UNIVERSITÉ
DE REIMS
CHAMPAGNE-ARDENNE

CENTRE DE RECHERCHE EN STIC

THÈSE

Présentée pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Spécialité

INFORMATIQUE

Par

Safa KHALOULI

MÉTA-HEURISTIQUES À BASE DE MODÈLES : APPLICATIONS À L'ORDONNANCEMENT D'ATELIER FLOW-SHOP HYBRIDE MONOCRITÈRE ET MULTICRITÈRE

Soutenue publiquement le 12 juillet 2010, devant le jury composé de :

Président :

M. Jean-Charles POMEROL	Professeur à l'Université Paris VI

Rapporteurs :

Mme. Marie-Claude PORTMANN	Professeur à l'Ecole des Mines de Nancy
M. Christophe GONZALES	Professeur à l'Université Paris 6

Examinateurs :

M. Herman AKDAG	Professeur à l'Université de Reims Champagne-Ardenne
M. Aziz MOUKRIM	Professeur à l'Université de Technologie de Compiègne

Encadrante de thèse :

Mme. Fatima GHEDJATI	Maître de Conférences à l'Université de Reims Champagne-Ardenne

Directeur de thèse :

M. Abdelaziz HAMZAOUI	Professeur à l'Université de Reims Champagne-Ardenne

À MES CHERS PARENTS

REMERCIEMENTS

Je tiens tout d'abord à remercier Monsieur Jean-Charles POMEROL, Professeur des Universités de Paris VI pour l'honneur qu'il me fait en présidant le jury de ma thèse.

Je tiens à exprimer toute ma gratitude à Madame Fatima GHEDJATI, Maître de Conférences à l'université de Reims Champagne-Ardenne, qui a encadré cette thèse. Elle a su orienter aux bons moments mes travaux de recherches en me faisant découvrir l'ordonnancement. Je la remercie infiniment pour sa disponibilité, pour ses conseils, pour sa confiance, pour ses encouragements, pour ses commentaires précieux qui m'ont permis de surmonter les difficultés et de progresser, et pour ses corrections avisées de mes travaux et de ce mémoire.

J'adresse aussi mes sincères remerciements à Monsieur Abdelaziz HAMZAOUI, Professeur à l'Université de Reims Champagne-Ardenne, et également mon directeur de thèse de m'avoir accueillie dans son équipe. Je lui suis reconnaissante pour ses encouragements, ses conseils et sa confiance.

J'exprime ma reconnaissance et mes remerciements à Madame Marie-Claude PORTMANN, Professeur à l'Ecole des Mines de Nancy, pour ses précieux conseils et pour avoir accepté d'être rapporteur.

Je remercie Monsieur Christophe GONZALES Professeur à l'Université Paris 6, pour l'attention et l'intérêt qu'il a bien voulu porter à mon travail et pour avoir accepté la lourde tâche de rapporteur.

Mes remerciements vont également à Monsieur Herman AKDAG, Professeur à l'Université de Reims Champagne-Ardenne, et à Monsieur Aziz MOUKRIM, Professeur à l'Université de Technologie de Compiègne, d'avoir accepté d'être examinateur de cette thèse.

Il me tient tout particulièrement à cœur de remercier mes parents pour m'avoir soutenu dans cette aventure et pour toute la confiance qu'ils m'ont apportée. Je n'oublierai pas mon frère, ma sœur ainsi que toute ma famille pour leurs encouragements. Mille Merci.

Enfin, je tiens à exprimer ma reconnaissance et mon amour à tous mes amis, mes collègues et à tous ceux qui de près ou de loin m'ont encouragé, m'ont supporté et soutenu dans des moments difficiles. Ils sont nombreux et tous présents dans mon cœur.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GÉNÉRALE

La théorie de l'ordonnancement est une branche de la recherche opérationnelle. Elle occupe un rôle important dans nombreux secteurs de l'économie, notamment en gestion de production des entreprises de biens ou de services. L'enjeu économique considérable de ces problèmes ainsi que leurs champs d'applications qui n'ont pas cessé d'augmenter et d'évoluer, ont suscité depuis des décennies une recherche intensive. La fonction ordonnancement vise à définir les dates d'exécution d'un ensemble de travaux en tenant compte de la disponibilité des ressources, de manière à optimiser un ou plusieurs critère(s) donné(s), tout en respectant les contraintes de fabrication et d'organisation.

