87	88	98	91	4,60	4,60	3,41	-7,14
Ex43	89	89	89	101	94	5,62	5,62	5,62	-6,93
Ex44	121	121	126	136	129	6,61	6,61	2,38	-5,15
Ex51	87	87	87	90	88	1,15	1,15	1,15	-2,22
Ex52	69	69	69	69	69	0,00	0,00	0,00	0,00
Ex53	74	74	74	74	74	0,00	0,00	0,00	0,00
Ex54	96	96	96	96	96	0,00	0,00	0,00	0,00
Ex61	118	118	118	124	120	1,69	1,69	1,69	-3,23
Ex62	98	98	98	105	98	0,00	0,00	0,00	-6,67
Ex63	103	103	104	107	103	0,00	0,00	-0,96	-3,74
Ex64	120	120	120	134	120	0,00	0,00	0,00	-10,45
Ex71	111	111	115	124	115	3,60	3,60	0,00	-7,26
Ex72	79	79	79	91	85	7,59	7,59	7,59	-6,59
Ex73	83	83	86	96	93	12,05	12,05	8,14	-3,13
Ex74	126	126	127	141	138	9,52	9,52	8,66	-2,13
Ex81	161	161	161	161	161	0,00	0,00	0,00	0,00
Ex82	151	151	151	151	151	0,00	0,00	0,00	0,00
Ex83	153	153	153	153	153	0,00	0,00	0,00	0,00
Ex84	163	163	163	167	163	0,00	0,00	0,00	-2,40
Ex91	116	116	118	121	116	0,00	0,00	-1,69	-4,13
Ex92	102	102	104	109	102	0,00	0,00	-1,92	-6,42
Ex93	105	105	106	112	105	0,00	0,00	-0,94	-6,25
Ex94	120	120	122	131	121	0,83	0,83	-0,82	-7,63
Ex101	146	147	147	152	148	1,37	0,68	0,68	-2,63
Ex102	135	135	136	144	135	0,00	0,00	-0,74	-6,25
Ex103	137	138	141	148	141	2,92	2,17	0,00	-4,73
Ex104	157	159	159	177	162	3,18	1,89	1,89	-8,47
moyenne						1,78%	1,71%	1,06%	-4,27%
écart type						0,03	0,03	0,03	0,03

Ref. 1 : [Larabi, 2010], Ref. 2 : [Deroussi and Norre, 2010]
$g_{la} = (C_1 - C_{la})/C_{la}$, $g_{sa} = (C_1 - C_{sa})/C_{sa}$
$g_{ga} = (C_1 - C_{ga})/C_{ga}$, $g_{tab} = (C_1 - C_{tab})/C_{tab}$

TABLE 5.4 – Comparaison des résultats de GATS pour la classe 2 avec et sans stock

Inst.	pour C_{max}		min stock 0	Gap
	C_{max} (C_1)	min stor.	C_2	%
Ex11	96	0	96	0
Ex12	82	0	82	0
Ex13	84	14	86	2,38
Ex14	103	8	107	3,88
Ex21	104	0	104	0
Ex22	76	5	79	3,95
Ex23	86	5	87	1,16
Ex24	108	0	108	0
Ex31	99	0	99	0
Ex32	85	10	86	1,18
Ex33	86	0	86	0
Ex34	114	46	116	1,75
Ex41	116	16	122	5,17
Ex42	91	27	98	7,69
Ex43	94	26	108	14,89
Ex44	129	66	138	6,98
Ex51	88	10	97	10,23
Ex52	69	12	83	20,29
Ex53	74	11	81	9,46
Ex54	96	9	99	3,13
Ex61	120	0	120	0
Ex62	98	7	103	5,10
Ex63	103	6	107	3,88
Ex64	120	0	120	0
Ex71	115	0	115	0
Ex72	85	13	90	5,88
Ex73	93	58	99	6,45
Ex74	138	0	138	0
Ex81	161	0	161	0
Ex82	151	0	151	0
Ex83	153	0	153	0
Ex84	163	48	170	4,29
Ex91	116	20	122	5,17
Ex92	102	15	105	2,94
Ex93	105	14	106	0,95
Ex94	121	30	125	3,31
Ex101	148	42	154	4,05
Ex102	135	17	140	3,70
Ex103	141	35	142	0,71
Ex104	162	61	167	3,09
moyenne				3,54%
écart type				4,34

$Gap = (C_2 - C_1)/C_1$

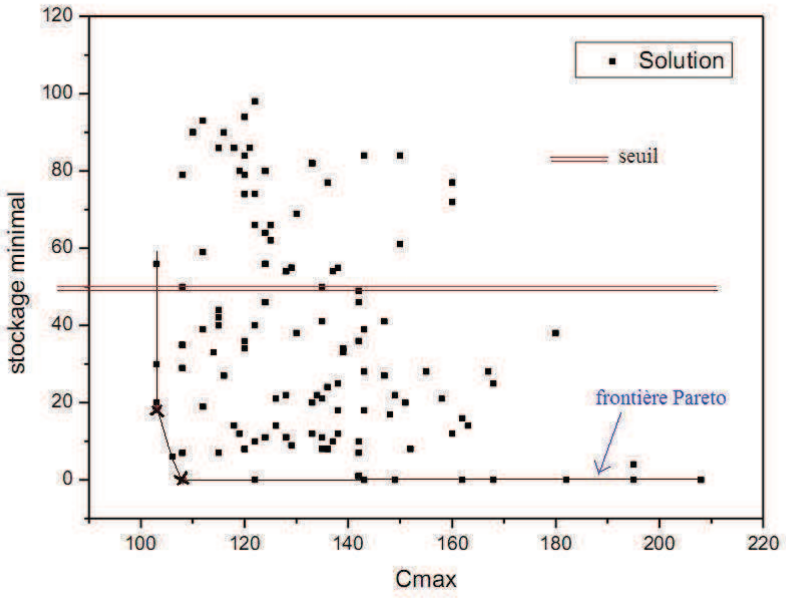

FIGURE 5.11 – Solutions trouvées pour l'instance Ex14

plutôt de type HSP pour les solutions avec stock minimal égal à 0, et ceux de type FMS pour les autres valeurs de ce critère. En termes d'exploitation, ce type de résultats offre plusieurs possibilités intéressantes :

- Tout d'abord, comme nous venons de l'évoquer, et plutôt que de fixer à l'avance des hypothèses sur le stock, nous pouvons résoudre un problème plus général, puis considérer la solution la plus intéressante en termes de compromis (*makespan*, stock). Ce type de résultat peut par ailleurs aussi être utilisé dans un processus de décision visant à dimensionner la capacité du stockage ;
- Pour aller plus loin dans ce raisonnement, et toujours dans le cas de problèmes avec stock illimité : bien que la capacité du stockage ne soit pas a priori un problème, la recherche d'une amélioration globale du système peut s'exprimer en termes d'encours et pas forcément toujours ou uniquement en termes de productivité. Dans ce cas, pour un seuil global de l'encours fixé préalablement, et qui peut se matérialiser sur la figure 5.11 par une droite parallèle à l'axe des abscisses, il suffit de prendre la meilleure solution en termes de C_{max} dans l'espace compris entre les deux axes horizontaux stock minimal=0 et stock minimal=seuil. Le principe de ce raisonnement peut être apparenté à une méthode ε-contrainte, puisque cela revient à optimiser une fonction (ici le C_{max}) sujette à des contraintes sur les autres fonctions (stockage) ;
- Dans le cas de problèmes sans stock et avec temps bornés, qui correspondent ici aux instances de HSP de la classe 4 : le fait d'avoir toutes les solutions, y compris pour le

problème relaxé en termes de stockage, permet de proposer des solutions meilleures pour le C_{max}, mais dégradées au niveau du stock (sous réserve que la contrainte de capacité des machines/cuves est respectée). Pour améliorer la productivité de l'atelier, l'évaluation du stock se traduit en termes d'augmentation nécessaire du temps maximal dans certaines cuves (machines). Il reste alors à vérifier auprès des chimistes que cette modification des paramètres garantit toujours la qualité des produits, moyennant peut-être des modifications des caractéristiques du système physique, telles que la concentration des bains dans les cuves. Dans ce cas, la proposition de solutions alternatives relaxées, si elles s'avèrent viables, peut devenir un vecteur d'amélioration des performances de l'atelier.

Pour la troisième classe d'instances proposée par [Deroussi and Norre, 2010], nous n'avons pas trouvé de résultats associés dans les références pour le critère C_{max}. Mais nous effectuons des comparaisons avec nos méthodes Tabou (pour évaluer l'apport de l'algorithme génétique) et SBN (qui donne ici de meilleurs résultats que le Tabou seul) (tableau 5.5). GATS améliore les résultats pour toutes les instances, en moyenne respectivement de 7,61% et 5,65% par rapport à ces deux méthodes. Notons que ces exemples considèrent à la fois le problème d'affectation des ressources de traitement et des ressources de transport, qui exploitent plus les fonctionnalités de notre algorithme que les classes précédentes. Par ailleurs, le tableau 5.5 montre les valeurs que nous avons obtenues avec GATS pour nos deux objectifs. Dans ce tableau, comme dans les précédents, nous pouvons remarquer que les meilleures solutions avec stock minimal nul ne sont pas toujours obtenues pour le meilleur *makespan*. L'écart est en moyenne de 5,40% (1,38% sans la dernière instance fjsp10). De plus, on note que le stock minimal ne correspond pas toujours à la valeur réelle. Ainsi, une valeur égale à zéro ne signifie pas que le stockage réel est égal à zéro. En effet, l'évaluation du stockage minimal est faite sur la base des dates au plus tôt. Nous évaluons ce stock minimal pour orienter la recherche vers des solutions qui peuvent correspondre à des problèmes sans aucun stock entre les machines. Elle nous permet de limiter l'exploration de l'espace des solutions. Par ailleurs, ce type de problèmes est aussi rencontré dans les ateliers où les temps de traitement sont bornés. En effet, sans stockage entre les machines, si les travaux ne sont pas autorisés à rester sur les machines après que le temps de traitement maximal soit atteint, alors les tâches de traitement doivent être réalisées en continu.
Le tableau 5.5 correspond à la cinquième classe d'instances où les temps de traitement maximaux sont strictement supérieurs aux temps minimaux. Nous avons trouvé la même valeur de *makespan* (colonne C_1) que dans le tableau 5.5 de la classe 3 avec temps fixes. C'est normal car notre calcul du *makespan* ne dépend pas du temps de traitement maximal. Mais si nous regardons l'équation 4.3 de notre modèle, le temps d'attente aux emplacements de stockage diminue lorsque le temps de traitement maximal augmente. En effet, dans le cas de temps fixes, dès qu'une tâche de traitement est terminée, le produit associé est placé dans l'emplacement de stockage en aval de la machine. Par contre, si la

TABLE 5.5 – Résultats pour les instances de classe 3 avec critère = *makespan*

	Tabou	SBN	GATS			Gaps%		
				min	min stock=0			
Inst.	C_{tab}	C_{sbn}	C_1	stock	C_2	g_{tab}	g_{sbn}	g_s
fjsp1	160	156	144	8	146	-10,00	-7,69	1,39
fjsp2	128	124	118	18	122	-7,81	-4,84	3,39
fjsp3	162	140	124	0	124	-23,46	-11,43	0,00
fjsp4	126	132	124	16	126	-1,59	-6,06	1,61
fjsp5	100	96	94	0	94	-6,00	-2,08	0,00
fjsp6	152	148	144	10	146	-5,26	-2,70	1,39
fjsp7	132	132	124	22	126	-6,06	-6,06	1,61
fjsp8	188	191	180	8	183	-4,26	-5,76	1,67
fjsp9	162	154	150	10	152	-7,41	-2,60	1,33
fjsp10	186	192	178	30	252	-4,30	-7,29	41,57
moyenne gap						-7,61%	-5,65%	5,40%
écart type gap						0,06	0,03	0,13

$g_{tab} = (C_1 - C_{tab})/C_{tab}$, $g_{sbn} = (C_1 - C_{sbn})/C_{sbn}$
$g_s = (C_2 - C_1)/C_1$

durée opératoire est bornée (avec $P_{ij}^+ > P_{ij}^-$), et quand la borne minimale est atteinte, la pièce reste encore sur la machine jusqu'à ce que le temps de traitement maximal P_{ij}^+ soit atteint, avant d'être transféré dans le stock aval, si aucune ressource de transport n'est disponible. Cela explique la diminution de la valeur du second critère (min stock) entre les deux tableaux. Suivant le même principe, il est à noter que toute solution sans stockage avec $P_{ij}^+ = P_{ij}^-$ est aussi solution dans le cas où $P_{ij}^+ = 1,2 \times P_{ij}^-$, mais que dans certains cas (fjsp4 et fjsp10), on arrive à améliorer encore la valeur du *makespan* (voir colonnes min stock=0 des deux tableaux).

Le tableau 5.7 présente les résultats de la 4ème classe, correspondant aux instances de HSP de ([Mateo et al., 2002]). Pour les solutions avec stockage minimal égal à zéro, nous avons appliqué la procédure de vérification (décrite dans la section 4.2.3) pour vérifier leur faisabilité. Parmi toutes les solutions recherchées, peu d'entre elles correspondent à un stockage minimal à zéro (colonne C_2), et encore moins à un stock réel à zéro (colonne C_3). Ainsi, dans les deux dernières colonnes du tableau 5.7, nous comparons nos meilleurs C_{max} sans stock (C_3, obtenus avec GATS) avec les bornes LB et UB définies au chapitre précédent. Pour 40% des instances, les écarts g_{min} et g_{max} sont tous négatifs, ce qui veut dire que nos C_{max} sont meilleurs que LB. Donc nous sommes sûrs que nous obtenons les meilleurs résultats. Pour 60% des cas, $LB < C_{max} < UB$. Dans ces cas, nous avons une bonne chance d'avoir les meilleurs résultats, si l'écart g_{min} est beaucoup plus petit que g_{max} (C_{max} est plus proche de LB que UB), ce qui est le plus souvent vrai pour ces instances. En conclusion, notre algorithme GATS donne des résultats satisfaisants pour ces instances de HSP très contraintes. Mais, comme le Tabou seul, GATS trouve ses

TABLE 5.6 – Résultats pour les instances de classe 5 (avec $P_{ij} = 1,2 \times P_{ij}$)

Inst.	sol. pour C_1	C_{max} stock	min stor=0 C_2	gap%
fjsp1	144	4	146	1,39
fjsp2	118	15	122	3,39
fjsp3	124	0	124	0,00
fjsp4	124	0	124	0,00
fjsp5	94	0	94	0,00
fjsp6	144	10	146	1,39
fjsp7	124	16	126	1,61
fjsp8	180	7	183	1,67
fjsp9	150	10	152	1,33
fjsp10	178	0	178	0,00
moyenne				1,08%
écartype				0,01

limites dès lors que la taille de ce type d'instances augmente. Cela explique que nous ne donnions pas de résultats pour les autres instances de classe 4.

5.3.3 Conclusion pour GATS

Pour la classe 1, tous les individus ont le même chromosome. Donc, l'amélioration ne peut pas venir de l'apport des algorithmes génétiques. Seule change la séquence totale associée à chaque individu. Donc, nous obtenons de meilleures solutions avec GATS qu'avec notre Tabou seul, car dans cette procédure nous avons exploré plus de séquences que dans la méthode Tabou. Pour les instances des classes 2 et 3, les apports des algorithmes génétiques sont mieux exploités. En effet, dans ces deux classes, il est possible de changer les affectations des robots, voire des machines pour la classe 3. Les individus ont alors des chromosomes différents et il est donc possible de bénéficier des effets du croisement et de la mutation. Maintenant, nous remarquons que l'amélioration est plus forte sur la classe où nous pouvons modifier l'affectation à la fois des ressources de traitement et de transport. En effet, si on considère une instance où il est possible uniquement de modifier l'affectation des ressources de transport (cas 1 avec Tabou), et la même instance où il est possible de modifier l'affectation des ressources de traitement et du transport (cas 2 avec GATS), nous pouvons dire que toute solution du cas 1 est une solution du cas 2. Donc comme le cas 2 offre un degré de liberté supplémentaire (dû à la flexibilité des machines), nous pouvons en espérer de meilleurs résultats. Les résultats des tableaux précédents vont dans ce sens.

Il est à noter ici que les tests réalisés sur les instances de la nouvelle classe 5 montrent que notre méthode GATS peut effectivement résoudre des problèmes les plus généraux

TABLE 5.7 – Résultats pour les instances de classe 4 de [Mateo et al., 2002], avec 5 *jobs*, 5 cuves et 1 robot

Inst.	Ref. T	LB	UB	meilleure sol. GATS pour Cmax C_1	min stor.	min stock 0 C_2	stock 0 for C_3	Gap (%) g_{min}	g_{max}
501	368	1106	1509	692	189	783	1099	-0.63	-27.17
502	319	982	1357	639	146	723	914	-6.92	-32.65
503	489	1294	1605	695	203	827	1422	9.89	-11.40
504	347	989	1309	565	217	633	924	-6.57	-29.41
505	405	1144	1483	686	306	813	1262	10.31	-14.90
506	370	1085	1468	622	228	733	1040	-4.15	-29.16
507	466	1248	1559	712	179	795	1315	5.37	-15.65
508	333	961	1281	620	284	702	1018	5.93	-20.53
509	338	971	1291	672	86	719	1082	11.43	-16.19
510	373	1029	1299	578	110	672	1028	-0.10	-20.86
511	432	1246	1632	721	436	847	1131	-9.23	-30.70
512	260	862	1227	666	140	713	949	10.09	-22.66
513	376	1096	1447	668	342	771	1151	5.02	-20.46
514	377	1110	1497	694	265	785	1025	-7.66	-31.53
515	325	903	1154	588	323	686	987	9.30	-14.47
516	494	1274	1501	840	58	894	1486	16.64	-1.00
517	384	1110	1436	697	174	800	1145	3.15	-20.26
518	368	1092	1479	697	212	767	1103	1.01	-25.42
519	421	1224	1610	778	198	866	1245	1.72	-22.67
520	325	903	1154	552	101	627	972	7.64	-15.77
521	392	1135	1469	712	300	788	1106	-2.56	-24.71
522	594	1710	2226	1026	349	1173	1622	-5.15	-27.13
523	576	1559	1892	1026	184	1155	1630	4.55	-13.85
524	323	992	1339	632	165	765	1011	1.92	-24.50
525	546	1554	1995	912	308	1045	1616	3.99	-19.00
526	512	1486	1927	989	56	1057	1438	-3.23	-25.38
527	388	1127	1461	723	300	828	1111	-1.42	-23.96
528	331	1008	1355	707	212	802	1026	1.79	-24.28
529	383	1156	1546	741	305	822	1119	-3.20	-27.62
530	558	1523	1856	1025	112	1159	1579	3.68	-14.92
moyenne								2,09%	-21,61%

$LB = 2 * T + \sum_{j\in[1,O_1-1],k,k'\in[1,r]}(PJ_{1jk} \times PJ_{1j+1k'} \times \sigma_{kk'}) + \sum_{j\in[1,O_1-1]} P^-_{1j}$,

$UB = 2 * T + \sum_{j\in[1,O_1-1],k,k'\in[1,r]}(PJ_{1jk} \times PJ_{1j+1k'} \times \sigma_{kk'}) + \sum_{j\in[1,O_1-1]} P^+_{1j}$.
$g_{min} = (C_3 - LB)/LB$, $g_{max} = (C_3 - UB)/UB$.

de GFJSSP. L'inconvénient est toutefois que nous n'avons pas de résultats de référence auxquels nous comparer.

Pour conclure, bien que l'algorithme génétique ait ici permis d'améliorer les résultats obtenus avec le Tabou seul, l'inconvénient déjà identifié pour le Tabou n'a pas complètement disparu. Il s'agit ici du problème lié à la faisabilité des solutions obtenues en un temps limité dans les cas les plus contraints (sans stock avec temps bornés pour le HSP) et pour les instances de plus grande taille (par exemple avec 21 ou 40 *jobs* pour la classe 4). Pour ces cas où peu de solutions faisables existent dans l'espace de recherche, l'algorithme génétique, a certes augmenté le nombre de solutions explorées, mais de manière insuffisante. Pour remédier à ce problème, nous avons donc pensé utiliser la procédure de Shifting Bottleneck améliorée que nous avons proposée au chapitre précédent. L'objectif est de combiner les avantages des diverses méthodes, en espérant que l'une pallie les défauts de l'autre et vice versa. Ainsi le SBN devrait augmenter la probabilité d'obtenir des solutions faisables dans la procédure GATS. Inversement, le GATS devrait pallier le fait que le SBN s'arrête dès qu'une solution faisable est trouvée, en offrant la possibilité d'améliorer simultanément plusieurs solutions initiales faisables.

5.4 Procédure globale GTSB

5.4.1 Principe

La figure 5.12 montre une seconde proposition de procédure globale : elle intègre la procédure *shifting bottleneck* à notre algorithme GATS combinant l'algorithme génétique et la recherche Tabou. Dans la suite, nous désignons cette procédure globale par le nom GTSB (pour *Genetic algorithm, Tabu search and Shifting Bottleneck*).

Dans l'étape d'initialisation, les chromosomes sont générés aléatoirement. Pour chacun d'eux, la procédure SBN est ensuite appliquée pour chercher une solution initiale faisable. Elle peut changer les affectations initiales des tâches de transport. A chaque génération de l'algorithme génétique, la recherche Tabou est utilisée pour améliorer la solution courante pour chaque individu pendant un nombre donné d'itérations. Le Tabou peut changer la séquence courante ainsi que l'affectation des tâches de transport. Au cours des générations successives, la population évolue par application des différents opérateurs. En outre, pour éviter de tomber dans un optimum local, nous utilisons à nouveau notre procédure SBN pour offrir une chance de réinitialisation aux individus qui n'ont pas été améliorés pendant un certain nombre de générations.
La manière dont nous avons intégré la procédure SBN dans l'algorithme GATS s'explique par :

– la nécessité de fournir une solution initiale faisable au Tabou ;

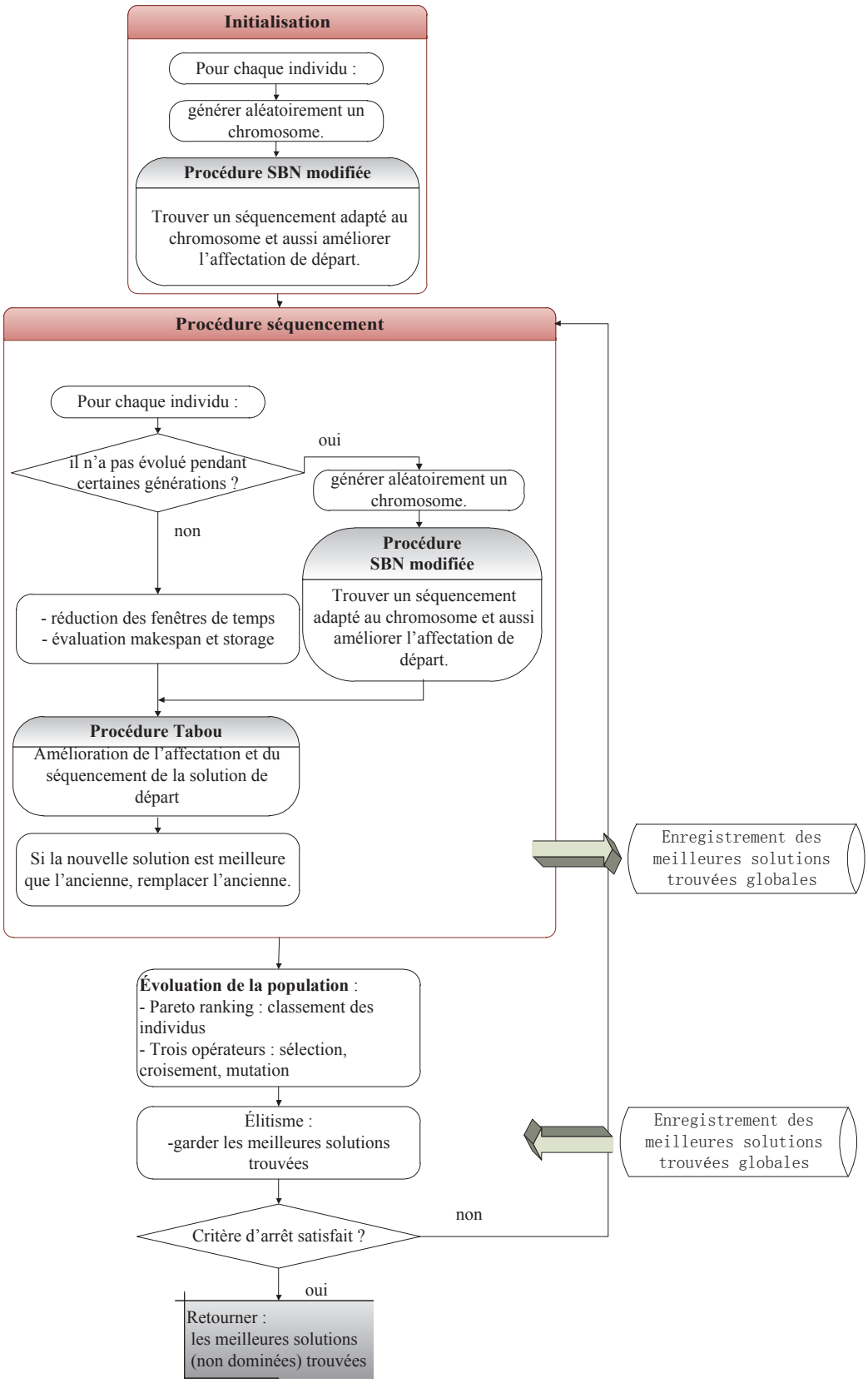

FIGURE 5.12 – Procédure globale proposée avec GA, SBN et Tabou.

– le fait que la complexité de notre heuristique SBN ne nous permet pas de l'appliquer à chaque individu au cours du processus d'évolution.

5.4.2 Tests et résultats

Nous avons comparé les *makespans* de la procédure GA avec SBN et Tabou (notamment GTSB) avec les résultats de référence, s'ils existent, ainsi qu'avec GATS et/ou nos deux heuristiques tabou et SBN) (voir tableaux 5.8 à 5.15. Ces résultats sont obtenus pour dix exécutions de notre algorithme, avec une taille de population de 50, un nombre maximal de générations de 1000, et un nombre d'itérations maximal de 50 pour le Tabou.

Pour les instances de classe 1 (tableau 5.8), GTSB n'est pas plus performant que les méthodes de référence : pour les six premières instances P01, nous relevons un écart variant entre 1,85% et 12,64%, avec un écart moyen de 7,13% (contre 4,36% avec GATS), et un écart moyen global de 25,43% pour l'ensemble des instances de classe 1. Notons que notre méthode est utilisée partiellement pour ce type d'instances sans problème d'affectation, et avec des temps de traitement fixes. Comparé avec nos méthodes SBN et Tabou, GTSB présente une amélioration moyenne de 5,45% par rapport à SBN, avec des performances moyennes équivalentes au Tabou (écart moyen de 0,63%). Mais il est moins efficace que GATS, au moins sur les plus petites instances P01, tout en améliorant certains résultats de GATS pour P02 (écart global moyen de 2,66%).

Pour la classe 2 (tableau 5.9), GTSB a retrouvé les meilleurs résultats connus pour 65% instances. L'écart moyen est de l'ordre de 1%. GTSB est meilleur que SBN et Tabou, avec une amélioration moyenne de l'ordre de 5%. Il est aussi un peu plus performant que GATS (de l'ordre de 0,64%). Comme pour GATS, ces résultats sont assez satisfaisants étant comparés à ceux de méthodes de référence dédiés à une seule catégorie de cas.

Les résultats des instances de classe 3 sont donnés dans les tableaux 5.10 et 5.11. Pour le critère de la date de sortie, l'écart moyen est de 3,12%, comparé avec la référence. GTSB a amélioré le résultat d'une instance (gain 5,36%). Pour cette dernière (fjsp5), nous avons trouvé plusieurs meilleures solutions. Une de ces solutions est présentée sous forme de diagramme de Gantt dans la figure 5.13. Dans cette figure, M_0 est le poste de chargement. Les tâches OP_{i0} ($i \in [1, n]$) correspondent aux tâches fictives de chargement avec des temps de traitement égaux à zéro.

Plusieurs solutions avec la même date de sortie 106 ont été trouvées : elles correspondent à des permutations de tâches sur les paires de machines M_1 et M_2, M_3 et M_4, M_5 et M_6, M_7 et M_8, et aussi à des réaffectations des tâches opératoires. Par exemple, la tâche OP_{52} peut aussi être affectée à M_6, sans dégrader la solution.

Comparé avec SBN et Tabou, les améliorations moyennes reportées dans les tableaux 5.10 et 5.11 sont de 2,96% et 8,48% pour le critère de date de sortie, et de 5,88% et 7,83% pour le critère de *makespan*. Par ailleurs, pour ce dernier critère, GTSB peut être à nouveau considéré comme équivalent à GATS (avec une amélioration moyenne de 0,23%, et en donnant les mêmes résultats pour 80% de cas).

TABLE 5.8 – Résultats pour les instances de classe 1

Inst.	Ref.	C_{SBN}	C_{Tabou}	C_{GATS}	C_{GTSB}	Gaps %				écartype
						Gap_1	Gap_2	Gap_3	Gap_4	GTSB
P01D1d1	87	156	106	96	98	12,64	-37,18	-7,55	2,08	4,22
P01D1t1	81	93	91	83	89	9,88	-4,30	-2,20	7,23	1,70
P01T2t1	74	93	83	79	81	9,46	-12,90	-2,41	2,53	1,20
P01T3t0	92	95	93	92	94	2,17	-1,05	1,08	2,17	0,71
P01D3d1	216	224	235	220	220	1,85	-1,79	-6,38	0,00	1,03
P01D2d1	148	158	169	155	158	6,76	0,00	-6,51	1,94	0,52
P02D1d1	1012	1454	1344	1339	1519	50,10	4,47	13,02	13,44	6,20
P02D1t0	989	1430	1283	1352	1421	43,68	-0,63	10,76	5,10	12,23
P02D1t1	983	1490	1382	1337	1332	35,50	-10,60	-3,62	-0,37	16,70
P02D2d1	1004	1576	1380	1445	1452	44,62	-7,87	5,22	0,48	17,46
P02D3d1	1078	1567	1512	1516	1519	40,91	-3,06	0,46	0,23	13,18
P02D5t2	1361	1576	1683	1689	1694	24,47	7,49	0,65	0,33	12,82
P02T1t1	978	1376	1271	1322	1372	40,29	-0,29	7,95	3,82	19,32
P02T2t1	993	1394	1280	1279	1278	28,70	-8,32	-0,16	-0,08	22,70
P02T5t2	1022	1413	1344	1339	1333	30,43	-5,66	-0,82	-0,41	19,81
moyenne						25,43%	-5,45%	0,63%	2,57%	9,99

$Gap_1 = (C_{GTSB} - Ref)/Ref$, $Gap_2 = (C_{GTSB} - C_{SBN})/C_{SBN}$
$Gap_3 = (C_{GTSB} - C_{Tabou})/C_{Tabou}$, $Gap_4 = (C_{GTSB} - C_{GATS})/C_{GATS}$
Ref : [Hurink and Knust, 2005]
Ref* : [Lacomme et al., 2010]

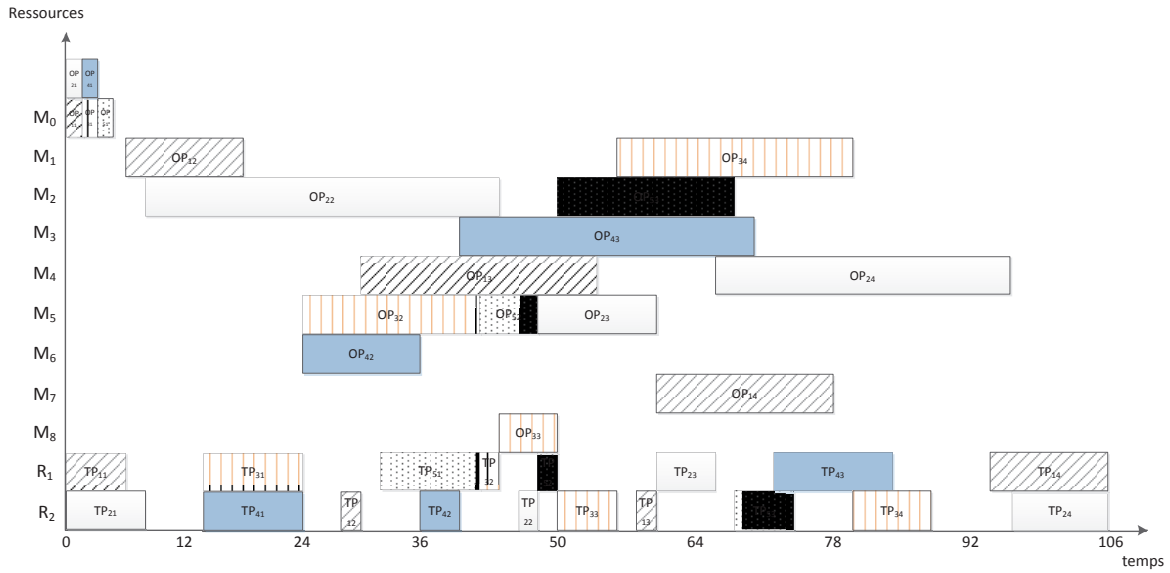

FIGURE 5.13 – Diagramme de Gantt d'une solution trouvée pour l'instance fjsp5

TABLE 5.9 – Résultats pour les instances de classe 2

Inst.	Ref.	C_{SBN}	C_{Tabou}	C_{GATS}	C_{GTSB}	Gaps %				écartype
						Gap_1	Gap_2	Gap_3	Gap_4	GTSB
Ex11	96	97	97	96	96	0,00	-1,03	-1,03	0,00	0,00
Ex12	82	82	82	82	82	0,00	0,00	0,00	0,00	0,00
Ex13	84	88	86	84	84	0,00	-4,55	-2,33	0,00	0,00
Ex14	103	110	108	103	103	0,00	-6,36	-4,63	0,00	0,00
Ex21	100	109	107	104	103	3,00	-5,50	-3,74	-0,96	0,71
Ex22	76	80	80	76	76	0,00	-5,00	-5,00	0,00	0,00
Ex23	86	87	86	86	86	0,00	-1,15	0,00	0,00	0,00
Ex24	108	126	118	108	108	0,00	-14,29	-8,47	0,00	2,67
Ex31	99	110	112	99	99	0,00	-10,00	-11,61	0,00	2,8
Ex32	85	87	85	85	85	0,00	-2,30	0,00	0,00	0,00
Ex33	86	89	93	86	86	0,00	-3,37	-7,53	0,00	0,00
Ex34	111	136	123	114	115	3,60	-15,44	-6,50	0,88	1,05
Ex41	112	131	124	116	114	1,79	-12,98	-8,06	-1,72	2,08
Ex42	87	96	98	91	90	3,45	-6,25	-8,16	-1,10	1,15
Ex43	89	99	101	94	89	0,00	-10,10	-11,88	-5,32	1,78
Ex44	121	132	136	129	129	6,61	-2,27	-5,15	0,00	1,66
Ex51	87	90	90	88	87	0,00	-3,33	-3,33	-1,14	0,48
Ex52	69	73	69	69	69	0,00	-5,48	0,00	0,00	0,00
Ex53	74	76	76	74	74	0,00	-2,63	-2,63	0,00	0,00
Ex54	96	99	96	96	96	0,00	-3,03	0,00	0,00	0,00
Ex61	118	123	124	120	122	3,39	-0,81	-1,61	1,67	0,00
Ex62	98	104	105	98	98	0,00	-5,77	-6,67	0,00	0,63
Ex63	103	106	107	103	103	0,00	-2,83	-3,74	0,00	0,00
Ex64	120	140	134	120	126	5,00	-10,00	-5,97	5,00	1,58
Ex71	111	122	124	115	114	2,70	-6,56	-8,06	-0,87	2,12
Ex72	79	86	91	85	79	0,00	-8,14	-13,19	-7,06	1,69
Ex73	83	91	96	93	85	2,41	-6,59	-11,46	-8,60	1,52
Ex74	126	149	141	138	130	3,17	-12,75	-7,80	-5,80	2,83
Ex81	161	161	161	161	161	0,00	0,00	0,00	0,00	0,00
Ex82	151	151	151	151	151	0,00	0,00	0,00	0,00	0,00
Ex83	153	153	153	153	153	0,00	0,00	0,00	0,00	0,00
Ex84	163	172	167	163	163	0,00	-5,23	-2,40	0,00	0,00
Ex91	116	123	121	116	116	0,00	-5,69	-4,13	0,00	0,97
Ex92	102	107	109	102	102	0,00	-4,67	-6,42	0,00	0,84
Ex93	105	107	112	105	105	0,00	-1,87	-6,25	0,00	0,48
Ex94	120	125	131	121	122	1,67	-2,40	-6,87	0,83	0,32
Ex101	146	163	152	148	148	1,37	-9,20	-2,63	0,00	1,29
Ex102	135	147	144	135	135	0,00	-8,16	-6,25	0,00	1,51
Ex103	137	149	148	141	139	1,46	-6,71	-6,08	-1,42	2
Ex104	157	189	177	162	162	3,18	-14,29	-8,47	0,00	2,15
moyenne						1,07%	-5,67%	-4,95%	-0,64%	0,86

$Gap_1 = (C_{GTSB} - Ref)/Ref$, $Gap_2 = (C_{GTSB} - C_{SBN})/C_{SBN}$
$Gap_3 = (C_{GTSB} - C_{Tabou})/C_{Tabou}$, $Gap_4 = (C_{GTSB} - C_{GATS})/C_{GATS}$
Ref : [Larabi, 2010]

TABLE 5.10 – Résultats pour les instances de classe 3 avec critère = date de sortie

Inst.	Ref.	C_{SBN}	C_{Tabou}	C_{GTSB}	Gaps %			écartype
					Gap_1	Gap_2	Gap_3	
fjsp1	160	176	188	170	6,25	-3,41	-9,57	2,31
fjsp2	138	138	154	138	0,00	0,00	-10,39	1,58
fjsp3	142	152	164	148	4,23	-2,63	-9,76	1,65
fjsp4	138	148	160	146	5,80	-1,35	-8,75	1,65
fjsp5	112	116	122	106	-5,36	-8,62	-13,11	2,35
fjsp6	158	166	178	164	3,80	-1,20	-7,87	1,70
fjsp7	150	166	174	160	6,67	-3,61	-8,05	1,89
fjsp8	197	204	208	197	0,00	-3,43	-5,29	2,06
fjsp9	166	176	180	170	2,41	-3,41	-5,56	1,35
fjsp10	188	206	216	202	7,45	-1,94	-6,48	1,89
moyenne					3,12%	-2,96%	-8,48%	1,84

$Gap_1 = (C_{GTSB} - Ref)/Ref$, $Gap_2 = (C_{GTSB} - C_{SBN})/C_{SBN}$
$Gap_3 = (C_{GTSB} - C_{Tabou})/C_{Tabou}$

TABLE 5.11 – Résultats pour les instances de classe 3 avec critère = *makespan*

Inst.	C_{GATS}	C_{SBN}	C_{Tabou}	C_{GTSB}	Gaps %			écartype
					Gap_1	Gap_2	Gap_3	
fjsp1	144	156	160	146	1,39	-6,41	-8,75	1,07
fjsp2	118	124	128	118	0,00	-4,84	-7,81	1,14
fjsp3	124	140	162	124	0,00	-11,43	-23,46	2,50
fjsp4	124	132	126	124	0,00	-6,06	-1,59	1,75
fjsp5	94	96	100	94	0,00	-2,08	-6,00	0,97
fjsp6	144	148	152	144	0,00	-2,70	-5,26	1,70
fjsp7	124	132	132	122	-1,61	-7,58	-7,58	1,35
fjsp8	180	191	188	181	0,56	-5,24	-3,72	1,83
fjsp9	150	154	162	146	-2,67	-5,19	-9,88	1,58
fjsp10	178	192	186	178	0,00	-7,29	-4,30	2,49
moyenne					-0,23%	-5,88%	-7,83%	1,64

$Gap_1 = (C_{GTSB} - C_{GATS})/C_{GATS}$, $Gap_2 = (C_{GTSB} - C_{SBN})/C_{SBN}$
$Gap_3 = (C_{GTSB} - C_{Tabou})/C_{Tabou}$

Les tableaux 5.12 et 5.13 comparent les résultats pour les instances de classe 4 (instances de [Mateo et al., 2002] avec 5 cuves, 5 jobs, 1 et 2 robots). Pour un seul robot, GTSB est meilleur que LB (borne inférieure pour le temps de 2-cycle) pour 83% des instances. Pour les autres résultats, s'ils sont plus proches de LB que de UB, il y a plus de chance d'avoir de meilleurs résultats que la référence. C'est le cas pour trois instances. GTSB est meilleur que SBN et GATS pour 1 robot (gains respectifs de 25,73% et 0,82%). Enfin comme nous n'avons pas de résultats de référence avec 2 robots, nous comparons GTSB uniquement avec SBN et Tabou. Les améliorations observées sont de 4,93% et 7,85% respectivement.

Nous avons aussi testé les autres instances de Mateo avec 6 à 10 cuves et 21 *jobs*, avec 1 et 2 robots. Les résultats sont montrés dans les tableaux I.1 à I.5 en annexe.

Enfin, pour les instances de classe 4 de [Paul et al., 2007], GTSB améliore légèrement les résultats de référence de 6,56%, ainsi que ceux obtenus par SBN (tableaux 5.14 et 5.15).

Les résultats de la classe 5 entre GATS et GTSB sont plus difficiles à analyser (tableau 5.16. En effet, GTSB améliore le makespan de GATS en moyenne de 0,48%. Par contre, avec GATS on évalue un stock minimal tandis que GTSB donne la valeur réelle du stock. L'écart de 19,96% du tableau n'est donc en fait pas significatif.

5.4.3 Conclusion pour GTSB

Les tests réalisés montrent que GTSB n'est pas forcément plus performant que GATS pour les instances de *job shop* avec temps fixes, stock infini et avec peu ou pas de flexibilité machines et ressources de transport. Toutefois des améliorations ont pu être observées. En outre et surtout, des résultats ont pu être obtenus pour des instances de taille plus importantes (Mateo à 21 *jobs* et Paul à 40 *jobs*). Au final, GTSB semble le meilleur compromis pour les différentes instances prises dans leur globalité. GTSB présente par ailleurs le même avantage que GATS pour la recherche d'un compromis entre les deux objectifs considérés, dans une démarche générale d'exploitation et/ou de conception de l'atelier. Le tableau 5.17 donne un classement de nos 4 méthodes pour les 5 classes d'instances. Il montre que GTSB est globalement plus performant que les 3 autres méthodes, pour la majorité des instances traitées.

5.5 Bilan

En combinant les deux heuristiques de séquencement présentées au chapitre précédent avec un algorithme génétique dédié à l'affectation des ressources, nous avons pu résoudre de manière satisfaisante des instances de différentes classes de problèmes d'ordonnancement d'ateliers avec ressources de transport, incluses dans le problème générique GFJSSP

TABLE 5.12 – Résultats pour les instances de classe 4 avec 5 cuves et 1 robot

Inst.	Ref. bornes		C_{SBN}	C_{GATS}	C_{GTSB}	Gaps %				écartype
	LB	UB				Gap_1	Gap_2	Gap_3	Gap_4	
501	1106	1509	1120	1099	1115	0,81	-26,11	-0,45	1,46	0
502	982	1357	962	914	906	-7,74	-33,24	-5,82	-0,88	0
503	1294	1605	1232	1422	1207	-6,72	-24,80	-2,03	-15,12	0
504	989	1309	943	924	937	-5,26	-28,42	-0,64	1,41	0
505	1144	1483	1105	1262	1105	-3,41	-25,49	0,00	-12,44	0
506	1085	1468	1083	1040	1059	-2,40	-27,86	-2,22	1,83	0
507	1248	1559	1233	1315	1240	-0,64	-20,46	0,57	-5,70	0
508	961	1281	890	1018	910	-5,31	-28,96	2,25	-10,61	0
509	971	1291	914	1082	960	-1,13	-25,64	5,03	-11,28	0
510	1029	1299	949	1028	949	-7,77	-26,94	0,00	-7,68	0
511	1246	1632	1107	1131	1107	-11,16	-32,17	0,00	-2,12	0
512	862	1227	837	949	837	-2,90	-31,78	0,00	-11,80	0
513	1096	1447	1022	1151	1022	-6,75	-29,37	0,00	-11,21	0
514	1110	1497	1046	1025	1046	-5,77	-30,13	0,00	2,05	0
515	903	1154	943	987	943	4,43	-18,28	0,00	-4,46	0
516	1274	1501	1351	1486	1314	3,14	-12,46	-2,74	-11,57	0
517	1110	1436	1201	1145	1098	-1,08	-23,54	-8,58	-4,10	0
518	1092	1479	1007	1103	1007	-7,78	-31,91	0,00	-8,70	0
519	1224	1610	1189	1245	1107	-9,56	-31,24	-6,90	-11,08	0
520	903	1154	895	972	895	-0,89	-22,44	0,00	-7,92	0
521	1135	1469	1078	1106	1078	-5,02	-26,62	0,00	-2,53	0
522	1710	2226	1754	1622	1516	-11,35	-31,90	-13,57	-6,54	0
523	1559	1892	1550	1630	1551	-0,51	-18,02	0,06	-4,85	0
524	992	1339	970	1011	972	-2,02	-27,41	0,21	-3,86	0
525	1554	1995	1457	1616	1387	-10,75	-30,48	-4,80	-14,17	0
526	1486	1927	1428	1438	1339	-9,89	-30,51	-6,23	-6,88	0
527	1127	1461	1137	1111	1259	11,71	-13,83	10,73	13,32	0
528	1008	1355	1029	1026	977	-3,08	-27,90	-5,05	-4,78	0
529	1156	1546	1145	1119	1322	14,36	-14,49	15,46	18,14	0
530	1523	1856	1495	1579	1495	-1,84	-19,45	0,00	-5,32	0
moyenne						-3,21	-25,73	-0,82	-4,91	0

$Gap_1 = (C_{GTSB} - LB)/LB$, $Gap_2 = (C_{GTSB} - UB)/UB$
$Gap_3 = (C_{GTSB} - C_{SBN})/C_{SBN}$, $Gap_4 = (C_{GTSB} - C_{GATS})/C_{GATS}$

TABLE 5.13 – Résultats pour les instances de classe 4 de [Mateo et al., 2002], avec 5 *jobs*, 5 cuves et 2 robots

Inst.	C_{SBN}	C_{Tabou}	C_{GTSB}	Gaps %		écartype
				Gap_1	Gap_2	
501	795	770	735	-7,55	-4,55	0
502	723	754	639	-11,62	-15,25	0
503	714	781	670	-6,16	-14,21	0
504	647	709	586	-9,43	-17,35	0
505	697	801	690	-1,00	-13,86	0
506	727	716	653	-10,18	-8,80	0
507	708	803	692	-2,26	-13,82	0
508	612	711	663	8,33	-6,75	0
509	642	692	660	2,80	-4,62	0
510	625	642	588	-5,92	-8,41	0
511	765	777	765	0,00	-1,54	0
512	684	740	686	0,29	-7,30	0
513	741	769	716	-3,37	-6,89	0
514	774	759	705	-8,91	-7,11	0
515	587	624	587	0,00	-5,93	0
516	756	786	755	-0,13	-3,94	0
517	760	713	678	-10,79	-4,91	0
518	725	754	722	-0,41	-4,24	0
519	825	834	727	-11,88	-12,83	0
520	556	650	530	-4,68	-18,46	0
521	728	757	728	0,00	-3,83	0
522	1107	1037	974	-12,01	-6,08	0
523	929	969	904	-2,69	-6,71	0
524	701	695	697	-0,57	0,29	0
525	958	877	814	-15,03	-7,18	0
526	969	866	794	-18,06	-8,31	0
527	805	789	739	-8,20	-6,34	0
528	692	771	687	-0,72	-10,89	0
529	865	840	776	-10,29	-7,62	0
530	979	985	1005	2,66	2,03	0
moyenne				-4,93	-7,85	0