En général, les méthodes de résolution utilisées pour ce type de problèmes prennent en considération deux facteurs : la qualité des solutions et le temps de résolution. Ainsi, elles peuvent êtres classées en deux catégories :

- les méthodes exactes qui garantissent l'optimalité mais avec un temps de calcul qui risque d'augmenter exponentiellement avec la taille du problème ;
- les méthodes approchées qui peuvent perdre en optimalité pour gagner en temps d'exécution.

La plupart des problèmes d'ordonnancement sont classés NP-difficiles en théorie de la complexité (Rinnooy Kan, 1976). Par conséquent, les méthodes approchées constituent une alternative intéressante pour leur résolution. Dans le cadre de cette thèse, nous nous intéressons particulièrement aux méthodes dites méta-heuristiques qui sont des heuristiques générales applicables à de nombreux problèmes. Ces méthodes ne fournissent aucune garantie sur la qualité des résultats, mais ont permis de résoudre avec succès plusieurs problèmes d'optimisation et en particulier une multitude de problèmes d'ordonnancement. Dans la littérature, nous pouvons distinguer parmi les problèmes d'ordonnancement d'atelier les plus rencontrés ceux à machine unique, à machines parallèles, le flow-shop, le job-shop, l'open-shop, etc.

Dans le cadre de l'étude théorique de cette thèse, nous nous focalisons sur l'ordonnancement d'atelier de type flow-shop hybride avec machines en parallèle identiques dans les cas monocritère et multicritère. En effet, ce type de structure d'atelier est intéressant pour la modélisation de plusieurs problèmes issus du monde industriel. Le premier critère que nous cherchons à minimiser est la date d'achèvement de tous les travaux. La compétitivité du service de production et la forte concurrence mondiale ont introduit un certain nombre de critères économiques liés à la production « juste-à-temps », dont l'étude ne cesse d'évoluer ces dernières années. Parmi ces critères d'optimisation juste-à-temps, nous nous intéressons à minimiser l'avance et le retard des travaux, ce qui représente la deuxième catégorie de critères abordée dans notre thèse. Le problème d'ordonnancement multicritère, que nous considérons, a pour but de minimiser à la fois la date d'achèvement et l'avance/retard de tous les travaux.

L'objet de cette thèse est donc de proposer et de tester des méthodes méta-heuristiques à base d'algorithmes de fourmis pour la résolution des problèmes considérés. Pour le cas multicritère, nous proposons d'hybrider ces dernières approches avec un module d'évaluation à base de la logique floue pour le choix d'une solution parmi toutes les solutions trouvées.

La thèse est organisée de la manière suivante.

Dans le premier chapitre, nous présentons une introduction aux problèmes d'ordonnancement. Nous rappelons brièvement ces notions, ainsi que les différentes méthodes de résolution.

Dans le second chapitre, nous décrivons le flow-shop hybride et nous dressons un état de l'art des travaux réalisés traitant ce problème. Ensuite, nous présentons, d'une manière générale, l'approche de résolution à base de modèles. Ainsi, nous rappelons le fonctionnement global des algorithmes de fourmis en présentant ces différentes variétés. Puis, nous donnons quelques domaines d'applications pour ce type de méta-heuristiques en insistant sur leurs applications aux problèmes d'ordonnancement.

Dans le troisième chapitre, nous présentons, d'une manière détaillée, les trois approches de résolution à base de fourmis, que nous avons conçues et développées pour minimiser la date d'achèvement de tous les travaux dans un atelier flow-shop hybride. Nous finissons ce chapitre par une série de tests pour évaluer et comparer les méthodes développées.

Dans le quatrième chapitre, nous nous intéressons à l'étude du problème d'ordonnancement flow-shop hybride avec critères d'optimisation « juste-à-temps ». Nous commençons par exposer les travaux existants dans la littérature concernant la production « juste-à-temps ». Ensuite, nous présentons, les approches de résolution à base de colonies de fourmis, que nous

avons mises en place pour ce type de problèmes d'ordonnancement. De même, une adaptation d'heuristiques constructives conçues pour le flow-shop classique est exposée. Nous terminons par une synthèse des résultats que nous avons conduits pour évaluer et comparer les méthodes proposées.