$Gap_1 = (C_{GTSB} - C_{SBN})/C_{SBN}$, $Gap_2 = (C_{GTSB} - C_{Tabou})/C_{Tabou}$

TABLE 5.14 – Résultats pour les instances de classe 4 : [Paul et al., 2007]

Inst.	Ref. résultats moyens		résultats SBN			Gaps
	ATW	écartype	meilleur	moyenne	écartype	$Gap_{ATW}(\%)$
1	28,69	1,72	21,28	23,41	1,51	-18,41
2	27,94	2,63	22,13	23,77	2,05	-14,93
3	27,76	2,63	21,77	22,96	0,75	-17,28
4	27,60	3,27	24,01	25,90	1,03	-6,18
5	27,91	3,23	20,29	22,21	1,08	-20,41
6	26,73	2,76	18,89	21,68	1,57	-18,87
7	27,37	3,03	21,59	24,26	2,12	-11,37
8	27,69	2,73	25,22	27,77	1,37	0,31
9	26,88	2,83	22,51	24,23	1,74	-9,85
10	27,07	3,05	25,34	27,58	1,87	1,90
11	27,74	3,48	25,02	27,24	1,18	-1,83
12	26,49	3,00	26,97	27,05	0,51	2,13
13	27,27	3,22	26,46	29,06	1,49	6,55
14	26,63	1,99	28,71	30,98	1,27	16,35
moyenne					1,4	-6,56

$Gap_{ATW} = (C_{max} - ATW)/ATW$

TABLE 5.15 – Résultats comparés avec SBN pour les instances de classe 4 de [Paul et al., 2007], avec 40 jobs, 18 cuves et 1 robot

Inst	SBN		GTSB		Gaps%	
	meilleure C_1	moyenne C'_1	meilleure C_2	moyenne C'_2	g_a	g_b
1	22,5617	23,7059	21,28	23,41	-5,70	-1,24
2	24,2683	24,0996	22,13	23,77	-8,82	-1,37
3	21,0972	23,1109	21,77	22,96	3,19	-0,65
4	21,5622	25,6507	24,01	25,90	11,33	0,97
5	20,5956	22,2144	20,29	22,21	-1,47	-0,02
6	22,3358	22,3048	18,89	21,68	-15,43	-2,78
7	22,7897	25,3120	21,59	24,26	-5,28	-4,16
8	24,0083	27,3814	25,22	27,77	5,06	1,44
9	22,0444	24,2213	22,51	24,23	2,10	0,03
10	26,1106	27,8892	25,34	27,58	-2,94	-1,10
11	24,7803	26,8266	25,02	27,24	0,95	1,52
12	23,3481	25,9461	26,97	27,05	15,51	4,25
13	26,4442	28,7997	26,46	29,06	0,06	0,90
14	28,1825	31,1244	28,71	30,98	1,86	-0,45
moyenne gap					0,03	-0,19
écart type gap					0,08	0,02

$g_a = (C_2 - C_1)/C_1$
$g_b = (C_{2'} - C_{1'})/C_{1'}$

TABLE 5.16 – Résultats pour les instances de classe 5 avec critère = *makespan*

Inst.	Meilleur C_{max} de GATS			Meilleur C_{max} de GTSB			Gaps%	
	avec min stock>0		Min stock=0	avec min stock>0		Min stock=0		
	C_{GATS}	stock	C'_{GATS}	C_{GTSB}	stock	C'_{GTSB}	g_1	g_2
fjsp1	144	4	146	144	54	168	0,00	16,67
fjsp2	118	15	122	120	44	136	1,69	13,33
fjsp3	124	0	124	124	61	162	0,00	30,65
fjsp4	124	0	124	118	54	130	-4,84	10,17
fjsp5	94	0	94	94	36	130	0,00	38,30
fjsp6	144	10	146	144	62	168	0,00	16,67
fjsp7	124	16	126	122	56	164	-1,61	34,43
fjsp8	180	7	183	180	7	205	0,00	13,89
fjsp9	150	10	152	150	10	168	0,00	12,00
fjsp10	178	0	178	178	0	202	0,00	13,48
moyenne							-0,48%	19,96%
écart type							0,02	0,10

$g_1 = (C_{GTSB} - C_{GATS})/C_{GATS}$, $g_2 = (C'_{GTSB} - C'_{GATS})/C'_{GATS}$

TABLE 5.17 – Synthese des résultats

Inst.	Tabou	SBN	GATS	GTSB	
Classe 1	2	4	1	3	GATS
Classe 2	3	4	2	1	GTSB
Classe 3 Cmax	4	3	2	1	
Classe 3 sortie	3	2		1	
Classe 4 Mateo et Company 5×5 2 robot	3	2	X	1	
Classe 4 Mateo et Company 5×5 et 1 robot	X	2	X	1	
Classe 4 Paul et al.	-	2	X	1	
Classe 5	X	X	2	1	
range moyen	3	2,7	2	1,25	

que nous avons défini au début de ce mémoire. Au final, notre méthode hybride GTSB donne des résultats qui ne sont pas si loin des meilleurs, pour une exploitation qui dépasse le cadre strict de résolution d'un problème spécifique : aide au dimensionnement du stock, aide à l'amélioration globale des performances, résolution de diverses classes de problèmes possibles, sans identification précise préalable. Nous pouvons conclure que notre algorithme métaheuristique hybride offre un bon compromis entre les performances et la flexibilité de résolution (capacité à traiter plusieurs types de problèmes complexes d'ordonnancement en *job shop*).

Chapitre 6

Conclusion

6.1 Conclusion

Le problème GFJSSP que nous traitons englobe un ensemble de contraintes associées à divers types de job shop avec transport étudiés dans la littérature. Cette généralisation implique une difficulté de modélisation supplémentaire, qui rend effectivement plus difficile la résolution du problème d'ordonnancement, y compris avec des heuristiques ou dans des cas particuliers, tels que ceux que nous avons évoqués au chapitre 4. Ainsi, l'amélioration des solutions de référence, qui est l'objet de plusieurs travaux dans la littérature dans les problèmes avec stockage illimité, est contrariée par la nécessité première de trouver des solutions faisables, ce qui est le premier challenge des cas sans stock. La combinaison de ces deux aspects explique en partie que les résultats obtenus avec nos méthodes ne soient parfois pas aussi bons que nous l'avions espéré. Toutefois, nous avons pu résoudre des problèmes complexes en termes de contraintes, et de taille relativement importante, y compris sur de nouvelles instances. Pour parvenir à ce résultat, nous avons combiné une procédure tabou ayant pour objectif d'améliorer une solution initiale, avec une méthode hybride de type shifting bottleneck (SBN) destinée à fournir au tabou une solution initiale faisable. Le tout étant intégré dans un algorithme génétique, qui alimente en particulier le SBN en affectation préalable des tâches de traitement.

6.2 Perspectives

En termes de perspectives, deux directions peuvent être envisagées à partir de nos travaux : elles concernent respectivement l'extension du modèle et l'amélioration des méthodes.

6.2.1 Extensions

Notre modèle générique actuel ne prend pas en compte la topologie de transport. Donc, en particulier, nous ne traitons pas le problème du risque de collision. En prenant en compte la structure du réseau de transport et les règles de pilotage associées, la faisabilité des solutions pourrait être remise en cause et sera donc à nouveau à valider. Une approche possible serait de chercher d'abord les solutions sans considérer les contraintes de topologie, en utilisant les algorithmes proposés. Puis la faisabilité de ces solutions serait analysée a posteriori relativement aux contraintes induites par ces réseaux. Cette méthode est intéressante pour des réseaux avec flexibilité de routage.

De plus, nous pourrions étendre notre modèle pour y intégrer de nouvelles hypothèses :

- rendre les *release dates* variables au lieu de les normaliser par zéro. Ces contraintes peuvent facilement être intégrées dans notre graphe générique en les ajoutant entre le nœud de départ et les nœuds de stockage amont de la première tâche de chaque *job*.

- rendre les *due dates* variables au lieu de leur affecter une très grande valeur. Cela pourrait induire la prise en compte d'un nouveau critère lié au retard.

- le temps de traitement pour chaque opération peut-être dépendant de la machine, ce qui correspond plus à un cas réel. Ainsi, l'affectation des ressources impacte plus le *makespan*.

- rendre le temps de déplacement en charge et/ou à vide dépendant non seulement des ressources de traitement, mais aussi des ressources de transport et/ou du *job* transporté. Le premier s'adresse aux ressources de transport non identiques, par exemple avec des vitesses de déplacement variées.

- permettre aux tâches de transport d'être effectuées par un sous-ensemble des ressources de transport. Pour appliquer cette contrainte, nous pouvons modifier une partie des valeurs des chromosomes. Les valeurs des bits associés aux tâches de transport sont choisies parmi un sous-ensemble des ressources associées, comme c'est déjà le cas pour les machines.

- ajouter des contraintes de temps bornés pour d'autres tâches que celles de traitement :

  1. le temps de déplacement en charge entre deux tâches opératoires successives est supérieur à une borne minimale. Ceci est lié à une topologie spécifique, par exemple pour deux chemins différents possibles entre deux ressources de traitement, où les temps de transport ne sont pas toujours symétriques.
  2. le temps de stockage en amont et/ou aval d'une machine peut être inclus dans d'autres types d'intervalles que $[0, 0]$ ou $[0, \infty]$. Cette contrainte assure

la qualité des produits, par exemple, dans les systèmes de production agro-alimentaires. Avec notre graphe générique, ces deux types de contraintes peut être prises en compte correctement.

- intégrer une capacité finie (non nulle) de certains *buffers*. Cela permettrait de prendre en compte non seulement le temps de stock mais aussi la charge en nombres de produits. Par contre, il serait moins facile de modifier le graphe pour représenter directement cette capacité.

6.2.2 Amélioration de la performance des méthodes

Différentes pistes d'amélioration des procédures mises en oeuvre dans notre méthode hybride peuvent être explorées, notamment pour réduire les temps de résolution. Plus précisément, ces améliorations pourraient se situer au niveau :

- de l'évaluation, notamment avec le graphe disjonctif générique, car elle est très coûteuse dans le processus de résolution ;
- de la détermination d'un voisinage plus efficace pour explorer l'espace des solutions ;
- de l'algorithme génétique : affinage des paramètres, recherche de nouveaux opérateurs, moins classiques et plus dédiés aux problèmes de *job shop* avec transport.

Quelques unes de ces pistes potentielles sont explicitées ci-dessous.

Choix multiple de ressources pour les tâches

Pour les données des travaux qui imposent le choix des ressources, la procédure SBN va rechercher les séquences faisables pour les ressources de traitement puis pour les ressources de transport. Si les données ne fixent pas les choix de ressources, la procédure doit affecter et ordonnancer ces ressources (cf. figure 6.1).

Si une tâche peut être traitée sur plusieurs ressources, il faut donc faire un choix avant d'arbitrer la disjonction. Si les données fixent l'affectation des tâches de transport et que la procédure heuristique n'arrive pas à trouver une solution faisable, on doit donc appliquer une procédure pour réaffecter toutes les tâches et ainsi avoir plus de chances d'avoir une solution faisable. De la même manière, une tâche opératoire peut avoir plusieurs choix de machines, et les données fournies peuvent fixer une affectation qui n'est pas faisable, ou encore qui donne moins de chances de trouver une solution faisable. On peut donc appliquer une procédure pour les affecter et ordonnancer. Pour cela, on affecte les tâches dynamiquement. Le principe de l'affectation est d'essayer d'affecter la tâche à une ressource qui est disponible le plus tôt.
L'idée générale est en fait ici d'appliquer aux traitements le même type d'heuristique HA que celle que nous avons appliquée aux transports. L'avantage ici est de pouvoir considérer de la même manière et simultanément les tâches de traitement et de transport.

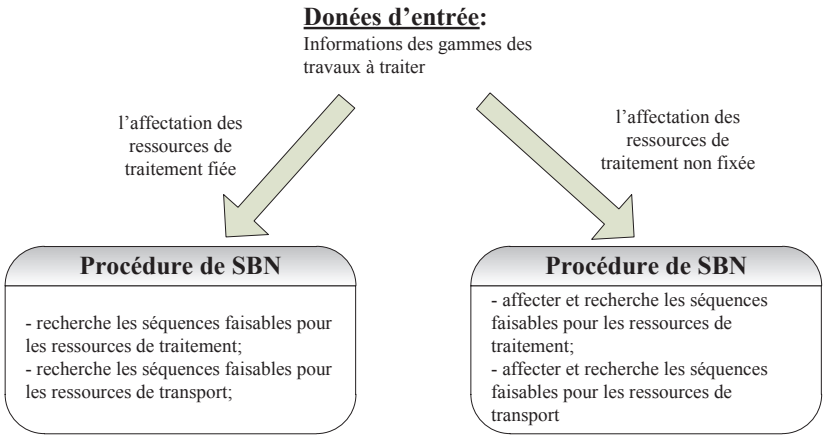

FIGURE 6.1 – Fonctionnement de la procédure SBN suivant le problème d'affectation.

Constructions partielles du graphe générique

La complexité de calcul du chemin le plus long dans un graphe dépend strictement du nombre de nœuds. Notre graphe générique permet d'intégrer efficacement les contraintes considérées et est capable de représenter plus de types de solutions que les graphes disjonctifs classiques. Le côté négatif de ce graphe est d'avoir plus de nœuds (respectivement 4 fois ou 2 fois plus par rapport aux graphes simples utilisés par [Rossé-Bloch, 1999] et [Hurink and Knust, 2005]), ce qui augmente énormément la complexité de l'évaluation (respectivement 64 ou 8 fois plus long par rapport aux deux autres graphes). Pour cela, nous pourrions envisager une construction au fur et à mesure d'un graphe partiel qui contient le nombre de *jobs* actuel dans le système de production. Il n'est alors plus nécessaire d'ajouter des arcs supplémentaires (comme proposé dans les cas particuliers du chapitre 4). Ce graphe partiel pourrait aussi être utilisé pour un problème d'ordonnancement dynamique.

Calcul parallèle

Pour réduire le temps de calcul, on peut utiliser des techniques de calcul parallèle ou distribué. Ces techniques décomposent un algorithme en plusieurs sous tâches, et les distribuent sur plusieurs processeurs où elles sont exécutées simultanément. Les processus doivent être synchronisés pour respecter les dépendances entre tâches. Trois éléments doivent être considérés pour une implémentation parallèle ou distribuée : un ensemble de ressources de calcul (processeurs ou ordinateurs), la parallélisation du problème, et son implémentation. Cela permet de diminuer le temps de calcul et de prendre en compte des problèmes de taille plus importante. Dans nos algorithmes, plusieurs tâches peuvent être parallélisées :

- l'évaluation de chaque individu qui peut être facilement parallélisée sur les différents cœurs d'un processeur grâce à OpenMP ;
- le calcul du chemin le plus long dans un graphe, facilement parallélisable sur un GPGPU [1] ;

1. General Purpose computing on Graphics Processing Units, soit calcul général effectué sur des processeurs graphiques.

Première partie

annexes

Annexe A

Implémentation du modèle mathématique

```
//Les variable de décision
dvar int c[1..n];
dvar int p[1..n][1..maxOi][1..m];
dvar int P[1..n][1..maxOi][1..m];
dvar int PJijk[1..n][1..maxOi][1..m] in 0..1;

dvar int t[1..n][1..maxOi-1][1..m][1..r];
dvar int T[1..n][1..maxOi-1][1..m][1..r];
dvar int TJijh[1..n][1..maxOi-1][1..r] in 0..1;
dvar int DT[1..n][1..maxOi-1][1..m][1..m] in 0..1;

dvar int US[1..n][1..maxOi][1..m];
dvar int DS[1..n][1..maxOi][1..m];

dvar int Stor;
dvar int+ Cmax;

minimize Cmax;
subject to {

//premier objectif
Cmax==max(i in 1..n)c[i];

//contraintes de domaines
forall(i in 1..n, k in 1..m) p[i][1][k]>=0;
forall(i in 1..n, j in 1..Oi, k in 1..m)
```

```
{
  US[i][j][k]>=0;
  DS[i][j][k]>=0;
}

forall(i in 1..n, j in 1..Oi[i]-1, k in 1..m, h in 1..r)
t[i][j][k][h]>=0;

//date de fin du job i
forall(i in 1..n) c1=c[i]==sum(k in 1..m)(p[i][Oi[i]][k]+P[i][Oi[i]][k]);

//second objectif
Stor==sum(i in 1..n, j in 2..Oi[i], k in 1..m)US[i][j][k]+
      sum(i in 1..n, j in 1..Oi[i]-1, k in 1..m)DS[i][j][k];

//calcul des durées de stockage en amont et en aval de l'opération OPij
forall(i in 1..n, j in 1..Oi[i]-1)
sum(k1 in 1..m, h in 1..r)(t[i][j][k1][h]+T[i][j][k1][h])==
     sum(k2 in 1..m)(p[i][j+1][k2]-US[i][j+1][k2]);

forall(i in 1..n, j in 1..Oi[i]-1, k in 1..m)
c6=DS[i][j][k]==sum(h in 1..r)t[i][j][k][h]-p[i][j][k]-P[i][j][k];

//l'opération OPij n'est réalisable que sur une seule machine
forall(i in 1..n, j in 1..Oi[i], k in 1..m: MPijk[i][j][k]<1) c8=PJijk[i][j][k]=0;
forall(i in 1..n, j in 1..Oi[i])c9=sum(k in 1..m) PJijk[i][j][k]=1;

//contraintes sur la date de début et la durée d'une opération de traitement
forall(i in 1..n, j in 1..Oi[i], k in 1..m)
c2=p[i][j][k]<= Mn*PJijk[i][j][k];
forall(i in 1..n, j in 1..Oi[i], k in 1..m)
c13=Pmoins[i][j]*PJijk[i][j][k]<=P[i][j][k]
forall(i in 1..n, j in 1..Oi[i], k in 1..m)
c14=P[i][j][k]<=Pplus[i][j]*PJijk[i][j][k];

//contraintes de précédence
forall(i in 1..n, j in 1..Oi[i], t in 1..Oi[i] : Uijj[i][j][t]>0)
   c7= sum(k in 1..m) (p[i][j][k] + P[i][j][k]) <= sum(k in 1..m) p[i][t][k];

//contraintes de capacité d'une ressource de traitement k
```

```
forall(i1 in 1..n, j1 in 1..Oi[i1], k in 1..m : MPijk[i1][j1][k]>0)
   sum(i2 in 1..n, j2 in 1..Oi[i2] : MPijk[i2][j2][k]>0)
   ((PJijk[i1][j1][k]>0)*(PJijk[i2][j2][k]>0)*
   (p[i2][j2][k]>=p[i1][j1][k])*(p[i2][j2][k] < p[i1][j1][k] + P[i1][j1][k]))<2;

//l'opération TPij n'est réalisable que sur un seul robot
forall(i in 1..n, j in 1..Oi[i]-1)
c10= sum(h in 1..r) TJijh[i][j][h]==1;

//contraintes sur la date de début et la durée d'une opération de transport
forall(i in 1..n, j in 1..Oi[i]-1,k in 1..m)
c3= sum(h in 1..r) t[i][j][k][h] <= Mn*PJijk[i][j][k];
forall(i in 1..n, j in 1..Oi[i]-1,k in 1..m)
c4= sum(h in 1..r) T[i][j][k][h] <= Mn*PJijk[i][j][k];
forall(i in 1..n, j in 1..Oi[i]-1,h in 1..r)
c24= sum(k in 1..m) t[i][j][k][h] <= Mn*TJijh[i][j][h];
forall(i in 1..n, j in 1..Oi[i]-1,h in 1..r)
c19= sum(k in 1..m) T[i][j][k][h] <= Mn*TJijh[i][j][h];

//contraintes sur les opérations de stockage USTij et DSTij
forall(i in 1..n, j in 1..Oi[i], k in 1..m)
c15= USmoins[i][j]*PJijk[i][j][k] <= US[i][j][k];
forall(i in 1..n, j in 1..Oi[i], k in 1..m)
c16= US[i][j][k] <= USplus[i][j]*PJijk[i][j][k];
forall(i in 1..n, j in 1..Oi[i]-1, k in 1..m)
c17= DSmoins[i][j]*PJijk[i][j][k] <= DS[i][j][k];
forall(i in 1..n, j in 1..Oi[i]-1, k in 1..m)
c18= DS[i][j][k] <= DSplus[i][j]*PJijk[i][j][k];

//contraintes sur la durée de la tâche TPij
//la variable DT est égale à 1 si la tâche TPij s'effectue de la machine k1 à la ma
forall(i in 1..n, j in 1..Oi[i]-1)
c20= sum(k1 in 1..m, k2 in 1..m) DT[i][j][k1][k2] == 1;
forall(i in 1..n, j in 1..Oi[i]-1, k1 in 1..m)
c21= sum(k2 in 1..m) DT[i][j][k1][k2] == PJijk[i][j][k1];
forall(i in 1..n, j in 1..Oi[i]-1, k2 in 1..m)
c22= sum(k1 in 1..m) DT[i][j][k1][k2] == PJijk[i][j+1][k2];
forall(i in 1..n, j in 1..Oi[i]-1, k in 1..m)
   c23= sum(h in 1..r) T[i][j][k][h] ==
    sum(k2 in 1..m) (DT[i][j][k][k2]*theta[k][k2]);
```

```
//contraintes de capacité d'une ressource de transport h
forall(i1 in 1..n, j1 in 1..Oi[i1]-1, k1 in 1..m,
k3 in 1..m, h in 1..r : MPijk[i1][j1][k1]*MPijk[i1][j1+1][k3]>0)
   sum(i2 in 1..n, j2 in 1..Oi[i2]-1,k2 in 1..m) ((PJijk[i1][j1][k1]>0)*
    (PJijk[i2][j2][k2]>0)*(PJijk[i1][j1+1][k3]>0)*(TJijh[i1][j1][h]>0)*
    (TJijh[i2][j2][h]>0)*(t[i2][j2][k2][h]>=t[i1][j1][k1][h])*
    (t[i2][j2][k2][h] < t[i1][j1][k1][h] + T[i1][j1][k1][h] + rho[k3][k2]))<2;
};
```

Annexe B

Algorithme de calcul du plus long chemin et de détection des circuits de longueur positive [Gondran and Minoux, 1985]

Soit un graphe donné $G = (V, U)$, où V est l'ensemble de nœuds et U est l'ensemble d'arcs. Chaque arc $u \in U$ est lié par deux nœuds appartenant à V. On définit :

- l_{ij}, $i, j \in V$:
  - la longueur de l'arc lié aux deux sommets i et j si $(i, j) \in U$;
  - $-\infty$ sinon.
- l'_{ij}, $i, j \in V$:
  - la longueur du plus long chemin entre i et j s'il existe ;
  - $-\infty$ sinon.
- matrice $L = (l_{ij})$ et $L' = (l'_{ij})$;

Le problème est de trouver le chemin le plus long $(\mu(i, j))$ entre deux sommets donnés (de i à j) tel que la longueur totale $l(\mu) = \sum_{u \in \mu} l(u)$ soit maximale.

L'algorithme de Dantzig peut calculer et détecter les circuits de longueur positive. Chaque fois qu'un nouveau sommet est concerné, il y a deux étapes à suivre : mettre à jour les longueurs du chemin entre n'importe quel ancien nœud et ce nouveau nœud ; vérifier si le nouveau sommet génère un circuit de longueur positive, si aucun circuit positif n'est détecté, mettre à jour les longueurs du chemin entre les deux anciens nœuds. Sa complexité est $O(|V|^3)$ ($|V|$: le nombre total de sommets).

La procédure est décrite ci-dessous :

Algorithm 14 Algorithme de Dantzig

for $i, j \in [1, |V|]$ **do**
  $l'_{ij} = l_{ij}$
end for
for $k \in [1, |V|]$ **do**
  for $i \in [1, k-1]$ **do**
    $l'_{ik+1} = max_{1 \leqslant j \leqslant k}(l'_{ij} + l'_{jk+1})$
    $l'_{k+1i} = max_{1 \leqslant j \leqslant k}(l'_{k+1j} + l'_{ji})$
  end for
  $t = max_{1 \leqslant i \leqslant k}(l'_{k+1i} + l'_{ik+1})$
  if $t < 0$ **then**
    s'arrêter *//circuit de longueur positive détecté*
  else
    for $i, j \in [1, k]$ **do**
      $l'_{ij} = max(l'_{ij}, l'_{ik+1} + l'_{k+1j})$
    end for
  end if
end for

Annexe C

Procédure de *Branch and Bound* avec règle EDD pour résoudre $1|r_j|Lmax$

Un algorithme par séparation et évaluation, en anglais *branch and bound*, est une méthode de résolution de problèmes d'optimisation combinatoire ou discrète. Elle décrit l'ensemble des solutions sous forme d'arborescence. C'est une méthode d'énumération dans laquelle on évite d'énumérer certaines mauvaises solutions en les comparant à une borne connue. Dans un bon algorithme par séparation et évaluation, lorsqu'une bonne solution réalisable est connue, les mauvaise branches sont éliminés sans avoir examiné chacun des nœuds. Seules les branches des solutions potentiellement bonnes sont donc explorées et énumérées. L'efficacité est donc dépendante de la qualité de la borne par défaut. Deux principes sont définis pour mettre en œuvre cette procédure :

- séparation : permet de diviser le problème en un certain nombre de sous-problèmes pour construire l'arborescence. En résolvant récursivement tous les sous-ensembles et en prenant la meilleure solution, le problème sera résolu.
- évaluation : évaluer un nœud pour se déplacer dans l'arborescence ou déterminer que le sous-ensemble des solutions associé à ce nœud ne contient pas de solution optimale. Pour cela, la méthode consiste à déterminer une borne pour l'ensemble des solutions. Si cette borne est pire que la meilleure solution trouvée jusqu'à présent, on assure que le sous-ensemble est inutile.

Le problème $1|r_j|L_{max}$ est de minimiser le L_{max} en ordonnançant n opérations exécutées sur une même ressource.

Règle de séparation

L'arborescence de recherche a une hiérarchie naturelle. Chaque niveau correspond à une séquence partielle de passage des opérations sur la ressource de traitement. Le premier niveau (notamment 0) n'est constitué que d'un nœud qui n'est lié à aucune opération. À chaque niveau k, les chemins depuis la racine jusqu'à chaque nœud de ce niveau correspondent aux différents sous séquencements de k premières tâches. Pour chaque nœud, il reste $n - k$ opérations à déterminer dans la séquence. Les nœuds du niveau suivant

sont les enfants, et le nombre de niveaux final est égal au nombre total d'opérations à ordonnancer. La figure C.1 montre l'espace des solutions explorées par l'arbre.

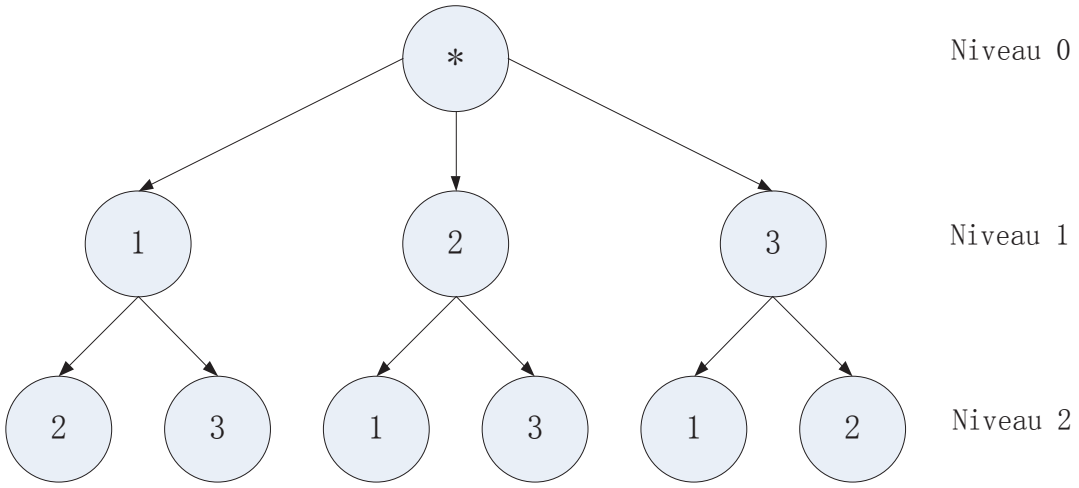

FIGURE C.1 – L'arborescence explorée.

Règle d'exploration

La règle permet d'éliminer certaines branches. Si pour un nœud du niveau k, les opérations OP_1 à OP_{k-1} sont séquencées aux premières $k-1$ positions, alors l'opération OP_k n'est prise en compte que si : $r_k < min_{i \in J}(max(t, r_i) + p_i)$, où r_k est la date de disponibilité de l'opération OP_k, J l'ensemble de tâches à ordonnancer, t la date de fin de OP_{k-1}, p_i la durée de OP_i. Si OP_k ne satisfait pas cette condition, il existe une opération parmi celles restantes qui peut être séquencée à cette place sans retarder l'exécution de OP_k. Avec cette règle, certains nœuds de niveau k sont éliminés sans explorer leurs enfants.

Évaluation Pour tous les nœuds restant à ordonnancer, les bornes inférieures peuvent être obtenues en utilisant la règle heuristique nommée "preemptive Earliest Due Date" (ou preemptive EDD) pour ordonnancer les opérations. Cette règle fournit une évaluation par défaut pour le problème sans préemption. La règle preemptive EDD peut être décrite par l'heuristique suivante [Rossé-Bloch, 1999] :

Algorithm 15 Heuristique de la règle preemptive EDD.

$E = \emptyset$ //*Initialisation de l'ensemble des opérations disponibles.*
while $J \neq \emptyset$ **do**
  $\rho_1 = min_{i \in J} r_i$
  if toutes les tâches sont disponibles à ρ_1 **then**
    $\rho_2 = \infty$
  else
    $\rho_2 = min_{i \in J} r_i | r_i \neq \rho_1$
  end if
  $E = j | r_j = \rho_1$
  $d_k = min_{j \in E} d_j$
  $l = min p_k, \rho_2 - \rho 1$
  séquencer OP_k à l'intervalle $[\rho_1, \rho_1 + l]$
  if $p_k \leqslant l$ **then**
    $J = J - k$
  else
    $p_k = p_k - l$
    for tout $j \in E$ **do**
      $r_j = \rho_1 + l$
    end for
  end if
end while

Annexe D

Données de [Bilge and Ulusoy, 1995]

Les ensembles de jobs sont montrés dans les tableaux de D.1 à D.10. M_0 est le poste de chargement. Les temps opératoires sont fixés, les capacités de stockage sont illimitées. Chaque tâche peut être traitée sur une seule ressource. Ces dix ensembles de jobs sont combinés avec des temps de déplacement différents, définis par quatre topologies. On a donc au total 40 instances différentes.

TABLE D.1 – Données d'ensemble de jobs 1

job	ressource (temps opératoires)			
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}
1	$M_0(0)$	$M_1(8)$	$M_2(16)$	$M_4(12)$
2	$M_0(0)$	$M_1(20)$	$M_3(10)$	$M_2(18)$
3	$M_0(0)$	$M_3(12)$	$M_4(8)$	$M_1(15)$
4	$M_0(0)$	$M_4(14)$	$M_2(18)$	
5	$M_0(0)$	$M_3(10)$	$M_1(15)$	

TABLE D.2 – Données d'ensemble de jobs 2

job	ressource (temps opératoires)			
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}
1	$M_0(0)$	$M_1(10)$	$M_4(18)$	
2	$M_0(0)$	$M_2(10)$	$M_4(18)$	
3	$M_0(0)$	$M_1(10)$	$M_3(20)$	
4	$M_0(0)$	$M_2(10)$	$M_3(15)$	$M_4(12)$
5	$M_0(0)$	$M_1(10)$	$M_2(15)$	$M_4(12)$
6	$M_0(0)$	$M_1(10)$	$M_2(15)$	$M_3(12)$

TABLE D.3 – Données d'ensemble de jobs 3

job	ressource (temps opératoires)				
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}	OP_{i5}
1	$M_0(0)$	$M_1(16)$	$M_3(15)$		
2	$M_0(0)$	$M_2(18)$	$M_4(15)$		
3	$M_0(0)$	$M_1(20)$	$M_2(10)$		
4	$M_0(0)$	$M_3(15)$	$M_4(10)$		
5	$M_0(0)$	$M_1(8)$	$M_2(10)$	$M_3(15)$	$M_4(17)$
6	$M_0(0)$	$M_2(10)$	$M_3(15)$	$M_4(8)$	$M_1(15)$

TABLE D.4 – Données d'ensemble de jobs 4

job	ressource (temps opératoires)					
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}	OP_{i5}	OP_{i6}
1	$M_0(0)$	$M_4(11)$	$M_1(10)$	$M_2(7)$		
2	$M_0(0)$	$M_3(12)$	$M_2(10)$	$M_4(8)$		
3	$M_0(0)$	$M_2(7)$	$M_3(10)$	$M_1(9)$	$M_3(8)$	
4	$M_0(0)$	$M_2(7)$	$M_4(8)$	$M_1(12)$	$M_2(6)$	
5	$M_0(0)$	$M_1(9)$	$M_2(7)$	$M_4(8)$	$M_2(10)$	$M_3(8)$

TABLE D.5 – Données d'ensemble de jobs 5

job	ressource (temps opératoires)			
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}
1	$M_0(0)$	$M_1(6)$	$M_2(12)$	$M_4(9)$
2	$M_0(0)$	$M_1(18)$	$M_3(6)$	$M_2(15)$
3	$M_0(0)$	$M_3(9)$	$M_4(3)$	$M_1(12)$
4	$M_0(0)$	$M_4(6)$	$M_2(15)$	
5	$M_0(0)$	$M_3(3)$	$M_1(9)$	

TABLE D.6 – Données d'ensemble de jobs 6

job	ressource (temps opératoires)			
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}
1	$M_0(0)$	$M_1(9)$	$M_2(11)$	$M_4(7)$
2	$M_0(0)$	$M_1(19)$	$M_2(20)$	$M_4(13)$
3	$M_0(0)$	$M_2(14)$	$M_3(20)$	$M_4(9)$
4	$M_0(0)$	$M_2(14)$	$M_3(20)$	$M_4(9)$
5	$M_0(0)$	$M_1(11)$	$M_3(16)$	$M_4(8)$
6	$M_0(0)$	$M_1(10)$	$M_3(12)$	$M_4(10)$

TABLE D.7 – Données d'ensemble de jobs 7

job	ressource (temps opératoires)			
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}
1	$M_0(0)$	$M_1(6)$	$M_4(6)$	
2	$M_0(0)$	$M_2(11)$	$M_4(9)$	
3	$M_0(0)$	$M_2(9)$	$M_4(7)$	
4	$M_0(0)$	$M_3(16)$	$M_4(7)$	
5	$M_0(0)$	$M_1(9)$	$M_3(18)$	$M_4(8)$
6	$M_0(0)$	$M_2(13)$	$M_3(19)$	$M_4(6)$
7	$M_0(0)$	$M_1(10)$	$M_2(9)$	$M_3(13)$
8	$M_0(0)$	$M_1(11)$	$M_2(9)$	$M_4(8)$

TABLE D.8 – Données d'ensemble de jobs 8

job	ressource (temps opératoires)				
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}	OP_{i5}
1	$M_0(0)$	$M_2(12)$	$M_3(21)$	$M_4(11)$	
2	$M_0(0)$	$M_2(12)$	$M_3(21)$	$M_4(11)$	
3	$M_0(0)$	$M_2(20)$	$M_3(10)$	$M_4(11)$	
4	$M_0(0)$	$M_2(20)$	$M_3(10)$	$M_4(11)$	
5	$M_0(0)$	$M_1(10)$	$M_2(14)$	$M_3(18)$	$M_4(9)$
6	$M_0(0)$	$M_1(10)$	$M_2(14)$	$M_3(18)$	$M_4(9)$

TABLE D.9 – Données d'ensemble de jobs 9

job	ressource (temps opératoires)				
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}	OP_{i5}
1	$M_0(0)$	$M_3(9)$	$M_1(12)$	$M_2(9)$	$M_4(6)$
2	$M_0(0)$	$M_3(16)$	$M_2(11)$	$M_4(9)$	
3	$M_0(0)$	$M_1(21)$	$M_2(18)$	$M_4(7)$	
4	$M_0(0)$	$M_2(20)$	$M_3(22)$	$M_4(11)$	
5	$M_0(0)$	$M_3(14)$	$M_1(16)$	$M_2(13)$	$M_4(9)$

TABLE D.10 – Données d'ensemble de jobs 10

job	ressource (temps opératoires)				
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}	OP_{i5}
1	$M_0(0)$	$M_1(11)$	$M_3(19)$	$M_2(16)$	$M_4(13)$
2	$M_0(0)$	$M_2(21)$	$M_3(16)$	$M_4(14)$	
3	$M_0(0)$	$M_3(8)$	$M_2(10)$	$M_1(14)$	$M_4(9)$
4	$M_0(0)$	$M_2(13)$	$M_3(20)$	$M_4(10)$	
5	$M_0(0)$	$M_1(9)$	$M_3(16)$	$M_4(18)$	
6	$M_0(0)$	$M_2(19)$	$M_1(21)$	$M_3(11)$	$M_4(15)$

Les tableaux suivants montrent les temps de déplacement à vide et en charge entre deux ressources de traitement. Les matrices ne sont pas toujours symétriques.

TABLE D.11 – Temps de déplacement pour la topologie 1

FMS1	M_0	$M_0 1$	M_2	M_3	M_4
M_0	0	6	8	10	12
M_1	12	0	6	8	10
M_2	10	6	0	6	8
M_3	8	8	6	0	6
M_4	6	10	8	6	0

TABLE D.12 – Temps de déplacement pour la topologie 2

FMS2	M_0	$M_0 1$	M_2	M_3	M_4
M_0	0	4	6	8	6
M_1	6	0	2	4	2
M_2	8	12	0	2	4
M_3	6	10	12	0	2
M_4	4	8	10	12	0

TABLE D.13 – Temps de déplacement pour la topologie 3

FMS3	M_0	$M_0 1$	M_2	M_3	M_4
M_0	0	2	4	10	12
M_1	12	0	2	8	10
M_2	10	12	0	6	8
M_3	4	6	8	0	2
M_4	2	4	6	12	0

TABLE D.14 – Temps de déplacement pour la topologie 4

FMS4	M_0	$M_0 1$	M_2	M_3	M_4
M_0	0	4	8	10	14
M_1	18	0	4	6	10
M_2	20	14	0	8	6
M_3	12	8	6	0	6
M_4	14	14	12	6	0

Annexe E

Données FJSP avec transport ([Deroussi and Norre, 2010])

[Deroussi and Norre, 2010] ont proposé des instances de FJSP en intégrant la première topologie utilisée dans les instances de Bilge & Ulsoy. Chaque tâche opératoire peut être traitée sur une des deux ressources de traitement. Il y a 8 machines ($M_1, ..., M_8$) et un poste de chargement (M_0), 2 ressources de transport. Les données des ensembles de jobs sont montrés dans les tableaux de E.1 à E.10. Pour ces instances, deux objectifs sont pris en compte : le makespan et la sortie du dernier job par le poste de chargement. Donc pour le deuxième critère, une tâche supplémentaire est ajoutée pour chaque job avec $P^{-}_{iO_i} = 0, MP_{iO_i} = M_0$. Les temps de déplacement en charge et à vide entre deux ressources de traitement sont identiques, et ils sont montrés dans le tableau E.11.

TABLE E.1 – Données d'ensemble de jobs 1

job	ressource (temps opératoires)			
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}
1	$M_0(0)$	$M_1, M_2(16)$	$M_3, M_4(32)$	$M_7, M_8(24)$
2	$M_0(0)$	$M_1, M_2(40)$	$M_5, M_6(20)$	$M_3, M_4(36)$
3	$M_0(0)$	$M_5, M_6(24)$	$M_7, M_8(16)$	$M_1, M_2(30)$
4	$M_0(0)$	$M_7, M_8(28)$	$M_3, M_4(36)$	
5	$M_0(0)$	$M_5, M_6(20)$	$M_1, M_2(30)$	
6	$M_0(0)$	$M_1, M_2(16)$	$M_3, M_4(32)$	$M_7, M_8(24)$
7	$M_0(0)$	$M_1, M_2(40)$	$M_5, M_6(20)$	$M_3, M_4(36)$

TABLE E.2 – Données d'ensemble de jobs 2

job	ressource (temps opératoires)			
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}
1	$M_0(0)$	$M_1, M_2(20)$	$M_7, M_8(36)$	
2	$M_0(0)$	$M_3, M_4(20)$	$M_7, M_8(36)$	
3	$M_0(0)$	$M_1, M_2(20)$	$M_5, M_6(40)$	
4	$M_0(0)$	$M_3, M_4(20)$	$M_5, M_6(30)$	$M_7, M_8(24)$
5	$M_0(0)$	$M_1, M_2(20)$	$M_3, M_4(30)$	$M_7, M_8(24)$
6	$M_0(0)$	$M_1, M_2(20)$	$M_3, M_4(30)$	$M_5, M_6(24)$

TABLE E.3 – Données d'ensemble de jobs 3

job	ressource (temps opératoires)				
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}	OP_{i5}
1	$M_0(0)$	$M_1, M_2(32)$	$M_5, M_6(30)$		
2	$M_0(0)$	$M_3, M_4(36)$	$M_7, M_8(30)$		
3	$M_0(0)$	$M_1, M_2(40)$	$M_3, M_4(20)$		
4	$M_0(0)$	$M_5, M_6(30)$	$M_7, M_8(20)$		
5	$M_0(0)$	$M_1, M_2(16)$	$M_3, M_4(20)$	$M_5, M_6(30)$	$M_7, M_8(34)$
6	$M_0(0)$	$M_3, M_4(20)$	$M_5, M_6(30)$	$M_7, M_8(16)$	$M_1, M_2(30)$

TABLE E.4 – Données d'ensemble de jobs 4

job	ressource (temps opératoires)					
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}	OP_{i5}	OP_{i6}
1	$M_0(0)$	$M_7, M_8(22)$	$M_1, M_2(20)$	$M_3, M_4(14)$		
2	$M_0(0)$	$M_5, M_6(24)$	$M_3, M_4(20)$	$M_7, M_8(16)$		
3	$M_0(0)$	$M_3, M_4(14)$	$M_5, M_6(20)$	$M_1, M_2(18)$	$M_5, M_6(16)$	
4	$M_0(0)$	$M_3, M_4(14)$	$M_7, M_8(16)$	$M_1, M_2(24)$	$M_3, M_4(12)$	
5	$M_0(0)$	$M_1, M_2(18)$	$M_3, M_4(14)$	$M_7, M_8(16)$	$M_3, M_4(20)$	$M_5, M_6(16)$

TABLE E.5 – Données d'ensemble de jobs 5

job	ressource (temps opératoires)			
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}
1	$M_0(0)$	$M_1, M_2(12)$	$M_3, M_4(24)$	$M_7, M_8(18)$
2	$M_0(0)$	$M_1, M_2(36)$	$M_5, M_6(12)$	$M_3, M_4(30)$
3	$M_0(0)$	$M_5, M_6(18)$	$M_7, M_8(6)$	$M_1, M_2(24)$
4	$M_0(0)$	$M_7, M_8(12)$	$M_3, M_4(30)$	
5	$M_0(0)$	$M_5, M_6(6)$	$M_1, M_2(18)$	

Table E.6 – Données d'ensemble de jobs 6

job	ressource (temps opératoires)			
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}
1	$M_0(0)$	$M_1, M_2(18)$	$M_3, M_4(22)$	$M_7, M_8(14)$
2	$M_0(0)$	$M_1, M_2(38)$	$M_3, M_4(40)$	$M_7, M_8(26)$
3	$M_0(0)$	$M_3, M_4(28)$	$M_5, M_6(40)$	$M_7, M_8(18)$
4	$M_0(0)$	$M_3, M_4(28)$	$M_5, M_6(40)$	$M_7, M_8(18)$
5	$M_0(0)$	$M_1, M_2(22)$	$M_5, M_6(32)$	$M_7, M_8(16)$
6	$M_0(0)$	$M_1, M_2(20)$	$M_5, M_6(24)$	$M_7, M_8(20)$

Table E.7 – Données d'ensemble de jobs 7

job	ressource (temps opératoires)			
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}
1	$M_0(0)$	$M_1, M_2(12)$	$M_7, M_8(12)$	
2	$M_0(0)$	$M_3, M_4(22)$	$M_7, M_8(18)$	
3	$M_0(0)$	$M_3, M_4(18)$	$M_7, M_8(14)$	
4	$M_0(0)$	$M_5, M_6(32)$	$M_7, M_8(14)$	
5	$M_0(0)$	$M_1, M_2(18)$	$M_5, M_6(36)$	
6	$M_0(0)$	$M_3, M_4(26)$	$M_5, M_6(38)$	$M_7, M_8(12)$
7	$M_0(0)$	$M_1, M_2(20)$	$M_3, M_4(18)$	$M_5, M_6(26)$
8	$M_0(0)$	$M_1, M_2(22)$	$M_3, M_4(18)$	$M_7, M_8(16)$

Table E.8 – Données d'ensemble de jobs 8

job	ressource (temps opératoires)				
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}	OP_{i5}
1	$M_0(0)$	$M_3, M_4(24)$	$M_5, M_6(42)$	$M_7, M_8(22)$	
2	$M_0(0)$	$M_3, M_4(24)$	$M_5, M_6(42)$	$M_7, M_8(22)$	
3	$M_0(0)$	$M_3, M_4(24)$	$M_5, M_6(42)$	$M_7, M_8(22)$	
4	$M_0(0)$	$M_3, M_4(24)$	$M_5, M_6(42)$	$M_7, M_8(22)$	
5	$M_0(0)$	$M_1, M_2(20)$	$M_3, M_4(28)$	$M_5, M_6(36)$	$M_7, M_8(9)$
6	$M_0(0)$	$M_1, M_2(20)$	$M_3, M_4(28)$	$M_5, M_6(36)$	$M_7, M_8(18)$

Table E.9 – Données d'ensemble de jobs 9

job	ressource (temps opératoires)				
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}	OP_{i5}
1	$M_0(0)$	$M_5, M_6(18)$	$M_1, M_2(24)$	$M_3, M_4(18)$	$M_7, M_8(12)$
2	$M_0(0)$	$M_5, M_6(32)$	$M_3, M_4(22)$	$M_7, M_8(18)$	
3	$M_0(0)$	$M_1, M_2(42)$	$M_3, M_4(36)$	$M_7, M_8(14)$	
4	$M_0(0)$	$M_3, M_4(40)$	$M_5, M_6(44)$	$M_7, M_8(22)$	
5	$M_0(0)$	$M_5, M_6(28)$	$M_1, M_2(32)$	$M_3, M_4(26)$	$M_7, M_8(18)$

TABLE E.10 – Données d'ensemble de jobs 10

job	ressource (temps opératoires)				
i	OP_{i1}	OP_{i2}	OP_{i3}	OP_{i4}	OP_{i5}
1	$M_0(0)$	$M_1(22)$	$M_3(38)$	$M_2(32)$	$M_4(26)$
2	$M_0(0)$	$M_2(42)$	$M_3(32)$	$M_4(28)$	
3	$M_0(0)$	$M_3(16)$	$M_2(20)$	$M_1(28)$	$M_4(18)$
4	$M_0(0)$	$M_2(26)$	$M_3(40)$	$M_4(20)$	
5	$M_0(0)$	$M_1(18)$	$M_3(32)$	$M_4(36)$	
5	$M_0(0)$	$M_2(38)$	$M_1(42)$	$M_3(22)$	$M_4(30)$

TABLE E.11 – Temps de déplacement

FMS4	M_0	$M_0$1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
M_0	0	6	8	6	8	10	12	10	12
M_1	8	0	2	8	2	4	6	4	6
M_2	6	10	0	10	8	2	4	6	4
M_3	12	4	6	0	6	8	10	8	10
M_4	10	2	4	6	0	6	8	2	8
M_5	8	8	2	8	6	0	6	4	2
M_6	6	10	8	10	8	6	0	6	4
M_7	12	4	6	4	2	8	10	0	10
M_8	10	6	4	6	4	2	8	2	0

Annexe F

Données d'instances de [Mateo et al., 2002]

Ces sont les instances de HSP générées par [Mateo and Companys, 2007]. Il y a deux types de jobs pour chaque instance. Le nombre de ressources de traitement (cuves) varie de 5 à 10. Les postes de chargement et déchargement sont dissociés. Donc il y au total de 7 à 12 ressources (y compris les postes de chargement et déchargement). Les gammes de chaque job commencent toujours depuis le poste de chargement, ensuite visitent chaque cuve une seule fois depuis la première jusqu'à la dernière, et finissent par le poste de déchargement. Les temps de traitement pour chaque tâche différents pour les deux types de jobs, sauf les temps passés dans les postes de chargement et déchargement ($[0, \infty]$). Il y une trentaine d'instances pour chaque groupe d'instances avec le même nombre de cuves. Les temps de traitement (depuis la première cuve jusqu'à la dernière cuve) et de transport sont donnés dans les tableaux de F.1 à F.1. Les temps de déplacement en charge et à vide entre deux cuves successives sont identiques et ils sont montrés dans les deux dernières colonnes des tableaux. Donc $\sigma_{kl} = |k - l| \times \sigma_{kk+1}$, $\tau_{kl} = |k - l| \times \tau_{kk+1}$.