Le cinquième chapitre, est consacré à la résolution d'un problème d'ordonnancement flow-shop hybride multicritère. Nous exposons dans cette partie une approche qui hybride un algorithme de fourmis pour trouver des solutions possibles au problème et un module d'aide à la décision pour l'évaluation des solutions générées. Dans ce travail, nous proposons un module d'évaluation floue qui utilise un opérateur d'agrégation via une intégrale de Choquet. Nous terminons ce chapitre par une série d'expérimentations.

Une conclusion générale fait une synthèse des problèmes abordés, des résultats obtenus et de quelques perspectives de recherche qui peuvent découler de ces études.

Chapitre 1

INTRODUCTION AUX PROBLÈMES D'ORDONNANCEMENT

Dans un premier temps, ce chapitre commence par une introduction aux problèmes d'ordonnancement. Ensuite, un panorama des différentes méthodes utilisées pour leur résolution est dressé. Enfin, nous présenterons la problématique de nos travaux ainsi que nos objectifs.

1 INTRODUCTION

Pour faire face à la concurrence, toute entreprise se doit d'être plus performante et d'avoir un niveau supérieur sur le plan technique et économique. Chaque entreprise doit donc se doter d'un système de production efficace qui réagisse aux exigences et évolutions du marché. En conséquence, les systèmes de production de biens ou de services n'ont cessé de se diversifier et de devenir plus complexe. Ils font partie de plusieurs domaines de l'économie, sous d'innombrables formes : l'industrie (les usines, les ateliers,...), les systèmes informatiques (processeurs,…), le secteur des services (administration, guichet, agence, hôpital,...).

Les besoins industriels ont entraîné l'apparition et le développement de nouvelles technologies pour mieux gérer ces systèmes productifs : d'où l'importance du rôle de la gestion de production. Cette dernière a pour objet *« la recherche d'une organisation efficace dans l'espace et dans le temps de toutes les activités relatives à la production afin d'atteindre les objectifs de l'entreprise »* (Giard, 1988). Parmi les différentes fonctions de la gestion de production, nous nous intéressons plus particulièrement à la fonction ordonnancement. Cette fonction présente une importance économique pour les entreprises et les domaines où elle s'applique sont très variés. Ces problèmes présentent un grand intérêt à la fois pour la communauté scientifique et pour les praticiens en milieu industriel, qui font face à des problématiques rendues complexes par les contraintes des applications réelles. C'est pourquoi, leurs applications ont fait l'objet d'un très grand nombre de travaux de recherche et d'ouvrages. Nous pouvons notamment citer ceux de (Carlier & Chrétienne, 1988; GOThA, 1993; Chu & Proth, 1996; Blazewicz et al., 1996; Esquirol & Lopez, 1999; Lopez & Roubellat, 2001).

Nous proposons dans ce chapitre une brève introduction, dans laquelle nous rappelons simplement les notions de base utiles à la compréhension de ce mémoire, en particulier à la présentation des problèmes d'ordonnancement d'atelier.

2 PRÉSENTATION DES PROBLÈMES D'ORDONNANCEMENT

En dehors de tout contexte d'application, les problèmes d'ordonnancement sont définis comme : la programmation dans le temps de l'exécution d'une série de tâches (activités) sur un ensemble de ressources physiques (humaines et techniques), en cherchant à optimiser certains critères ou objectifs (financiers ou technologiques), tout en respectant les contraintes

de fabrication et d'organisation. Établir un ordonnancement revient donc à coordonner l'exécution de toutes les tâches, en utilisant au mieux les ressources disponibles (Esquirol & Lopez, 1999). En d'autre termes, il s'agit de :

« *déterminer ce qui va être fait, quand, où et avec quels moyens ; étant donné un ensemble de tâches à accomplir, le problème d'ordonnancement consiste à déterminer quelles tâches doivent être exécutées et à assigner des dates et des ressources à ces tâches de façon à ce que les tâches soient, dans la mesure du possible, accomplies en temps utile, au moindre coût et dans les meilleures conditions* » (Parunak, 1985).