TABLE F.1 – Données des instances de Mateo et al. avec 5 cuves

Inst.	P_{ij}^-, P_{ij}^+					σ, τ_{kk+1}	
501	50,114	80,184	60,145	80,191	28,67	8	12
	49,126	71,178	54,152	53,159	56,166		
502	55,123	57,138	43,104	67,151	62,143	7	10
	45,123	20,52	68,204	73,207	25,73		
503	20,47	49,121	22,54	68,167	67,148	10	15
	49,131	78,206	27,78	30,88	30,81		
504	46,107	51,115	79,194	24,52	35,87	7	10
	43,125	22,55	40,103	49,131	64,169		
505	50,117	57,134	51,117	44,96	54,131	9	13
	72,207	27,76	79,213	40,112	46,119		

continué sur la page suivante

suite de la page précédente

Inst.	P_{ij}^-, P_{ij}^+					σ, τ_{kk+1}	
506	77,184	57,142	67,160	45,105	27,65	8	12
	33,361	33,249	43,286	28,112	23,208		
507	20,47	49, 121	22, 54	68, 167	67, 148	10	15
	71, 417	43, 412	44, 372	62, 320	27, 215		
508	46, 107	51, 115	79, 194	24, 52	35, 87	7	10
	43, 279	73, 762	24, 168	56, 274	76, 693		
509	46, 107	51, 115	79, 194	24, 52	35, 87	7	10
	76 ,299	26, 268	72, 673	72, 484	74, 726		
510	25, 55	34, 78	79, 182	48, 111	25, 55	8	12
	33, 361	33, 249	43, 286	28, 112	23, 208		
511	71, 167	63, 146	49, 109	45, 109	58, 141	8	16
	40, 109	45, 130	40, 109	59, 16	2 34, 99		
512	43, 98	61, 143	75, 167	35, 79	68, 160	5	10
	57, 168	72, 196	44, 130	75, 193	69 ,177		
513	68, 149	23, 54	29, 65	74, 183	66, 160	7	14
	48, 139	26, 75	77, 224	72, 190	20, 58		
514	68, 167	39, 95	69, 165	51, 122	45, 110	7	14
	48, 139	26 , 75	77, 224	72, 190	20 , 58		
515	32 , 72	70 ,168	31, 75	20, 50	28, 67	6	12
	53, 135	62, 177	47, 121	50 ,142	58, 159		
516	28, 67	40 , 95	26, 59	29, 66	43 ,106	10	20
	28, 164	61, 378	26, 225	32, 203	57, 486		
517	63, 144	63 ,146	37 , 88	30, 70	53, 124	8	16
	27 ,153	30, 183	22,207	38, 308	55, 371		
518	68 ,167	39 , 95	69, 165	51 ,122	45, 110	7	14
	44, 257	32 ,339	54, 179	71, 290	69, 259		
519	71, 167	63, 146	49,109	45 ,109	58, 141	8	16
	31 ,231	78 ,423	21, 128	59, 178	78, 663		
520	32, 72	70 ,168	31, 75	20, 50	28 , 67	6	12
	24 ,134	37 ,135	69 ,572	58, 246	46 ,332		
521	40 , 89	63, 155	34, 84	73, 171	33, 78	6	18
	44 ,129	27 , 75	80 ,238	55 ,148	27 , 70		
522	63, 157	70 ,161	80, 199	73, 174	74, 185	9	27
	55 ,158	41, 121	30 , 86	50, 138	45 ,134		
523	54 ,128	42 , 94	47 ,113	32, 77	70, 166	9	27
	55, 158	41, 121	30 , 86	50,138	45 ,134		
524	35 , 81	80 ,187	38, 94	58 ,130	45 ,111	5	15
	32 , 92	48 ,123	77, 215	67, 194	33 , 83		
525	58, 128	76 ,185	64, 156	48 ,118	72, 172	8	24
	43, 110	21 , 60	63 ,179	26 , 72	23, 60		
526	58, 128	76 ,185	64, 156	48,118	72, 172	8	24
	32, 352	43 ,210	40, 135	23, 90	43, 158		
527	40 , 89	63 ,155	34, 84	73, 171	33, 78	6	18

continué sur la page suivante

suite de la page précédente

Inst.	P_{ij}^-, P_{ij}^+					σ, τ_{kk+1}	
	46 ,403	63, 370	78 ,386	24, 227	45, 411		
528	35 , 81	80 ,187	38 , 94	58, 130	45, 111	5	15
	59, 260	74 ,662	43, 248	50, 464	71, 457		
529	40 , 92	62, 148	47, 116	66, 159	67 ,157	6	18
	46, 403	63, 370	78, 386	24, 227	45, 411		
530	54, 128	42, 94	47, 113	32 , 77	70, 166	9	27
	75, 727	72 ,249	56, 492	51 ,316	74, 289		

TABLE F.2 – Données des instances de Mateo et al. avec 6 cuves

Inst.	P_{ij}^-, P_{ij}^+						σ, τ_{kk+1}	
601	60, 134	63, 147	80 ,199	71, 161	62, 145	72, 161	8	12
	38, 110	63, 188	45, 117	34, 86	37, 95	78, 213		
602	60, 134	63, 147	80, 199	71 ,161	62, 145	72, 161	8	12
	75, 210	72, 189	22, 62	66, 188	52 ,150	70 ,191		
603	30, 70	48, 119	70 ,155	45 ,105	24 ,57	72, 170	6	9
	73, 212	55, 146	69, 189	44,122	34, 86	48, 128		
604	30, 70	48 ,119	70, 155	45,105	24 ,57	72, 170	6	9
	73, 212	38, 113	69, 183	23, 59	64, 179	49, 131		
605	30, 70	59 ,144	44 ,108	45, 109	24, 57	32 ,76	6	9
	73, 212	38, 113	69, 183	23, 59	64 ,179	49, 131		
606	63, 153	49 ,120	62, 155	59,133	31, 76	26, 62	5	7
	74, 310	67 ,673	22, 219	44, 444	59, 360	21, 190		
607	30, 70	48, 119	70, 155	45 ,105	24, 57	72 ,170	6	9
	40, 381	40, 403	27, 116	50,159	24, 160	44, 197		
608	31, 74	29, 65	56, 137	71, 160	49 ,117	72, 167	6	9
	40, 381	40, 403	27, 116	50, 159	24, 160	44 ,197		
609	71, 158	21 ,49	57, 130	31, 68	29, 68	29, 64	10	15
	30, 184	28 ,270	47, 162	65 ,391	67 ,403	25, 219		
610	30, 70	59, 144	44, 108	45 ,109	24 ,57	32 ,76	6	9
	51, 477	53 ,366	35,304	68, 741	35 ,225	60, 462		
611	33, 77	32, 74	44, 98	23, 57	53, 122	56, 134	7	14
	77, 228	31, 82	23, 63	36, 93	40, 105	44, 125		
612	73, 172	65, 152	50, 115	79, 186	37, 83	74, 173	9	18
	59 175	76 192	30 84	51 148	72 182	57 147		
613	47, 114	42, 95	47 ,106	72, 168	74 ,181	57, 140	9	18
	59, 175	76, 192	30, 84	51, 148	72, 182	57, 147		
614	39, 93	65 ,155	78, 171	69, 152	24, 59	35, 79	9	18
	26, 70	32, 85	54, 143	22 ,63	70, 182	50, 126		
615	39, 93	65, 155	78 ,171	69, 152	24, 59	35, 79	9	18
	59, 175	76, 192	30, 84	51, 148	72, 182	57, 147		
616	70, 155	56, 140	62, 139	43, 102	34, 81	79, 173	8	16
	33, 144	79, 788	73, 686	43, 140	54, 398	24, 177		

continué sur la page suivante

suite de la page précédente

Inst.	P_{ij}^-, P_{ij}^+							σ, τ_{kk+1}	
617	73, 172	65, 152	50 ,115	79, 186	37 ,83	74, 173		9	18
	42, 229	70, 399	20, 88	26, 123	56,601	36, 200			
618	73, 161	63, 151	44, 101	35, 84	61, 144	69, 167		7	14
	55, 397	28 ,245	62 ,532	39 ,217	60 ,537	56 ,378			
619	47, 114	42 ,95	47, 106	72, 168	74, 181	57, 140		9	18
	49, 437	56, 435	56, 392	37, 383	66, 648	75, 322			
620	26, 57	65, 146	66, 147	64, 143	37,85	60, 136		9	18
	42, 229	70, 399	20, 88	26, 123	56, 601	36, 200			
621	66, 156	52, 119	54, 120	75, 182	45, 99	30 ,71		10	30
	24, 70	30 ,76	30, 83	33, 92	79 ,217	31, 80			
622	24, 52	56, 128	68, 165	59, 135	37, 88	36 ,90		6	18
	30, 84	25, 71	44,124	27, 75	32 ,85	63, 178			
623	21, 50	20 ,46	38, 84	21, 49	59, 130	28, 63		7	21
	27, 68	54 ,147	26, 68	63, 186	45, 126	29, 82			
624	68, 159	47, 116	68, 149	35, 81	26, 61	23, 55		8	24
	31, 93	23, 65	71, 210	68, 171	28, 73	76, 196			
625	59, 138	26, 61	44 ,101	30, 67	28, 65	38, 92		7	21
	27, 68	54, 147	26, 68	63, 186	45, 126	29 ,82			
626	21, 46	47, 105	45 ,107	77, 183	37, 83	21 ,48		8	24
	25, 232	71, 218	61 ,502	47, 447	58, 615	41 ,124			
627	24, 52	56 ,128	68,165	59, 135	37, 88	36, 90		6	18
	34, 274	26 ,252	57, 172	56, 515	67, 646	45, 380			
628	68, 159	47, 116	68, 149	35, 81	26, 61	23, 55		8	24
	25, 232	71, 218	61, 502	47, 447	58, 615	41 ,124			
629	21, 50	20, 46	38, 84	21, 49	59, 130	28, 63		7	21
	23, 95	57, 335	40 ,333	26, 85	34 ,247	29 c118			
630	39, 86	59, 136	52, 126	39, 88	77, 170	68, 160		5	15
	78, 642	45, 352	70, 269	23 ,236	46, 172	27, 281			

TABLE F.3 – Données des instances de Mateo et al. avec 7 cuves

Inst.	P_{ij}^-, P_{ij}^+							σ, τ_{kk+1}	
701	75, 173	67, 156	80, 183	65, 152	45, 99	71, 169	75, 184	7	10
	54, 150	49 ,127	76, 220	24, 71	57, 152	67, 189	29, 75		
702	74, 171	57, 139	56 132	70, 165	55, 128	65, 145	33, 80	9	13
	23, 67	62 177	35, 99	32,87	67, 177	70, 181	64, 164		
703	21, 49	28, 66	61, 136	46, 102	28, 69	24, 56	72, 169	10	15
	58, 169	71, 199	73, 208	59, 154	32, 84	58, 163	41, 110		
704	68, 161	79, 181	37, 86	46, 111	28 ,63	55, 127	35, 78	6	9
	31, 91	49, 131	39, 115	36, 99	78, 205	56, 158	30, 88		
705	53,131	33, 75	24, 56	54, 125	31, 72	70 ,159	30, 72	9	13
	23, 67	62, 177	35, 99	32 ,87	67 ,177	70 ,181	64 ,164		
706	75 ,173	67, 156	80, 183	65, 152	45, 99	71, 169	75, 184	7	10
	41, 409	67, 731	34 ,331	30, 150	50 ,205	34 ,213	30, 329		

continué sur la page suivante

suite de la page précédente

Inst.	P_{ij}^{-}, P_{ij}^{+}							σ, τ_{kk+1}	
707	53, 131	33 ,75	24 ,56	54, 125	31, 72	70 ,159	30 ,72	9	13
	79, 247	31, 124	45 ,458	39, 330	29 ,242	36 ,176	47, 206		
708	75, 173	67, 156	80, 183	65 ,152	45,99	71, 169	75 ,184	7	10
	61, 374	47, 403	73 ,238	29, 310	32, 330	29 ,153	52, 265		
709	50, 119	58 ,131	28, 66	50 ,124	56, 138	32, 74	60, 140	7	14
	41, 102	62 ,178	70, 197	76 ,199	76 ,216	78, 226	58, 168		
710	50 ,119	58, 131	28 ,66	50, 124	56, 138	32 ,74	60, 140	7	14
	41, 102	62, 178	70 ,197	76 ,199	76, 216	78, 226	58, 168		
711	55,134	59, 147	43, 106	37 ,89	36, 88	73, 177	20, 48	8	16
	47,129	37, 107	61, 182	55, 138	59 ,150	36, 107	38, 98		
712	50 ,119	58, 131	28, 66	50,124	56, 138	32, 74	60, 140	7	14
	52, 143	48, 141	21 ,63	48, 138	21, 56	41, 107	41, 104		
713	74, 168	35, 79	71, 177	54 ,118	61,140	37 ,81	52, 122	5	10
	64, 162	80, 215	75, 215	54 ,136	23 ,59	73,197	36, 106		
714	74, 168	35, 79	71, 177	54,118	61, 140	37 ,81	52, 122	5	10
	24 ,126	36 ,143	24 ,248	66 ,522	69, 299	60, 312	25 ,259		
715	55, 134	59, 147	43, 106	37 ,89	36 ,88	73 ,177	20 ,48	8	16
	78, 672	66 ,709	58 ,419	27, 235	38, 119	41,172	69 ,351		
716	55 ,134	59,147	43, 106	37, 89	36, 88	73 ,177	20 ,48	8	16
	30, 282	65,340	66 ,426	64, 679	28, 188	52, 206	30 ,314		
717	57 ,136	62, 147	46, 106	38 ,91	62, 139	42, 97	39, 93	6	12
	76,568	42 ,233	21 ,99	76 ,387	50 ,205	43 ,326	77, 772		
718	50, 119	58 ,131	28, 66	50, 124	56, 138	32, 74	60, 140	7	14
	79, 279	62, 340	43, 203	51, 279	36, 150	31, 142	32, 289		
719	31, 74	52, 115	74, 172	50, 114	30, 68	43, 103	21, 51	9	27
	31, 87	62, 160	29, 79	52, 138	28, 79	62, 176	57, 160		
720	79, 187	73 ,181	26, 58	61, 138	51, 122	29, 63	73, 174	5	15
	63,187	61, 166	51, 149	36, 107	43, 113	34, 97	51, 130		
721	60 ,133	45, 105	67 ,163	24, 60	72 ,170	72,168	36, 83	8	24
	78, 209	76, 206	22, 65	50 ,142	66 ,182	36,98	51, 142		
722	60, 133	45, 105	67, 163	24, 60	72, 170	72, 168	36, 83	8	24
	35, 100	61, 159	64, 182	65, 167	68, 180	56, 160	77, 211		
723	70, 159	31, 75	27 ,60	76, 170	47, 107	30, 69	53 ,123	8	24
	35, 100	61, 159	64, 182	65, 167	68 ,180	56 ,160	77, 211		
724	63 ,142	37, 91	68,162	22, 52	32 ,73	66, 156	50 ,121	10	30
	41 ,318	37 ,378	32 ,165	73 ,498	56 ,581	54, 472	31 ,223		
725	68,168	39, 85	58, 131	32, 73	27, 66	27 ,66	44 ,110	7	21
	71,339	53, 551	58, 281	62, 316	57, 443	79 ,544	76, 775		
726	31 ,74	52, 115	74 ,172	50, 114	30, 68	43 ,103	21, 51	9	27
	60, 470	52, 546	70, 632	42, 409	30,292	24 ,96	47, 462		
727	60, 133	45, 105	67 ,163	24 ,60	72, 170	72, 168	36, 83	8	24
	37, 294	80, 655	52, 494	72, 734	43,345	48, 243	30, 324		
728	48 ,106	37, 92	72, 166	54,123	64 ,151	58, 131	69, 170	5	15

continué sur la page suivante

suite de la page précédente

Inst.	P_{ij}^-, P_{ij}^+							σ, τ_{kk+1}
	45 ,460	59, 319	42, 137	57, 438	34 ,248	25 ,178	60, 538	

TABLE F.4 – Données des instances de Mateo et al. avec 8 cuves

Inst.	P_{ij}^-, P_{ij}^+								σ, τ_{kk+1}	
801	36,87	33,78	35,85	37,91	38,86	41,100	46,107	40,93	5	7
	63,173	49,140	72,203	53,138	46,132	76,199	40,116	24,65		
802	73,168	66,148	75,178	41,100	61,150	22,53	66,146	45,110	7	10
	33,97	31,92	28,71	27,79	61,154	42,117	37,110	63,175		
803	75,170	30,72	24,56	36,87	51,112	50,112	23,51	77,171	7	10
	33,97	31,92	28,71	27,79	61,154	42,117	37,110	63,175		
804	60,142	67,147	53,125	32,80	35,86	29,68	45,108	26,64	5	7
	26,77	27,74	48,135	71,212	44,113	58,147	80,201	27,70		
805	45,112	64,142	47,110	76,187	30,70	29,67	24,53	77,178	7	10
	60,343	61,463	41,141	52,276	61,635	57,239	75,793	52,465		
806	67,150	40,94	75,166	28,63	46,107	52,126	36,85	63,156	8	12
	55,215	73,231	77,240	46,432	50,152	71,608	62,663	52,260		
807	53,120	75,183	65,156	52,117	33,81	24,53	36,86	75,171	8	12
	55,215	73,231	77,240	46,432	50,152	71,608	62,663	52,260		
808	76,177	49,108	56,125	65,157	60,137	31,74	50,114	22,54	7	10
	60,343	61,463	41,141	52,276	61,635	57,239	75,793	52,465		
809	76,177	49,108	56,125	65,157	60,137	31,74	50,114	22,54	7	10
	56,598	26,98	36,394	21,169	31,319	51,391	29,279	55,233		
810	43,99	21,47	34,83	66,162	67,162	47,111	73,168	21,50	6	12
	67,196	38,98	44,130	64,175	78,234	37,100	46,134	37,96		
811	61,146	26,64	73,174	61,149	20,47	20,49	33,73	42,92	6	12
	67,196	38,98	44,130	64,175	78,234	37,100	46,134	37,96		
812	70,166	39,96	63,152	59,139	22,49	42,100	58,136	50,117	8	16
	41,110	69,189	62,174	53,144	52,130	71,184	53,148	61,167		
813	24,59	50,121	37,88	37,86	79,193	22,50	35,85	32,71	8	16
	28,77	35,105	39,109	33,91	63,183	52,151	45,117	30,86		
814	41,101	32,79	64,145	74,180	50,111	40,89	21,50	69,166	6	12
	67,196	38,98	44,130	64,175	78,234	37,100	46,134	37,96		
815	24,59	50,121	37,88	37,86	79,193	22,50	35,85	32,71	8	16
	40,284	29,284	41,380	55,179	55,174	43,255	58,594	69,275		
816	64,150	29,65	27,64	42,97	60,137	52,122	53,131	26,62	10	20
	62,680	57,460	58,371	71,582	63,224	52,550	78,312	31,309		
817	61,139	21,49	60,135	24,54	60,137	61,148	26,64	41,98	5	10
	73,582	59,311	74,519	52,288	80,802	60,372	55,533	71,733		
818	61,139	21,49	60,135	24,54	60,137	61,148	26,64	41,98	5	10
	45,238	53,517	57,437	36,165	45,453	53,578	36,354	34,205		
819	47,108	39,95	69,160	74,162	58,142	28,61	59,143	58,142	8	16
	40,284	29,284	41,380	55,179	55,174	43,255	58,594	69,275		

continué sur la page suivante

suite de la page précédente

Inst.	P_{ij}^-, P_{ij}^+								σ, τ_{kk+1}	
820	41,95	52,125	52,114	48,118	22,51	77,185	80,185	66,156	9	27
	44,122	66,168	44,111	79,216	50,135	49,144	74,211	37,102		
821	41,95	52,125	52,114	48,118	22,51	77,185	80,185	66,156	9	27
	53,150	53,134	38,109	68,186	71,210	80,234	27,71	60,178		
822	28,65	49,112	37,84	41,100	50,117	54,121	78,178	74,168	8	24
	53,140	44,117	34,87	63,169	45,134	29,77	27,73	39,104		
823	62,140	67,166	74,172	70,172	54,132	53,130	76,173	71,168	8	24
	53,140	44,117	34,87	63,169	45,134	29,77	27,73	39,104		
824	20,48	46,103	47,105	36,88	21,51	41,95	72,171	38,94	8	24
	40,113	71,187	59,162	43,127	46,127	44,126	56,149	35,96		
825	62,140	67,166	74,172	70,172	54,132	53,130	76,173	71,168	8	24
	52,455	68,647	45,275	24,168	35,240	35,339	71,294	33,172		
826	41,98	78,185	48,110	70,157	41,91	75,177	49,120	35,84	5	15
	31,228	56,230	71,456	50,159	61,533	50,408	29,222	45,381		
827	41,95	52,125	52,114	48,118	22,51	77,185	80,185	66,156	9	27
	64,363	52,295	45,395	42,180	39,158	23,181	61,244	78,486		
828	41,95	52,125	52,114	48,118	22,51	77,185	80,185	66,156	9	27
	36,121	74,623	52,478	39,363	36,301	59,184	31,170	62,289		
829	52,122	39,89	63,144	74,175	46,106	55,132	68,155	67,147	6	18
	50,174	79,814	63,395	32,281	30,259	61,373	24,184	56,256		

TABLE F.5 – Données des instances de Mateo et al. avec 9 cuves

Inst.	P_{ij}^-, P_{ij}^+									σ, τ_{kk+1}	
901	42, 100	66, 152	36, 87	21, 50	29, 68	32, 74	68 ,168	60, 146	36, 85	9	13
	65, 184	65, 189	76, 192	21, 53	38, 113	56, 160	49, 128	51, 152	22, 56		
902	39, 97	67, 152	41, 92	39, 88	30, 66	47, 108	32, 76	71 ,174	78, 194	9	13
	65, 184	65, 189	76, 192	21, 53	38, 113	56, 160	49, 128	51, 152	22, 56		
903	52, 128	68, 170	43, 101	57, 133	32, 72	40, 90	66, 159	53, 129	30, 73	7	10
	48, 134	29, 74	72, 201	69 ,187	39, 103	59, 150	20, 55	49, 136	29, 85		
904	49, 114	41, 95	58, 133	42, 93	66 ,162	30, 73	63, 142	58, 142	32, 79	9	13
	65, 184	65 ,189	76, 192	21, 53	38, 113	56 ,160	49,128	51, 152	22, 56		
905	52, 128	68, 170	43 ,101	57, 133	32,72	40, 90	66, 159	53 ,129	30 ,73	7	10
	53, 151	80, 215	24, 65	27, 67	39, 105	40, 107	50, 133	44,131	56, 156		
906	52, 128	68, 170	43, 101	57, 133	32, 72	40,90	66 ,159	53, 129	30, 73	7	10
	33, 288	57, 362	47, 207	80, 387	43, 309	65, 622	38, 416	79, 516	64, 687		
907	39, 97	67, 152	41,92	39, 88	30 ,66	47, 108	32, 76	71, 174	78, 194	9	13
	21, 109	62, 576	59, 211	47,260	74, 253	53,212	42,207	50, 178	52, 568		
908	50 ,124	33 ,78	67 ,162	21 ,49	30,68	23,54	26 ,64	33, 75	64, 159	6	9
	34, 328	45, 221	76 ,577	74,757	71, 410	36, 395	34, 226	45, 290	60 ,285		
909	49, 114	41, 95	58, 133	42 ,93	66,162	30,73	63,142	58,142	32,79	9	13
	21,109	62,576	59,211	47,260	74,253	53,212	42,207	50,178	52,568		
910	50, 124	33, 78	67,162	21,49	30,68	23,54	26,64	33,75	64,159	6	9
	79,784	55,424	52,347	39,153	24,235	52,371	72,601	60,499	29,284		

continué sur la page suivante

suite de la page précédente

Inst.	P_{ij}^-, P_{ij}^+									σ, τ_{kk+1}	
911	64,159	28,61	25,56	40,97	32,80	32,78	53,116	26,60	50,124	7	14
	36,91	78,204	68,188	60,156	76,201	28,81	58,153	36,99	30,82		
912	60,136	59,136	76,189	32,70	48,110	42,102	65,148	64,154	24,53	6	12
	55,143	57,142	61,181	57,164	45,117	54,153	58,169	42,121	20,60		
913	71, 171	48,110	76,174	47,109	69,159	35, 78	69,163	44,107	40,90	6	12
	55,143	57,142	61,181	57,164	45,117	54,153	58,169	42,121	20,60		
914	65,154	55,126	31,68	34,80	21,52	35,78	76,189	44,105	51,123	8	16
	54,154	40,108	77,223	58,151	55,161	69,198	20,58	52,132	61,178		
915	44,102	62,141	30,71	33,74	76,180	45,100	32,73	72,178	49,115	8	16
	46,297	42,203	39,425	41,310	61,651	48,313	64,242	49,261	60,202		
916	76,180	71,163	25,60	24,54	66,151	53, 121	56, 129	69 ,154	47 ,105	10	20
	43,298	24, 192	37, 339	58, 633	37, 338	76, 501	22, 122	21, 92	27, 209		
917	24, 57	51, 126	74, 172	32 ,75	61 ,135	47 ,109	24, 58	52, 122	79, 185	8	16
	63, 218	69, 658	75,557	38, 343	36 ,181	28 ,240	41, 297	21 ,84	22, 200		
918	24,57	51,126	74,172	32,75	61,135	47,109	24,58	52,122	79,185	8	16
	46,297	42,203	39,425	41,310	61,651	48,313	64,242	49,261	60,202		
919	71, 171	48, 110	76, 174	47, 109	69,159	35, 78	69,163	44, 107	40,90	6	12
	62, 439	41, 439	74, 351	53 ,463	76, 575	50, 172	53, 168	46, 376	71, 310		
920	22, 50	68 ,167	62 ,155	75 ,171	41 ,98	47 ,108	36 ,86	39, 87	68 ,156	9	27
	35, 102	51, 137	41, 113	60 ,163	24 ,61	30, 86	74, 204	28, 71	36, 95		
921	56, 133	74 ,169	65 ,145	62, 138	72, 164	23 ,57	77 ,183	54 ,119	77, 187	8	24
	78, 218	61, 157	35, 98	26, 73	80, 212	57, 163	29 ,84	49, 125	77 ,209		
922	31,71	26,61	63,152	27,66	39,92	55,126	65,149	70,171	59,137	7	21
	65,166	53,132	77,198	54,156	50,128	33,91	45,114	30,76	70,180		
923	68,163	63,148	33,74	77,184	58,142	45,111	76,185	49,117	40,90	5	15
	28,81	29,79	77,197	23,61	30,86	73,195	70,199	36,99	68,197		
924	22,50	68,167	62,155	75,171	41,98	47,108	36,86	39,87	68,156	9	27
	78,203	21,53	60,152	68,170	58,164	68,200	56,149	43,123	39,105		
925	56,133	74,169	65,145	62,138	72,164	23,57	77,183	54,119	77,187	8	24
	34,264	29,141	62,369	56,547	33,350	59,431	70,393	30 ,164	70,561		
926	69,168	52,129	21,47	27,66	35,81	79,194	62,153	62,152	74,171	7	21
	67,701	24,83	39,293	61,635	69,227	60,225	39,348	28,124	68,394		
927	28,64	45,110	53,127	73,166	70,156	78,195	28,63	79,185	21,48	6	18
	47,474	44,238	46,500	30,111	27,106	52,546	77,644	75,799	63,282		
928	69,168	52,129	21,47	27,66	35,81	79,194	62,153	62,152	74,171	7	21
	57,262	32,136	72,611	40,190	51,347	80,475	72,333	56,366	35,262		
929	22,50	68,167	62,155	75,171	41, 98	47, 108	36, 86	39, 87	68, 156	9	27
	32,320	49,344	70,669	41,447	38,407	56,368	55,329	71,521	22,84		

TABLE F.6 – Données des instances de Mateo et al. avec 9 cuves

Inst.	P_{ij}^-, P_{ij}^+									σ, τ_{kk+1}	
101	42,98	58,127	36, 82	70,170	22,55	46,112	71,158	44,108	72,173	21,47	6	9
	78,214	31,90	49,141	33,88	32,93	74,197	44,111	69,202	37,97	65,174		
102	43,99	74,183	67,148	51,118	55,125	55,135	36,79	75,165	62,144	76,180	7	10

continué sur la page suivante

suite de la page précédente

Inst.	P_{ij}^-, P_{ij}^+										σ, τ_{kk+1}	
	79,222	59,168	36,107	50,125	21,61	64,189	66,180	43,121	70,201	33,85		
103	22,52	75,171	21,46	36,88	37,84	49,114	54,131	45,109	38,91	70,167	9	13
	54,138	68,202	37,103	47,134	74,190	25,67	42,110	41,103	50,130	68,204		
104	22,52	75,171	21,46	36,88	37,84	49,114	54,131	45,109	38,91	70,167	9	13
	24,70	57,149	67,185	35,93	73,205	21,55	59,173	68,185	62,156	69,182		
105	73,167	45,104	40,88	34,75	62,140	33,73	33,76	50,120	41,97	46,110	6	9
	78,214	31,90	49,141	33,88	32,93	74,197	44,111	69,202	37,97	65,174		
106	42,98	58,127	36,82	70,170	22,55	46,112	71,158	44,108	72,173	21,47	6	9
	43,388	72,545	37,355	57,439	54,249	21,84	27,159	44,346	76,407	57,227		
107	58,129	41,93	73,163	37,89	68,162	43,95	66,156	51,117	37,85	21,46	10	15
	68,503	48,239	51,254	27,144	58,354	48,209	46,494	24,177	78,611	60,440		
108	43,99	74,183	67,148	51,118	55,125	55,135	36,79	75,165	62,144	76,180	7	10
	54,472	39,276	43,181	59,581	54,229	29,100	26,161	28,142	41,235	51,449		
109	70,156	76,177	23,52	26,65	69,161	43,95	76,169	69,171	76,174	59,133	5	7
	47,231	70,600	73,746	39,135	54,489	33,159	64,466	44,270	42,294	28,208		
110	22,52	75,171	21,46	36,88	37,84	49,114	54,131	45,109	38,91	70,167	9	13
	41,422	37,247	66,292	26,155	29,316	29,317	30,217	77,617	48,237	33,333		
111	57,134	79,179	24,57	80,177	39,96	22,49	63,139	68,159	20,46	55,125	8	16
	36,90	60,179	47,123	61,180	73,198	71,188	24,70	79,201	40,107	23,65		
112	24,57	69,167	30,72	77,189	55,128	72,171	46,105	79,194	60,138	76,174	9	18
	52,155	42,121	77,213	57,155	56,150	74,206	65,191	67,196	36,98	78,228		
113	61,151	61,136	34,76	35,81	57,136	44,106	44,98	73,182	74,166	38,87	6	12
	66,197	33,90	38,112	77,216	79,229	66,197	65,164	43,110	51,137	68,201		
114	48,114	50,112	40,89	79,196	49,111	58,135	65,149	45,111	38,85	57,133	7	14
	67,187	79,221	77,193	43,123	64,166	40,111	44, 112	45,123	52, 139	67,200		
115	67, 159	60 ,138	37 ,88	77 ,173	73 ,174	21 ,48	31 ,77	43 ,97	22,52	67,155	10	20
	50, 139	42 ,124	52 ,139	54 ,161	73 ,189	62 ,183	36 ,102	50, 148	31 ,80	45 ,134		
116	61,151	61,136	34,76	35,81	57,136	44,106	44,98	73,182	74,166	38,87	6	12
	55,603	71,514	47,429	68,270	55,547	31,125	38,343	63,308	27,110	38,282		
117	23,56	79,178	21,46	63,154	59,139	25,55	51,113	48,111	27,62	48,119	5	10
	28,141	75,300	42,367	45,214	53,564	48,155	50,231	60,550	43,294	38,346		
118	48,114	50,112	40,89	79, 196	49, 111	58 ,135	65, 149	45,111	38,85	57, 133	7	14
	78 ,384	61,217	47, 359	52, 216	56 ,321	79 ,764	59, 189	56 ,286	69 ,421	30,295		
119	67,159	60,138	37,88	77,173	73,174	21,48	31,77	43,97	22,52	67,155	10	20
	34,132	27,258	49,282	58,544	28,279	70,315	42,204	72,784	50,343	43,415		
120	23,54	80,182	30,71	66,164	45, 100	33 ,72	26 ,61	24,59	36, 85	27, 66	9	18
	70 ,363	22,200	21,161	71,365	66,570	49,377	43,285	69, 272	47, 334	25 ,117		
121	60,146	23,56	43,106	79,178	34,75	68,155	56,129	32,79	68,151	25,62	6	18
	66,180	36,102	59,150	62,170	21,59	35,94	21,52	42,122	63,166	63,175		
122	21,51	62,143	58,142	55,132	29,66	39,87	77,179	63,140	63,147	71,162	9	27
	43,107	54,148	31,80	55,142	22,65	21,58	28,74	73,216	49,127	40,101		
123	58,144	32,80	35,79	78,194	70,161	40,99	78,194	56,124	25,58	76,187	7	21
	47,129	22,64	78,225	21,61	50,131	71,213	31,88	26,71	45,119	52,148		

continué sur la page suivante

suite de la page précédente

Inst.	P_{ij}^-, P_{ij}^+										σ, τ_{kk+1}	
124	75,167	26,58	61,139	33,80	60,138	43,104	60,147	44,98	59,130	53,126	8	24
	51,144	67,190	39,115	40,108	45,117	68,173	35,89	24,65	78,232	68,196		
125	75,186	25,62	52,125	43,105	78,186	20,48	59,134	31,68	72,172	41,91	7	21
	47,129	22,64	78,225	21,61	50,131	71,213	31,88	26,71	45,119	52,148		
126	21,51	62,143	58,142	55,132	29,66	39,87	77,179	63,140	63,147	71,162	9	27
	32,160	44,202	50,240	27,165	30,285	72,706	74,482	62,430	70,383	38,210		
127	75,167	26,58	61,139	33, 80	60, 138	43 ,104	60 ,147	44, 98	59, 130	53, 126	8	24
	65, 318	78, 738	27 ,269	80 ,373	64, 637	66, 639	37, 154	42,461	31, 237	78, 687		
128	65,153	52,121	65,146	73,177	24,58	38,91	43,98	74,185	26,58	57,132	7	21
	78,694	64,325	68,620	37,117	58,264	49,233	75,803	55,570	21,108	24,165		
129	60,146	23,56	43,106	79,178	34,75	68,155	56,129	32,79	68,151	25,62	6	18
	54,366	28,291	46,397	31,285	27,105	64,227	60,394	25,119	73,260	62,238		
130	60,146	23,56	43,106	79,178	34,75	68,155	56,129	32,79	68,151	25,62	6	18
	61,558	20,153	76,700	77,246	77,238	66,348	39,335	54,396	69,531	76,609		

Annexe G

Données de [Paul et al., 2007]

Ces instances consistent en 14 problème à 40 jobs. Il y 18 cuves dans la ligne de production (voir tableau G.1). Ces instances sont crées à partir de quatre types de jobs : CAA, YAL, SAD et SAS (voir tableau G.2). Les 14 instances se composent à partir les quatre types de jobs avec différentes rations (voir tableau G.3). MPS dans le tableau est le *minimal part set*. Par exemple, une production composée de (68% CAA, 16% YAL, 8% SAD, 8% SAS) pour un ensemble de 12 jobs correspond approximativement à un MPS (8,2,1,1). Pour un MPS (a_1, a_2, a_3, a_4), la valeur γ qui correspond la variation d'instance, est calculée par $\gamma = 1/n \times \sum_i (a_i - \sum_j a_j/n)^2$. Les temps de déplacement en charge et à vide entre deux cuves M_i et M_j égalent $25 + 2 \times |i - j|$ (second), et $2 \times |i - j|$ (second).

TABLE G.1 – La ligne de production des instances de [Paul et al., 2007]

cuves M_i	opération trempe	temps de traitement (min) minimale	maximal
1	*Chromic acid anodising* (CAA)	23	26
2	rinçage 1 de CAA	3	4
3	rinçage 2 de CAA	3	4
4	rinçage 3 de CAA	15	∞
5	séchage	15	∞
6	rinçage circulation	3	∞
7	*yellow alodining* (YAL)	2	3
8	rinçage 1 de YAL	3	4
9	rinçage 2 de YAL	3	4
10	rinçage de scellage 2	2	∞
11	rinçage de scellage 1	2	3
12	scellage	40	15
13	anodisation supplémentaire à l'acide (SAA)	70	73
14	rinçage 1 de SAA	2	3
15	rinçage 2 de SAA	2	∞
16	teinture de scellage (sealing dyeing)	40	45
17	rinçage de teinture	3	∞
18	teinture	15	20

TABLE G.2 – quatre types de jobs des instances de [Paul et al., 2007]

type	séquence de visite des cuves T	totale temps opératoires minimale (min)
CAA	M_1, M_2, M_3, M_6, M_5	62
YAL	M_7, M_8, M_9, M_6, M_5	26
SAD	$M_{13}, M_{14}, M_{15}, M_{18}, M_{17}, M_{16}, M_{11}, M_{10}, M_6, M_5$	154
SAS	$M_{13}, M_{14}, M_{15}, M_{12}, M_{11}, M_{10}, M_6, M_5$	136

TABLE G.3 – Compositions des instances de [Paul et al., 2007]

instance	MPS(CAA,YAL,SAD,SAS)	γ	instance	MPS(CAA,YAL,SAD,SAS)	γ
1	(9,1,1,1)	12,0	8	(6,2,2,2)	3,0
2	(8,2,1,1)	8,5	9	(5,4,2,1)	2,5
3	(7,3,1,1)	6,0	10	(5,3,3,1)	2,0
4	(7,2,2,1)	5,5	11	(5,3,2,2)	1,5
5	(6,4,1,1)	4,5	12	(4,4,2,2)	1,0
6	(5,5,1,1)	4,0	13	(4,3,3,2)	0,5
7	(6,3,2,1)	3,5	14	(3,3,3,3)	0,0

Annexe H

Résultats de SBN pour [Mateo et al., 2002] avec 6 à 10 cuves

Ces résultats sont obtenus par la méthode SBN proposée, sur 100 exécutions. Les meilleurs résultats obtenus sont comparés avec les résultats de référence. Notre programme est écrit en C++, et il est testé sur un CPU de 2,8GHZ (AMD Phenom X2). Les temps d'exécution varient entre une heure et trois heures selon les tailles d'instances. Pour résoudre ces instances, nous avons appliqué la procédure des cas particuliers développée au chapitre 4.

TABLE H.1 – Résultats pour les instances de [Mateo et al., 2002] avec 6 cuves, 21 jobs, un et deux robots [Mateo et al., 2002]

Inst.	Ref. T	*LB*	*UB*	Cmax de 1 robot C_1	$Gap_{LB}(\%)$	$Gap_{UB}(\%)$	Cmax de 2 robots C_2	$Gap(\%)$
601	450	4992	5531	5126	2,68	-7,32	3639	-29,01
602	440	4892	5431	5056	3,35	-6,90	3148	-37,74
603	337	3722	4109	3899	4,76	-5,11	2443	-37,34
604	336	3712	4099	3970	6,95	-3,15	2306	-41,91
605	366	3957	4287	3960	0,08	-7,63	2290	-42,17
606	269	3029	3438	3301	8,98	-3,98	2084	-36,87
607	316	3512	3899	3674	4,61	-5,77	2112	-42,51
608	332	3691	4103	3996	8,26	-2,61	2193	-45,12
609	525	5593	5892	5843	4,47	-0,83	3056	-47,70
610	362	3917	4247	3740	-4,52	-11,94	2330	-37,70
611	461	4949	5270	5206	5,19	-1,21	2708	-47,98
612	596	6464	6967	6539	1,16	-6,14	3435	-47,47
613	621	6675	7140	7085	6,14	-0,77	3295	-53,49
614	606	6496	6895	6726	3,54	-2,45	3675	-45,36
615	662	7056	7455	7106	0,71	-4,68	3601	-49,32
616	499	5446	5892	5494	0,88	-6,75	3146	-42,74
617	581	6314	6817	6785	7,46	-0,47	3318	-51,10
618	420	4643	5106	4706	1,36	-7,83	2755	-41,46
619	567	6135	6600	6077	-0,95	-7,92	3260	-46,36
620	583	6274	6670	6854	9,24	2,76	3388	-50,57
621	739	7922	8347	8079	1,98	-3,21	4496	-44,35
622	510	5506	5884	5776	4,90	-1,84	2779	-51,89
623	548	5814	6049	5589	-3,87	-7,60	4077	-27,05
624	653	6965	7319	6855	-1,58	-6,34	3959	-42,25
625	568	6052	6351	5943	-1,80	-6,42	3351	-43,61
626	586	6276	6600	6365	1,42	-3,56	5080	-20,19
627	477	5176	5554	5026	-2,90	-9,51	2951	-41,29
628	632	6755	7109	7038	4,19	-1,00	3934	-44,10
629	525	5584	5819	5612	0,50	-3,56	3119	-44,42
630	386	4299	4731	4326	0,63	-8,56	2607	-39,74
moyenne					2,59	-4,74		-42,43

$Gap_{LB} = (C_1 - LB)/LB$
$Gap_{UB} = (C_1 - UB)/UB$
$Gap = (C_2 - C_1)/C_1$

TABLE H.2 – Résultats d'instances de [Mateo et al., 2002] avec 7 cuves, 21 jobs, un et deux robots

Inst.	Ref. T	LB	UB	Cmax de 1 robot C_1	$Gap_{LB}(\%)$	$Gap_{UB}(\%)$	Cmax de 2 robots C_2	$Gap(\%)$
701	446	5018	5656	5423	8,07	-4,12	3116	-42,54
702	653	7044	7594	6401	-9,13	-15,71	3660	-42,82
703	690	7300	7667	7181	-1,63	-6,34	4380	-39,01
704	408	4500	4959	4781	6,24	-3,59	2696	-43,61
705	622	6619	7014	6510	-1,65	-7,19	3831	-41,15
706	418	4738	5376	5142	8,53	-4,35	3112	-39,48
707	598	6379	6774	7024	10,11	3,69	3364	-52,11
708	426	4818	5456	5038	4,57	-7,66	2861	-43,21
709	558	6026	6484	5977	-0,81	-7,82	4223	-29,35
710	558	6026	6484	6105	1,31	-5,85	3552	-41,82
711	635	6801	7267	6476	-4,78	-10,88	3390	-47,65
712	578	6226	6684	5962	-4,24	-10,80	3081	-48,32
713	370	4164	4665	4290	3,03	-8,04	2687	-37,37
714	360	4064	4565	4255	4,70	-6,79	2539	-40,33
715	579	6241	6707	6732	7,87	0,37	3440	-48,90
716	600	6451	6917	6296	-2,40	-8,98	3388	-46,19
717	452	4962	5425	5007	0,91	-7,71	2844	-43,20
718	539	5836	6294	5712	-2,12	-9,25	3183	-44,28
719	775	8267	8663	8801	6,46	1,59	6063	-31,11
720	460	5112	5643	5051	-1,19	-10,49	2918	-42,23
721	788	8448	8954	8490	0,50	-5,18	4699	-44,65
722	784	8408	8914	8462	0,64	-5,07	4701	-44,45
723	778	8306	8735	8559	3,05	-2,01	4795	-43,98
724	928	9858	10317	9699	-1,61	-5,99	5137	-47,04
725	658	7043	7447	7548	7,17	1,36	4603	-39,02
726	741	7927	8323	9123	15,09	9,61	4903	-46,26
727	724	7808	8314	7925	1,50	-4,68	4418	-44,25
728	438	4902	5439	5108	4,20	-6,09	2921	-42,82
moyenne					2,30	-5,28		42,75

$Gap_{LB} = (C_1 - LB)/LB$
$Gap_{UB} = (C_1 - UB)/UB$
$Gap = (C_2 - C_1)/C_1$

TABLE H.3 – Résultats d'instances de [Mateo et al., 2002] avec 8 cuves, 21 jobs, un et deux robots

Inst.	Ref. T	LB	UB	Cmax de 1 robot C_1	$Gap_{LB}(\%)$	$Gap_{UB}(\%)$	Cmax de 2 robots C_2	$Gap(\%)$
801	412	4489	4910	4734	5,46	-3,58	2397	-49,37
802	585	6389	6993	7033	10,08	0,57	3508	-50,12
803	605	6506	6971	6620	1,75	-5,04	3824	-42,24
804	390	4310	4783	4942	14,66	3,32	2654	-46,30
805	583	6312	6839	7381	16,94	7,93	4335	-41,27
806	645	6965	7505	7457	7,06	-0,64	3838	-48,53
807	680	7321	7875	7484	2,23	-4,97	4574	-38,88
808	579	6289	6826	5987	-4,80	-12,29	3161	-47,20
809	573	6229	6766	6991	12,23	3,33	3157	-54,84
810	582	6300	6810	6550	3,97	-3,82	3535	-46,03
811	563	6074	6532	7077	16,51	8,34	3509	-50,42
812	776	8307	8859	8391	1,01	-5,28	4780	-43,03
813	676	7220	7657	7990	10,66	4,35	4097	-48,72
814	570	6199	6729	6759	9,03	0,45	3411	-49,53
815	627	6730	7167	7184	6,75	0,24	4314	-39,95
816	805	8583	9058	9882	15,13	9,10	7015	-29,01
817	490	5344	5814	5496	2,84	-5,47	3391	-38,30
818	410	4544	5014	5060	11,36	0,92	3718	-26,52
819	712	7696	8277	7433	-3,42	-10,20	4688	-36,93
820	1065	11331	11922	11711	3,35	-1,77	8047	-31,29
821	1066	11341	11932	11641	2,65	-2,44	6100	-47,60
822	831	8937	9471	9337	4,48	-1,41	5096	-45,42
823	922	9963	10689	10182	2,20	-4,74	5555	-45,44
824	809	8627	9061	8755	1,48	-3,38	5263	-39,89
825	878	9523	10249	9508	-0,16	-7,23	4901	-48,45
826	515	5722	6307	6249	9,21	-0,92	3801	-39,17
827	980	10481	11072	10478	-0,03	-5,36	7752	-26,02
828	972	10401	10992	10414	0,12	-5,26	5630	-45,94
829	627	6896	7502	7244	5,05	-3,44	4052	-44,06
moyenne					5,79	-1,68		-42,78