Les différentes données sont donc : *les tâches*, *les ressources*, *les contraintes* et *les objectifs*. Une solution à un tel problème consiste à trouver une planification des tâches sur les ressources en optimisant les objectifs et en respectant les contraintes. Dans le cadre de notre étude, nous nous intéressons principalement aux problèmes d'ordonnancement d'atelier. Dans ce cas, une tâche est généralement désignée par le terme opération. Il s'agit d'une entité élémentaire d'un travail (job en anglais). Quant aux ressources, elles sont représentées par des machines. La modélisation d'un tel problème dépend fortement des différents paramètres tels que les contraintes et les objectifs.

2.1 *MODÉLISATION DES PROBLÈMES D'ORDONNANCEMENT D'ATELIER*

2.1.1 Les travaux et les opérations

La réalisation d'un ordonnancement d'atelier est décomposable en travaux (dits aussi pièces ou produits). Une opération est une entité élémentaire d'un travail qui a une date de début et une date de fin. Son exécution est caractérisée par une durée appelée temps opératoire où temps d'exécution et nécessite l'utilisation d'une ou plusieurs ressources.

Les opérations peuvent être exécutées par morceaux, on parle alors d'*opérations morcelables* ou encore *de problèmes préemptifs*. Dans ce cas, l'exécution d'une opération peut être interrompue et reprise ultérieurement tout en poursuivant l'exécution de l'opération à partir de l'état où elle a été interrompue (Carlier & Chrétienne, 1988). Il existe aussi certains cas où la préemption d'une opération est permise mais son exécution nécessite de recommencer l'opération totalement ou partiellement. Dans d'autres cas, les opérations doivent être exécutées sans interruption. Il s'agit d'opérations *non morcelables* (Esquirol & Lopez, 1999) qui ne peuvent pas être interrompues avant que leur exécution sur une ressource donnée ne soit terminée.

2.1.2 Les ressources

Les travaux requièrent, pour leur traitement, des moyens techniques (comme les machines) et/ou des mains d'œuvres que l'on qualifie de « ressources ». Parmi les ressources, nous pouvons distinguer trois catégories (Esquirol & Lopez, 1999).

- ***Les ressources renouvelables*** qui sont à nouveau réutilisable après avoir réalisé un travail (tels que les équipements, les ouvriers,…). Parmi ces ressources, nous retrouvons les ressources disjonctives (non partageables) qui ne peuvent exécuter qu'un seul travail à un instant donné et les ressources cumulatives (partageables) qui peuvent exécuter un nombre limité de travaux simultanément.
- ***Les ressources consommables*** qui s'épuisent après leur utilisation et ne sont plus disponibles pour les travaux restants à exécuter (comme la matière première, budget, …).
- ***Les ressources doublement contraintes ou limitées*** qui combinent les contraintes liées aux deux catégories précédentes. Elles présentent une limitation de leur utilisation instantanée et de leur consommation globale : c'est le cas des ressources d'énergie (pétrole, électricité, etc.).

2.1.3 Les contraintes

Les contraintes expriment des restrictions sur les valeurs que peuvent prendre simultanément les variables de décision, liées aux travaux et aux ressources. Il s'agit des conditions à respecter lors de la construction d'un ordonnancement pour qu'il soit réalisable. Il existe deux catégories de contraintes : *les contraintes temporelles* et *les contraintes de ressources* (Esquirol & Lopez, 1999). Le premier type concerne les contraintes d'antériorité ou de cohérence technologique, qui décrivent des relations d'ordre relatif entre les différents travaux, les délais de fabrication imposés et les contraintes d'interdiction, d'autorisation ou d'interruption des travaux, etc. Quant au second type, il est lié aux caractéristiques d'utilisation et de disponibilité des ressources utilisées par les travaux. Deux types de contraintes de ressources se distinguent par rapport à la nature disjonctive[1] ou cumulative[2] des ressources.