$Gap_{LB} = (C_1 - LB)/LB$
$Gap_{UB} = (C_1 - UB)/UB$
$Gap = (C_2 - C_1)/C_1$

TABLE H.4 – Résultats d'instances de [Mateo et al., 2002] avec 9 cuves, 21 jobs, un et deux robots

Inst.	Ref. T	LB	UB	Cmax de 1 robot C_1	$Gap_{LB}(\%)$	$Gap_{UB}(\%)$	Cmax de 2 robots C_2	$Gap(\%)$
901	794	8460	9000	8926	5,51	-0,82	4811	-46,10
902	799	8564	9167	8952	4,53	-2,35	5089	-43,15
903	711	7651	8265	7897	3,22	-4,45	3778	-52,16
904	793	8499	9093	8728	2,69	-4,01	4767	-45,38
905	692	7461	8075	7876	5,56	-2,46	3861	-50,98
906	660	7141	7755	7034	-1,50	-9,30	4004	-43,08
907	773	8304	8907	8898	7,15	-0,10	5071	-43,01
908	569	6127	6613	7183	17,24	8,62	4143	-42,32
909	723	7799	8393	8444	8,27	0,61	4682	-44,55
910	568	6117	6603	7081	15,76	7,24	3601	-49,15
911	793	8420	8901	8627	2,46	-3,08	4265	-50,56
912	697	7560	8188	8242	9,02	0,66	3975	-51,77
913	712	7739	8401	6828	-11,77	-18,72	3565	-47,79
914	852	9092	9655	8978	-1,25	-7,01	5073	-43,50
915	786	8463	9054	9010	6,46	-0,49	5039	-44,07
916	929	9977	10607	10147	1,70	-4,34	5488	-45,92
917	759	8194	8789	9059	10,56	3,07	5139	-43,27
918	768	8284	8879	8502	2,63	-4,25	5076	-40,30
919	600	6619	7281	6658	0,59	-8,56	3433	-48,44
920	991	10638	11258	10879	2,27	-3,37	8017	-26,31
921	1026	11060	11795	12720	15,01	7,84	6441	-49,36
922	919	9835	10425	9774	-0,62	-6,24	5452	-44,22
923	628	6939	7644	8320	19,90	8,84	3791	-54,44
924	1056	11288	11908	11496	1,84	-3,46	6543	-43,08
925	987	10670	11405	10497	-1,62	-7,96	5594	-46,71
926	849	9181	9861	10653	16,03	8,03	5273	-50,50
927	732	7975	8614	8712	9,24	1,14	4593	-47,28
928	847	9161	9841	9388	2,48	-4,60	5299	-43,56
929	971	10438	11058	10396	-0,40	-5,99	6546	-37,03
moyenne					5,27	-1,91		-45,45

$Gap_{LB} = (C_1 - LB)/LB$
$Gap_{UB} = (C_1 - UB)/UB$
$Gap = (C_2 - C_1)/C_1$

TABLE H.5 – Résultats d'instances de [Mateo et al., 2002] avec 10 cuves, 21 jobs, un et deux robots

Inst.	Ref. T	*LB*	*UB*	Cmax de 1 robot C_1	$Gap_{LB}(\%)$	$Gap_{UB}(\%)$	Cmax de 2 robots C_2	$Gap(\%)$
101	737	7951	8599	8184	2,93	-4,83	3697	-54,83
102	848	9184	9966	9582	4,33	-3,85	4347	-54,63
103	875	9340	9946	9734	4,22	-2,13	5582	-42,65
104	918	9770	10376	9664	-1,08	-6,86	5675	-41,28
105	761	8166	8759	8370	2,50	-4,44	4078	-51,28
106	652	7101	7749	8038	13,20	3,73	3647	-54,63
107	910	9760	10400	10083	3,31	-3,05	5681	-43,66
108	781	8514	9296	9251	8,66	-0,48	4156	-55,08
109	456	5224	5990	6337	21,31	5,79	3552	-43,95
110	797	8560	9166	9276	8,36	1,20	5424	-41,53
111	952	10203	10857	10189	-0,14	-6,15	5816	-42,92
112	1032	11106	11913	11195	0,80	-6,03	6314	-43,60
113	781	8463	9161	8818	4,19	-3,74	4665	-47,10
114	911	9793	10499	10073	2,86	-4,06	4931	-51,05
115	1056	11278	11941	12248	8,60	2,57	6366	-48,02
116	756	8213	8911	8384	2,08	-5,91	4195	-49,96
117	632	6874	7463	8146	18,50	9,15	3806	-53,28
118	805	8733	9439	8504	-2,62	-9,91	4818	-43,34
119	1038	11098	11761	11728	5,68	-0,28	8639	-26,34
120	916	9748	10272	10958	12,41	6,68	7441	-32,10
121	911	9796	10445	9835	0,40	-5,84	5676	-42,29
122	1106	11895	12606	11869	-0,22	-5,85	8971	-24,42
123	1005	10829	11601	11849	9,42	2,14	6194	-47,73
124	1062	11398	12071	11508	0,97	-4,66	6479	-43,70
125	960	10327	11008	10679	3,41	-2,99	5884	-44,90
126	1093	11765	12476	11947	1,55	-4,24	7429	-37,82
127	1004	10818	11491	11795	9,03	2,65	6183	-47,58
128	948	10228	10930	11171	9,22	2,20	5645	-49,47
129	821	8896	9545	10381	16,69	8,76	6210	-40,18
moyenne					5,88	-1,39		-44,80

$Gap_{LB} = (C_1 - LB)/LB$
$Gap_{UB} = (C_1 - UB)/UB$
$Gap = (C_2 - C_1)/C_1$

Annexe I

Résultats de GTSB pour [Mateo et al., 2002] avec 6 à 10 cuves

Ces résultats sont obtenus par la méthode GTSB proposée avec une taille de population de 10, un nombre maximum de générations de 10, un nombre d'itérations tabou de 50. Nous montrons les meilleures solutions obtenues en trois heures. Notre programme est écrit en C++, et il est testé sur un CPU de 2,8GHZ (AMD Phenom X2). Les temps d'exécution varient entre trois et cinq heures selon les tailles d'instances. Comparés avec les résultats obtenus par SBN en appliquant la procédure des cas particuliers. Il y a donc beaucoup moins de solutions explorées par GTSB. Cela explique pourquoi les résultats montrés ci-dessous sont moins bons que ceux de l'annexe H.

TABLE I.1 – Résultats pour les instances de [Mateo et al., 2002] avec 6 cuves, 21 jobs, un et deux robots [Mateo et al., 2002]

Inst.	Ref. T	LB	UB	Cmax de 1 robot C_1	$Gap_{LB}(\%)$	$Gap_{UB}(\%)$	Cmax de 2 robots C_2	$Gap(\%)$
601	450	4992	5531	5249	5,15	-5,10	3703	-29,45
602	440	4892	5431	4938	0,94	-9,08	3718	-24,71
603	337	3722	4109	4287	15,18	4,33	2994	-30,16
604	336	3712	4099	4031	8,59	-1,66	3056	-24,19
605	366	3957	4287	4123	4,20	-3,83	2970	-27,97
606	269	3029	3438	3306	9,14	-3,84	2692	-18,57
607	316	3512	3899	3714	5,75	-4,74	2701	-27,28
608	332	3691	4103	4005	8,51	-2,39	2666	-33,43
609	525	5593	5892	5473	-2,15	-7,11	4307	-21,30
610	362	3917	4247	3737	-4,60	-12,01	2838	-24,06
611	461	4949	5270	4792	-3,17	-9,07	4004	-16,44
612	596	6464	6967	6967	7,78	0,00	4062	-41,70
613	621	6675	7140	7482	12,09	4,79	5673	-24,18
614	606	6496	6895	7086	9,08	2,77	4951	-30,13
615	662	7056	7455	7564	7,20	1,46	5279	-30,21
616	499	5446	5892	6032	10,76	2,38	3825	-36,59
617	581	6314	6817	6703	6,16	-1,67	4979	-25,72
618	420	4643	5106	4728	1,83	-7,40	3436	-27,33
619	567	6135	6600	6179	0,72	-6,38	3312	-46,40
620	583	6274	6670	6625	5,59	-0,67	4706	-28,97
621	739	7922	8347	8134	2,68	-2,55	6028	-25,89
622	510	5506	5884	6170	12,06	4,86	2747	-55,48
623	548	5814	6049	6446	10,87	6,56	4059	-37,03
624	653	6965	7319	6624	-4,90	-9,50	5436	-17,93
625	568	6052	6351	5943	-1,80	-6,42	3345	-43,72
626	586	6276	6600	6492	3,44	-1,64	3893	-40,03
627	477	5176	5554	5850	13,02	5,33	3320	-43,25
628	632	6755	7109	6783	0,41	-4,59	5286	-22,07
629	525	5584	5819	5443	-2,53	-6,46	4344	-20,19
630	386	4299	4731	4527	5,30	-4,31	3378	-25,38
moyenne					4,91	-2,60		-29,99

$Gap_{LB} = (C_1 - LB)/LB$
$Gap_{UB} = (C_1 - UB)/UB$
$Gap = (C_2 - C_1)/C_1$

TABLE I.2 – Résultats d'instances de [Mateo et al., 2002] avec 7 cuves, 21 jobs, un et deux robots

Inst.	Ref. T	LB	UB	Cmax de 1 robot C_1	$Gap_{LB}(\%)$	$Gap_{UB}(\%)$	Cmax de 2 robots C_2	$Gap(\%)$
701	446	5018	5656	5287	5,36	-6,52	4193	-20,69
702	653	7044	7594	7028	-0,23	-7,45	5427	-22,78
703	690	7300	7667	7181	-1,63	-6,34	5713	-20,44
704	408	4500	4959	4969	10,42	0,20	3308	-33,43
705	622	6619	7014	6510	-1,65	-7,19	4916	-24,49
706	418	4738	5376	4651	-1,84	-13,49	4018	-13,61
707	598	6379	6774	7024	10,11	3,69	4682	-33,34
708	426	4818	5456	5321	10,44	-2,47	3445	-35,26
709	558	6026	6484	6111	1,41	-5,75	4223	-30,90
710	558	6026	6484	6333	5,09	-2,33	4319	-31,80
711	635	6801	7267	6672	-1,90	-8,19	3839	-42,46
712	578	6226	6684	6116	-1,77	-8,50	4715	-22,91
713	370	4164	4665	4477	7,52	-4,03	3611	-19,34
714	360	4064	4565	4162	2,41	-8,83	3327	-20,06
715	579	6241	6707	7073	13,33	5,46	4068	-42,49
716	600	6451	6917	6861	6,36	-0,81	5197	-24,25
717	452	4962	5425	5102	2,82	-5,95	3703	-27,42
718	539	5836	6294	5749	-1,49	-8,66	3762	-34,56
719	775	8267	8663	8801	6,46	1,59	6430	-26,94
720	460	5112	5643	5363	4,91	-4,96	4039	-24,69
721	788	8448	8954	8459	0,13	-5,53	6555	-22,51
722	784	8408	8914	8336	-0,86	-6,48	6374	-23,54
723	778	8306	8735	10110	21,72	15,74	6401	-36,69
724	928	9858	10317	10489	6,40	1,67	7031	-32,97
725	658	7043	7447	7566	7,43	1,60	5980	-20,96
726	741	7927	8323	9690	22,24	16,42	6362	-34,34
727	724	7808	8314	8158	4,48	-1,88	6284	-22,97
728	438	4902	5439	5368	9,51	-1,31	3838	-28,50
moyenne					5,26	-2,51		-27,65

$Gap_{LB} = (C_1 - LB)/LB$
$Gap_{UB} = (C_1 - UB)/UB$
$Gap = (C_2 - C_1)/C_1$

TABLE I.3 – Résultats d'instances de [Mateo et al., 2002] avec 8 cuves, 21 jobs, un et deux robots

Inst.	Ref. T	LB	UB	Cmax de 1 robot C_1	$Gap_{LB}(\%)$	$Gap_{UB}(\%)$	Cmax de 2 robots C_2	$Gap(\%)$
801	412	4489	4910	4955	10,38	0,92	3462	-30,13
802	585	6389	6993	7309	14,40	4,52	4146	-43,28
803	605	6506	6971	6735	3,52	-3,39	5168	-23,27
804	390	4310	4783	4961	15,10	3,72	3396	-31,55
805	583	6312	6839	7309	15,80	6,87	4258	-41,74
806	645	6965	7505	7941	14,01	5,81	4582	-42,30
807	680	7321	7875	7698	5,15	-2,25	6215	-19,26
808	579	6289	6826	6310	0,33	-7,56	4266	-32,39
809	573	6229	6766	6666	7,02	-1,48	4939	-25,91
810	582	6300	6810	7038	11,71	3,35	4164	-40,84
811	563	6074	6532	7278	19,82	11,42	4004	-44,98
812	776	8307	8859	8157	-1,81	-7,92	6610	-18,97
813	676	7220	7657	6883	-4,67	-10,11	5598	-18,67
814	570	6199	6729	7266	17,21	7,98	4109	-43,45
815	627	6730	7167	7166	6,48	-0,01	5787	-19,24
816	805	8583	9058	9525	10,98	5,16	6849	-28,09
817	490	5344	5814	5303	-0,77	-8,79	4322	-18,50
818	410	4544	5014	1714	-62,28	-65,82	3432	100,23
819	712	7696	8277	9274	20,50	12,05	6772	-26,98
820	1065	11331	11922	12941	14,21	8,55	8249	-36,26
821	1066	11341	11932	13004	14,66	8,98	8073	-37,92
822	831	8937	9471	8949	0,13	-5,51	6929	-22,57
823	922	9963	10689	10630	6,69	-0,55	7231	-31,98
824	809	8627	9061	9521	10,36	5,08	6946	-27,05
825	878	9523	10249	9526	0,03	-7,05	7715	-19,01
826	515	5722	6307	5842	2,10	-7,37	4291	-26,55
827	980	10481	11072	11849	13,05	7,02	7887	-33,44
828	972	10401	10992	10826	4,09	-1,51	7761	-28,31
829	627	6896	7502	7011	1,67	-6,54	4741	-32,38
moyenne					5,86	-1,53		-25,68

$Gap_{LB} = (C_1 - LB)/LB$
$Gap_{UB} = (C_1 - UB)/UB$
$Gap = (C_2 - C_1)/C_1$

TABLE I.4 – Résultats d'instances de [Mateo et al., 2002] avec 9 cuves, 21 jobs, un et deux robots

Inst.	Ref. T	LB	UB	Cmax de 1 robot C_1	$Gap_{LB}(\%)$	$Gap_{UB}(\%)$	Cmax de 2 robots C_2	$Gap(\%)$
901	794	8460	9000	8502	0,50	-5,53	6542	-23,05
902	799	8564	9167	8585	0,25	-6,35	6494	-24,36
903	711	7651	8265	7814	2,13	-5,46	4535	-41,96
904	793	8499	9093	8473	-0,31	-6,82	6619	-21,88
905	692	7461	8075	7705	3,27	-4,58	6290	-18,36
906	660	7141	7755	7034	-1,50	-9,30	4304	-38,81
907	773	8304	8907	8539	2,83	-4,13	5089	-40,40
908	569	6127	6613	7183	17,24	8,62	4213	-41,35
909	723	7799	8393	8380	7,45	-0,15	6688	-20,19
910	568	6117	6603	7078	15,71	7,19	4161	-41,21
911	793	8420	8901	8342	-0,93	-6,28	6423	-23,00
912	697	7560	8188	8242	9,02	0,66	4499	-45,41
913	712	7739	8401	8248	6,58	-1,82	4700	-43,02
914	852	9092	9655	9165	0,80	-5,08	6947	-24,20
915	786	8463	9054	8661	2,34	-4,34	7386	-14,72
916	929	9977	10607	10128	1,51	-4,52	7179	-29,12
917	759	8194	8789	8111	-1,01	-7,71	6648	-18,04
918	768	8284	8879	8502	2,63	-4,25	7274	-14,44
919	600	6619	7281	6992	5,64	-3,97	4300	-38,50
920	991	10638	11258	10606	-0,30	-5,79	8240	-22,31
921	1026	11060	11795	13720	24,05	16,32	6512	-52,54
922	919	9835	10425	12365	25,72	18,61	7478	-39,52
923	628	6939	7644	8364	20,54	9,42	4993	-40,30
924	1056	11288	11908	11397	0,97	-4,29	8835	-22,48
925	987	10670	11405	10497	-1,62	-7,96	6320	-39,79
926	849	9181	9861	10653	16,03	8,03	7941	-25,46
927	732	7975	8614	8712	9,24	1,14	5259	-39,63
928	847	9161	9841	9388	2,48	-4,60	5876	-37,41
929	971	10438	11058	10396	-0,40	-5,99	8754	-15,79
moyenne					5,89	-1,34		-30,94

$Gap_{LB} = (C_1 - LB)/LB$
$Gap_{UB} = (C_1 - UB)/UB$
$Gap = (C_2 - C_1)/C_1$

TABLE I.5 – Résultats d'instances de [Mateo et al., 2002] avec 10 cuves, 21 jobs, un et deux robots

Inst.	Ref. T	*LB*	*UB*	Cmax de 1 robot C_1	$Gap_{LB}(\%)$	$Gap_{UB}(\%)$	Cmax de 2 robots C_2	$Gap(\%)$
101	737	7951	8599	8125	2,19	-5,51	6892	-15,18
102	848	9184	9966	9582	4,33	-3,85	7609	-20,59
103	875	9340	9946	9346	0,06	-6,03	7662	-18,02
104	918	9770	10376	9581	-1,93	-7,66	7587	-20,81
105	761	8166	8759	8232	0,81	-6,02	6919	-15,95
106	652	7101	7749	8119	14,34	4,77	4503	-44,54
107	910	9760	10400	10103	3,51	-2,86	8068	-20,14
108	781	8514	9296	9359	9,92	0,68	5170	-44,76
109	456	5224	5990	6267	19,97	4,62	4044	-35,47
110	797	8560	9166	8833	3,19	-3,63	7039	-20,31
111	952	10203	10857	9853	-3,43	-9,25	5755	-41,59
112	1032	11106	11913	11195	0,80	-6,03	6314	-43,60
113	781	8463	9161	7818	-7,62	-14,66	4743	-39,33
114	911	9793	10499	10023	2,35	-4,53	4952	-50,59
115	1056	11278	11941	13849	22,80	15,98	8801	-36,45
116	756	8213	8911	9866	20,13	10,72	5278	-46,50
117	632	6874	7463	7865	14,42	5,39	6000	-23,71
118	805	8733	9439	9753	11,68	3,33	7477	-23,34
119	1038	11098	11761	14028	26,40	19,28	8686	-38,08
120	916	9748	10272	9862	1,17	-3,99	7681	-22,12
121	911	9796	10445	9706	-0,92	-7,08	7748	-20,17
122	1106	11895	12606	11869	-0,22	-5,85	8955	-24,55
123	1005	10829	11601	14985	38,38	29,17	8037	-46,37
124	1062	11398	12071	11115	-2,48	-7,92	9251	-16,77
125	960	10327	11008	10549	2,15	-4,17	8408	-20,30
126	1093	11765	12476	11947	1,55	-4,24	9872	-17,37
127	1004	10818	11491	11795	9,03	2,65	9382	-20,46
128	948	10228	10930	10638	4,01	-2,67	8862	-16,69
129	821	8896	9545	8815	-0,91	-7,65	6127	-30,49
moyenne					6,75	-0,59		-28,77

$Gap_{LB} = (C_1 - LB)/LB$
$Gap_{UB} = (C_1 - UB)/UB$
$Gap = (C_2 - C_1)/C_1$

Bibliographie

[cht, 2010] (2010). *Solving cyclic scheduling of two hoists in electroplating line*.

[Abdelmaguid et al., 2004] Abdelmaguid, T., Nassef, A., Kamal, B., and Hassan, M. (2004). A hybrid GA/heuristic approach to the simultaneous scheduling of machines and automated guided vehicles. *International Journal of Production Research*, 42(2) :267–281.

[Adams et al., 1988] Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management science*, pages 391–401.

[Agnetis, 2000] Agnetis, A. (2000). Scheduling no-wait robotic cells with two and three machines. *European Journal of Operational Research*, 123(2) :303–314.

[Andrea and Gino, 2007] Andrea, R. and Gino, D. (2007). Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimization method. *Robotics and Computer-Integrated Manufacturing*, 23 :503–516.

[Anwar and Nagi, 1998] Anwar, M. and Nagi, R. (1998). Integrated scheduling of material handling and manufacturing activities for just-in-time production of complex assemblies. *International journal of production research*, 36(3) :653–681.

[Armstrong et al., 1996] Armstrong, R., Gu, S., and Lei, L. (1996). A greedy algorithm to determine the number of transporters in a cyclic electroplating process. *IIE transactions*, 28(5) :347–355.

[Baptiste et al., 1993] Baptiste, P., Legeard, B., Manier, M., and Varnier, C. (1993). Optimization with constraint logic programming : the hoist scheduling problem solved with various solvers. In *Applications of Artificial Intelligence in Engineering*, pages 599–614.

[Bilge and Tanchoco, 1997] Bilge, Ü. and Tanchoco, J. (1997). Agv systems with multi-load carriers : basic issues and potential benefits. *Journal of manufacturing systems*, 16(3) :159–174.

[Bilge and Ulusoy, 1995] Bilge, Ü. and Ulusoy, G. (1995). A time window approach to simultaneous scheduling of machines and material handling system in an FMS. *Operations Research*, 43(6) :1058–1070.

[Billaut et al., 1997] Billaut, J., Tacquard, C., and Martineau, P. (1997). Modeling fms

scheduling problems as hybrid flowshop scheduling problems. *Studies in Informatics and Control*, 6(1) :25–30.

[Bloch et al., 2008] Bloch, C., Manier, M.-A., Baptiste, P., and Varnier, C. (2008). *Production Scheduling, Hoist Scheduling Problem*. CAM - Control Systems, Robotics and Manufacturing Series. Pierre Lopez and François Roubellat.

[Bonhomme et al., 2002] Bonhomme, P., Aygalinc, P., and Calvez, S. (2002). Firing instant approach to control time critical systems in multi-product processing. In *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, pages 97–102. IEEE.

[Bozer and Srinivasan, 1992] Bozer, Y. and Srinivasan, M. (1992). Tandem agv systems : a partitioning algorithm and performance comparison with conventional agv systems. *European Journal of Operational Research*, 63(2) :173–191.

[Brauner, 2006] Brauner, N. (2006). Identical part production in cyclic robotic cells : a state of the art. *Internal note, Laboratoire Leibniz, Institut IMAG, Grenoble, France*.

[Brauner et al., 2005] Brauner, N., Castagna, P., NewAuthor1, Finke, G., Lacomme, P., Martineau, P., Moukrim, A., Soukhal, A., Tacquard, C., and Tchernev, N. (2005). Ordonnancement dans les systèmes flexibles de production. *Journal Européen des Systèmes Automatisés*, 39(8) :925–964.

[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(1-3) :107–127.

[Brucker and Kampmeyer, 2008] Brucker, P. and Kampmeyer, T. (2008). A general model for cyclic machine scheduling problems. *Discrete Applied Mathematics*, 156(13) :2561–2572.

[Carlier and Chrétienne, 1988] Carlier, J. and Chrétienne, P. (1988). *Problèmes d'ordonnancement (modélisation / complexité / algorithmes)*. Masson.

[Caumond et al., 2009] Caumond, A., Lacomme, P., Moukrim, A., and Tchernev, N. (2009). An MILP for Scheduling Problems in an FMS with one Vehicle. *European Journal of Operational Research*, 199(3) :706–722.

[Caux et al., 1992] Caux, C., Fleury, G., Gourgand, M., and Kellert, P. (1992). Couplage simulation à évènements discrets-méthodes stochastiques pour l'ordonnancement d'un atelier de traitement de surface. In *Proceedings of the European Simulation Multiconference*, pages 1–3.

[Caux et al., 1995] Caux, C., Fleury, G., Gourgand, M., and Kellert, P. (1995). Couplage méthodes d'ordonnancement-simulation pour l'ordonnancement de systèmes industriels de traitement de surface. *RAIRO. Recherche opérationnelle*, 29(4) :391–413.

[Cavory et al., 2005] Cavory, G., Dupas, R., and Goncalves, G. (2005). A genetic approach to solving the problem of cyclic job shop scheduling with linear constraints. *European Journal of Operational Research*, 161(1) :73–85.

[Che et al., 2002] Che, A., Chu, C., and Chu, F. (2002). Multicyclic hoist scheduling with constant processing times. *IEEE Transactions on Robotics and Automation*, 18(1) :69–80.

[Che et al., 2003] Che, A., Chu, C., and Levner, E. (2003). A polynomial algorithm for 2-degree cyclic robot scheduling. *European Journal of Operational Research*, 145(1) :31–44.

[Chen et al., 2002] Chen, H., Chu, C., and Proth, J. (2002). Cyclic scheduling of a hoist with time window constraints. *IEEE Transactions on Robotics and Automation*, 14(1) :144–152.

[Choi et al., 2009] Choi, J., Oh, S., and Pedrycz, W. (2009). Identification of fuzzy relation models using hierarchical fair competition-based parallel genetic algorithms and information granulation. *Applied Mathematical Modelling*, 33(6) :2791–2807.

[Collart Dutilleul and Denat, 1998] Collart Dutilleul, S. and Denat, J. (1998). P-time petri nets and the hoist scheduling problem. In *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, volume 1, pages 558–563. IEEE.

[Corréa et al., 2007] Corréa, A. I., Langevin, A., and Rousseau, L.-M. (2007). Scheduling and routing of automated guided vehicles : A hybrid approach. *Computers & Operations Research*, 34(6) :1688–1707. Part Special Issue : Odysseus 2003 Second International Workshop on Freight Transportation Logistics.

[Crama et al., 2000] Crama, Y., Kats, V., de Klundert, J. V., and Levner, E. (2000). Cyclic scheduling in robotic flowshops. *Annals of Operations Research*, 96(1) :97–124.

[Crama and Van de Klundert, 1997a] Crama, Y. and Van de Klundert, J. (1997a). Cyclic scheduling of identical parts in a robotic cell. *Operations Research*, 45(6) :952–965.

[Crama and Van de Klundert, 1997b] Crama, Y. and Van de Klundert, J. (1997b). Robotic flowshop scheduling is strongly np-complete. Technical Report 010, Maastricht : METEOR, Maastricht Research School of Economics of Technology and Organization.

[Dawande et al., 2005] Dawande, M., Geismar, H., Sethi, S., and Sriskandarajah, C. (2005). Sequencing and scheduling in robotic cells : Recent developments. *Journal of Scheduling*, 8(5) :387–426.

[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.

[Deroussi and Norre, 2010] Deroussi, L. and Norre, S. (2010). Simultaneous scheduling of machines and vehicles for the flexible job shop problem. In *International Conference on Metaheuristics and Nature Inspired Computing*.

[Drobouchevitch et al., 2006] Drobouchevitch, I., Sethi, S., and Sriskandarajah, C. (2006). Scheduling dual gripper robotic cell : One-unit cycles. *European journal of operational research*, 171(2) :598–631.

[Eiben et al., 1994] Eiben, A., Raué, P., and Ruttkay, Z. (1994). Genetic algorithms with multi-parent recombination. *Parallel Problem Solving from Natureâ PPSN III*, pages 78–87.

[El Amraoui et al., 2012] El Amraoui, A., Manier, M.-A., El Moudni, A., and Benrejeb, M. (2012). Resolution of the two-part cyclic hoist scheduling problem with bounded processing times in complex lines. *European Journal of Industrial Engineering (EJIE)*, 6(4).

[Elmi et al., 2011] Elmi, A., Solimanpur, M., Topaloglu, S., and Elmi, A. (2011). A simulated annealing algorithm for the job shop cell scheduling problem with intercellular moves and reentrant parts. *Computers & Industrial Engineering*.

[Esquirol et al., 1999] Esquirol, P., Lopez, P., and (19..-.... professeur de robotique) Lopez, P. (1999). *L'ordonnancement*. Économica.

[Fiedler and Meyer, 2008] Fiedler, C. and Meyer, W. (2008). Deadlock-and collision-control in robotic flow shops. *Lecture Notes in Engineering and Computer Science*, 2173.

[Finke et al., 1996] Finke, G., Gueguen, C., and Brauner, N. (1996). Robotic cells with buffer space. In *ECCO IX Conference Proceedings*. Dublin City University.

[Fleury, 1995] Fleury, G. (1995). Applications de méthodes stochastiques inspirées du recuit simulé à des problèmes d'ordonnancement. *Automatique-productique informatique industrielle*, 29(4-5) :445–470.

[Fogel et al., 1996] Fogel, L., Owens, A., and Walsh, M. (1996). *Genetic Algorithms in Search, Optimization and Machine Learning*. Wiley and sons. NY.

[Fondrevelle, 2005] Fondrevelle, J. (2005). *Résolution exacte de problèmes d'ordonnancement de type flowshop de permutation en présence de contraintes d'écarts temporels entre opérations*. PhD thesis, Doctorat en Informatique, Institut National de Polytechnique de Lorraine.

[Gambardella and Mastrolilli, 1996] Gambardella, M. and Mastrolilli, M. (1996). Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(3).

[Geismar et al., 2003] Geismar, H., Dawande, M., and Sriskandarajah, C. (2003). Scheduling constant travel-time dual gripper robotic cells with parallel machines. *under review at Operations Research*.

[Geismar et al., 2006] Geismar, H., Dawande, M., and Sriskandarajah, C. (2006). Throughput optimization in constant travel-time dual gripper robotic cells with parallel machines. *Production and Operations Management*, 15(2) :311–328.

[ghazali Talbi, 1999] ghazali Talbi, E. (1999). Métaheuristiques pour l'optimisation combinatoire multi-objectif : Etat de l'art.

[Giffler and Thompson, 1960] Giffler, J. and Thompson, G. (1960). Algorithms for solving production scheduling problems. *Operations Research*, 8(4) :487–503.

[Glover, 1990] Glover, F. (1990). Tabu search : A tutorial. *Interfaces*, 20(4) :74–94.

[Goldberg, 1989] Goldberg, D. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-wesley.

[Gondran and Minoux, 1985] Gondran, M. and Minoux, M. (1985). *Graphes et algorithmes*. Eyrolles Paris.

[Gottlieb and Paulmann, 1998] Gottlieb, J. and Paulmann, L. (1998). Genetic algorithms for the fixed charge transportation problem. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 330–335. IEEE.

[Graham et al., 1979] Graham, R., Lawler, E., Lenstra, J., and Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of discrete Mathematics*, 5(2) :287–326.

[Gultekin et al., 2007] Gultekin, H., Akturk, M., and Karasan, O. E. (2007). Scheduling in a three-machine robotic flexible manufacturing cell. *Computers & operations research*, 34(8) :2463–2477.

[Hall et al., 1998] Hall, N. G., Kamounb, H., and Sriskandarajah, C. (1998). Scheduling in robotic cells : Complexity and steady state analysis. *European Journal of Operational Research*, 109-1 :43–65.

[Heragu and Kusiak, 1988] Heragu, S. and Kusiak, A. (1988). Machine layout problem in flexible manufacturing systems. *Operations Research*, 36(2) :258–268.

[Hertz et al., 1996] Hertz, A., Mottet, Y., and Rochat, Y. (1996). On a scheduling problem in a robotized analytical system. *Discrete applied mathematics*, 65(1) :285–318.

[Holland, 1992] Holland, J. (1992). Genetic algorithms. *Scientific American*, 267(1) :66–72.

[Hurink and 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 and 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.

[Johnson and Brandeau, 1993] Johnson, M. and Brandeau, M. (1993). An analytic model for design of a multivehicle automated guided vehicle system. *Management Science*, pages 1477–1489.

[Kamalabadi et al., 2008] Kamalabadi, N., Gholami, S., and Mirzaei, A. (2008). A new solution for the cyclic multiple-part type three-machine robotic cell problem based on

the particle swarm meta-heuristic. *Journal of Industrial and Systems Engineering*, 1(4) :304–317.

[Knust, 1999] Knust, S. (1999). *Shop-Scheduling problem with transportation*. PhD thesis, Fachbereich Mathematik/Informatik Universitat Osnabruck.

[Kuntay et al., 2006] Kuntay, I., Xu, Q., Uygun, K., and Huang, Y. (2006). Environmentally conscious hoist scheduling for electroplating facilities. *Chemical Engineering Communications*, 193(3) :273–292.

[Lacomme, 1998] Lacomme, P. (1998). *Optimisation des systèmes industriels de production : méthodes stochastiques et approche multi-agents*. PhD thesis, Uhniversité Blaise Pascal, Clermont Ferrand, France.

[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*, pages 285–292.

[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. *International Journal of Production Economics*.

[Larabi, 2010] Larabi, M. (2010). *Le problème de job-shop avec transport : modélisation et optimisation*. PhD thesis, Université Blaise Pascal - Clermont Ferrand II.

[Lei and Wang, 1991] Lei, L. and Wang, T. (1991). The minimum common-cycle algorithm for cyclic scheduling of two material handling hoists with time window constraints. *Management Science*, pages 1629–1639.

[Lei and Wang, 1994] Lei, L. and Wang, T. (1994). On the optimal cyclic schedules of single hoist electroplating processes. *IIE Transactions*, 26(2) :25–33.

[Leung and Levner, 2006] Leung, J. and Levner, E. (2006). An efficient algorithm for multi-hoist cyclic scheduling with fixed processing times. *Operations research letters*, 34(4) :465–472.

[Levner et al., 2007] Levner, E., Kats, V., and De Pablo, D. (2007). Cyclic scheduling in robotic cells : An extension of basic models in machine scheduling theory. *Multiprocessor Scheduling*, page 1.

[Levner et al., 1997] Levner, E., Kats, V., and Levit, V. (1997). An improved algorithm for cyclic flowshop scheduling in a robotic cell. *European Journal of Operational Research*, 97(3) :500–508.

[Levner et al., 1998] Levner, E., Meyzin, L., and Ptuskin, A. (1998). Periodic scheduling of a transporting robot under incomplete input data : A fuzzy approach. *Fuzzy Sets and Systems*, 98 :255–266.

[Li et al., 2006] Li, D., Wu, C., Tsai, T., and Chang, F. (2006). Using mega-fuzzification and data trend estimation in small data set learning for early fms scheduling knowledge. *Computers & operations research*, 33(6) :1857–1869.

[Liu and Jiang, 2005] Liu, J. and Jiang, Y. (2005). An efficient optimal solution to the two-hoist no-wait cyclic scheduling problem. *Operations Research*, pages 313–327.

[Liu et al., 2002] Liu, J., Jiang, Y., and Zhou, Z. (2002). Cyclic scheduling of a single hoist in extended electroplating lines : a comprehensive integer programming solution. *IIE Transactions*, 34(10) :905–914.

[Logendran and Sriskandarajah, 1996] Logendran, R. and Sriskandarajah, C. (1996). Sequencing of robot activities and parts in two-machine robotic cells. *International journal of production research*, 34(12) :3447–3463.

[Malmborg, 1990] Malmborg, C. (1990). A model for the design of zone control automated guided vehicle systems. *International Journal of Production Research*, 28(10) :1741–1758.

[Mangione et al., 2003a] Mangione, F., Brauner, N., Penz, B., et al. (2003a). Flow shop robotisé à quatre machine sans attente.

[Mangione et al., 2003b] Mangione, F., Brauner, N., Penz, B., et al. (2003b). Optimal cycles for the robotic balanced no-wait flow shop.

[Mangione et al., 2003c] Mangione, F., Brauner, N., Penz, B., et al. (2003c). Three-tank hoist scheduling problem with unbounded or zero-width processing windows.

[Manier and Bloch, 2003] Manier, M. and Bloch, C. (2003). A classification for hoist scheduling problems. *International Journal of Flexible Manufacturing Systems*, 15(1) :37–55.

[Manier and Lamrous, 2006] Manier, M. and Lamrous, S. (2006). Design and scheduling of electroplating facilities. In *Service Systems and Service Management, 2006 International Conference on*, volume 2, pages 1114–1119. IEEE.

[Manier and Lamrous, 2008] Manier, M. and Lamrous, S. (2008). An evolutionary approach for the design and scheduling of electroplating facilities. *Journal of Mathematical Modelling and Algorithms*, 7(2) :197–215.

[Manier et al., 1994] Manier, M., Varnier, C., and Baptiste, P. (1994). A multi-hoists scheduling problem approach. In *Proc. of the 4th Int. Workshop on Project Management and Scheduling*, pages 110–115.

[Manier et al., 2000] Manier, M., Varnier, C., and Baptiste, P. (2000). Constraint-based model for the cyclic multi-hoists scheduling problem. *Production Planning & Control*, 11(3) :244–257.

[Manier-Lacoste, 1994] Manier-Lacoste, M.-A. (1994). *Contribution à l'ordonnancement cyclique du système de manutention d'une ligne de galvanoplastie*. PhD thesis, Thèse de doctorat, Besançon.

[Mateo and Companys, 2007] Mateo, M. and Companys, R. (2007). New computational experiences on the hoist scheduling problem for cyclic manufacturing of different products. Technical report, Universitat Politècnica de Catalunya. Departament d'Organització d'Empreses Sense grup de recerca.

[Mateo et al., 2000] Mateo, M., Companys, R., and Bautista, J. (2000). Bounded cycle time for the cyclic hoist scheduling problem. In *I World Conference on Production and Operations Management (Sevilla)*.

[Mateo et al., 2002] Mateo, M., Companys, R., and Bautista, J. (2002). Resolution of graphs with Bounded Cycle Time for the Cyclic Hoist Scheduling Problem. In *8th International Workshop on Project Management and Scheduling*, pages 257–260. Citeseer.

[Mateo Doll, 2001] Mateo Doll, M. (2001). *Procedimientos de secuenciación y programación en un sistema productivo de estaciones en serie con transportadores asíncronos de material*. PhD thesis, Universitat Politècnica de Catalunya.

[Mei et al., 2009] Mei, D., Du, X., and Chen, Z. (2009). Optimization of dynamic parameters for a traction-type passenger elevator using a dynamic byte coding genetic algorithm. *Proceedings of the Institution of Mechanical Engineers, Part C : Journal of Mechanical Engineering Science*, 223(3) :595–605.

[Michalewicz et al., 1992] Michalewicz, Z., Janikow, C., and Krawczyk, J. (1992). A modified genetic algorithm for optimal control problems. *Computers & Mathematics with Applications*, 23(12) :83–94.

[Müller, 1983] Müller, T. (1983). Automated guided vehicles. *IFS (Publications) Ltd./Springer-Verlag, UK/Berlin*.

[Narayanan and Moore, 1996] Narayanan, A. and Moore, M. (1996). Quantum-inspired genetic algorithms. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 61–66. IEEE.

[Nonaka et al., 2012] Nonaka, Y., Erdős, G., Kis, T., Nakano, T., and Váncza, J. (2012). Scheduling with alternative routings in cnc workshops. *CIRP Annals-Manufacturing Technology*.

[Nowicki and Smutnicki, 1996] Nowicki, E. and Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management science*, pages 797–813.

[Paul et al., 2007] Paul, H., Bierwirth, C., and Kopfer, H. (2007). A heuristic scheduling procedure for multi-item hoist production lines. *International Journal of Production Economics*, 105(1) :54–69.

[Phillips and Unger, 1976] Phillips, L. and Unger, P. (1976). Mathematical programming solution of a hoist scheduling program. *Aiie Transactions*, 8(2) :219–225.

[Pisinger and Ropke, 2007] Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8) :2403–2435.

[Pundit and Palekar, 1990] Pundit, R. and Palekar, U. (1990). Job shop scheduling with explicit material handling considerations. Technical report, Working paper, Dept. of M and IE, Univ. of Illinois at Urbana-Champaign, Urbana, IL61801.

[Riera and Yorke-Smith, 2002] Riera, D. and Yorke-Smith, N. (2002). An improved hybrid model for the generic hoist scheduling problem. *Annals of Operations Research*, 115(1) :173–191.

[Rossé-Bloch, 1999] Rossé-Bloch, C. (1999). *Contribution à l'ordonnancement dynamique de lignes de traitement de surface.* Thèse de doctorat, Université de Franche-Comté, UFR Sciences et Techniques, Besançon.

[Sabuncuoglu and Hommertzheim, 1992] Sabuncuoglu, I. and Hommertzheim, D. (1992). Dynamic dispatching algorithm for scheduling machines and automated guided vehicles in a flexible manufacturing system. *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, 30(5) :1059–1079.

[Savelsbergh and Sol, 1998] Savelsbergh, M. and Sol, M. (1998). Drive : Dynamic routing of independent vehicles. *Operations Research*, pages 474–490.

[Sethi et al., 2001] Sethi, S., Sidney, J., and Sriskandarajah, C. (2001). Scheduling in dual gripper robotic cells for productivity gains. *Robotics and Automation, IEEE Transactions on*, 17(3) :324–341.

[Sethi et al., 1992] Sethi, S., Sriskandarajah, C., Sorger, G., Blazewicz, J., and Kubiak, W. (1992). Sequencing of parts and robot moves in a robotic cell. *J. Flexible Manufacturing Systems*, 4 :331–358.

[Shapiro and Nuttle, 1988] Shapiro, G. and Nuttle, H. (1988). Hoist scheduling for a pcb electroplating facility. *IIE transactions*, 20(2) :157–167.

[Song et al., 1993] Song, W., ZELDA, B., and RICHARD, L. (1993). An algorithm for scheduling a chemical processing tank line. *Production Planning & Control*, 4(4) :323–332.

[Sriskandarajah et al., 2004] Sriskandarajah, C., Drobouchevitch, I., Sethi, S., and Chandrasekaran, R. (2004). Scheduling multiple parts in a robotic cell served by a dual-gripper robot. *Operations Research*, pages 65–82.

[Subai et al., 2006] Subai, C., Baptiste, P., and Niel, E. (2006). Scheduling issues for environmentally responsible manufacturing : The case of hoist scheduling in an electroplating line. *International Journal of Production Economics*, 99(1-2) :74–87.

[SUBAÏ et al., 2003] SUBAÏ, C., NIEL, E., and BAPTISTE, P. (2003). Vers un pilotage propre des lignes de traitement de surface. In *4e Conférence Francophone de MOdélisation et SIMulation, MOSIM'03*. "Organisation et Conduite d'Activités dans l'Industrie et les Services".

[Subbaiah et al., 2009] Subbaiah, K., Rao, M., and Rao, K. (2009). Scheduling of agvs and machines in fms with makespan criteria using sheep flock heredity algorithm. *International Journal of Physical Sciences*, 4(2) :139–148.

[Tacquard and Martineau, 2001] Tacquard, C. and Martineau, P. (2001). Automatic notation of the physical structure of a flexible manufacturing system. *International Journal of Production Economics*, 74(1) :279–292.

[Ting, 2005] Ting, C. (2005). On the mean convergence time of multi-parent genetic algorithms without selection. *Advances in Artificial Life*, pages 403–412.

[Ulusoy et al., 1997] Ulusoy, G., Sivrikaya-erifolu, F., and Bilge, Ü. (1997). A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles. *Computers & Operations Research*, 24(4) :335–351.

[Van der Meer, 2000] Van der Meer, J. (2000). *Operational control of internal transport*. PhD thesis, Erasmus University Rotterdam.

[Varnier and Baptiste, 1995] Varnier, C. and Baptiste, P. (1995). A clp approach for finding a transition schedule between two cyclic mono-product productions in electroplating facilities. In *International Conference on Industrial Engineering and Production Management*, pages 372–381.

[Whitley et al., 1991] Whitley, D., Mathias, K., and Fitzhorn, P. (1991). *Delta coding : An iterative search strategy for genetic algorithms*. Citeseer.

[Xu and Huang, 2004] Xu, Q. and Huang, Y. (2004). Graph-assisted cyclic hoist scheduling for environmentally benign electroplating. *Industrial & engineering chemistry research*, 43(26) :8307–8316.

[Yang et al., 2004] Yang, J., Jaillet, P., and Mahmassani, H. (2004). Real-time multivehicle truckload pickup and delivery problems. *Transportation Science*, 38(2) :135–148.

Résumé

Nos travaux concernent l'étude d'une extension d'un problème d'ordonnancement bien connu sous l'appellation job shop. Nous appelons cette extension le *General Flexible Job Shop Scheduling Problem* (GFJSSP), qui se rencontre dans différents types d'ateliers ayant comme caractéristique commune d'être soumis à des contraintes dues à des ressources de transport. Le GFJSSP se caractérise par l'intégration de machines et robots flexibles. Le terme *General* induit par ailleurs la présence de robots dont la capacité est supposée unitaire dans notre étude, des temps opératoires bornés, et la possibilité de prise en compte d'emplacements de stockage spécifiques. Après avoir défini l'atelier et le problème correspondant à cette extension, nous avons proposé deux modélisations du GFJSSP ainsi défini : une première modélisation mathématique, et une modélisation graphique, qui correspond à une généralisation du graphe disjonctif couramment utilisé pour les problèmes de job shop. Nous avons ensuite abordé la résolution suivant deux étapes : tout d'abord en nous focalisant sur l'aspect séquencement des tâches de traitement et de transport, pour lequel nous avons élaboré deux méthodes heuristiques (de type Tabou et basée sur une procédure de shifting bottleneck améliorée) ; puis en intégrant dans un deuxième temps la problématique de l'affectation induite par la flexibilité de certaines ressources. Pour cette dernière étape, nous avons combiné les méthodes précédentes avec un algorithme génétique. L'algorithme hybride obtenu nous permet de résoudre des instances de la littérature correspondant à divers cas spécifiques, avec des résultats assez proches des meilleures méthodes dédiées. A termes, il pourrait être intégré dans un système d'aide à la décision général qui s'affranchirait de la phase d'identification préalable du type de job shop considéré, et serait adapté à la résolution de nombreux cas (avec ou sans problème d'affectation, temps de traitement fixes ou bornés, avec ou sans stockage, etc.).

Mots-clés: Ordonnancement, Job-shop flexible avec transport, Temps opératoires bornés, Stockage, Modélisation, Graphe disjonctif, Tabou, Shifting bottelneck modifié, Heuristiques, Algorithme génétique

Abstract

Our work focuses on an extension of the well known job shop scheduling problem. We call this extension the General Flexible Job Shop Scheduling Problem (GFJSSP). It occurs in various kinds of workshops which are particularly constrained by one or several transportation resources (called robots). GFJSSP is characterized by the flexibility of both machines and robots. In the studied problem, the term General involves unitary capacity transportation resources, bounded processing times, and possible input/output buffers for machines. After defining the workshop and the corresponding problem, we proposed two kinds of model for the GFJSSP : a mathematical model, and a graphical one. This last one is a generalization of the disjunctive graph commonly used for job shop problems. We then addressed the resolution in two steps : firstly, by focusing

on the sequencing of processing and transportation tasks. For this purpose we have developed two heuristics (Tabu search and an improved shifting bottleneck procedure). Secondly, we have considered the assignment problem involved by the flexibility of some resources. For this last step, we combined the above methods with a genetic algorithm. This hybrid algorithm allowed us to solve various specific cases of instances in the literature, with performance rather close to the best dedicated methods. In terms/the future, it could be integrated into/within a general decision support system which could emancipate from the identification phase prior to the considered type of job shop, and would be suitable for solving many cases (with or without assignment problem, fixed or bounded processing times, with or without storage, and so on).

Keywords: Scheduling, Flexible Job-shop with transport, bounded processing times, Storage, Modeling, Disjunctive Graph, Tabu Search, modified Shifting bottelneck, Heuristics, Genetic Algorithm