[1] qui interdit d'exécuter simultanément deux travaux en utilisant la même ressource
[2] qui permet d'exécuter plusieurs travaux en parallèle

2.1.4 Les critères d'optimisation

Pour évaluer la qualité d'un ordonnancement, des mesures connues sous l'appellation de critères, dites aussi objectifs, sont utilisées. Ces critères sont généralement liés aux temps, aux ressources ou aux coûts. Il s'agit, le plus souvent, de considérer le maximum ou la somme (qui peut être une somme totale ou pondérée) sur tous les travaux d'une certaine mesure ou d'une combinaison de plusieurs mesures. Parmi les critères les plus utilisés, nous retrouvons :

- ***la durée totale de l'ordonnancement*** (« makespan ») définie par $C_{\max} = \max C_j$, C_j étant la date de fin d'exécution du travail *j* qui représente la date d'achèvement du travail le plus tardif. En minimisant ce critère, nous avons une utilisation plus efficace des ressources (Ruiz & Allahverdi, 2009) ;
- ***la somme des dates de fin des travaux*** (« total completion time ») $\sum_{1\le j\le n} C_j$;
- ***le plus grand retard algébrique*** défini par $L_{\max} = \max L_j$, tel que $L_j = C_j - d_j$ est le retard algébrique (« lateness » en anglais) et d_j désigne la date de fin souhaitée ou encore la date d'échéance (« due date » en anglais) ;
- ***le plus grand retard vrai*** $T_{\max} = \max_{1\le j\le n} T_j$, où $T_j = \max\left(0, C_j - d_j\right)$ est le retard vrai (« tardiness » en anglais) du travail *j* ;
- ***la somme des retards*** $\sum_{1\le j\le n} T_j$;
- ***le nombre de travaux en retard*** $\sum_{1\le j\le n} U_j$, U_j étant l'indicateur de retard qui est égal à 1 si $T_j > 0$ et 0 sinon ;
- ***la somme des avances*** $\sum_{1\le j\le n} E_j$, tel que $E_j = \max\left(0, d_j - C_j\right)$ représente l'avance (« earliness » en anglais) du travail *j* ;
- ***la somme des encours*** $\sum_{1\le j\le n} F_j$, définie par le temps de séjour des travaux (« flow time » en anglais) dans l'atelier. $F_j = C_j - r_j$, r_j étant la date de disponibilité du travail *j*.

Ces critères peuvent être classés en deux types :

- ***critère régulier*** : il s'agit d'un critère dont le coût d'une solution est une fonction croissante des dates d'achèvement des travaux (Gupta et al., 2002). Ainsi la valeur de la fonction objectif ne peut pas diminuer par l'insertion de temps mort entre deux travaux consécutifs (Hoogeveen, 2005) ;

– ***critère irrégulier*** : dans ce cas le critère est non régulier car il n'est pas monotone croissant comme la plupart des critères en ordonnancement. Ainsi, il est possible d'améliorer une solution en introduisant un temps mort entre deux travaux consécutifs.

Le traitement de l'ordonnancement dans la littérature s'est tout d'abord orienté vers une optimisation monocritère, puis face à l'évolution et à la concurrence des systèmes de production, ces critères se sont diversifiés et l'orientation vers une optimisation multicritère est devenue de plus en plus importante (nous revenons par la suite sur cette notion, dans le Chapitre 5).

2.2 *MÉTHODES DE RÉSOLUTION*

Les problèmes d'ordonnancement d'atelier sont des problèmes d'optimisation combinatoire qui nécessitent d'effectuer un nombre important de calculs pour obtenir un ordonnancement admissible (ou réalisable) qui optimise le (ou les) critère(s) retenu(s) en tenant compte des contraintes. Le développement de la théorie de complexité a permis de classifier ces problèmes selon leurs difficultés (Rinnooy Kan, 1976; Graham et al., 1979). Dans la plupart des cas, ils ont été démontrés NP-difficiles (Lawler et al., 1993). Par conséquent, il n'existe pas de méthodes universelles permettant de résoudre efficacement tous les problèmes d'ordonnancement sans que le temps de résolution n'augmente de façon exponentielle avec la taille de celui-ci (Nous renvoyons le lecteur aux ouvrages de (Rinnooy Kan, 1976; Garey & Johnson, 1979; Blazewicz et al., 1996) pour plus de détails sur la théorie de la complexité).

Dans la littérature, nous retrouvons plusieurs approches de résolution des problèmes d'ordonnancement qui font appel aux techniques de l'optimisation combinatoire. Si une méthode peut fournir une solution optimale, il s'agit alors de méthode optimale ou exacte. Dans ce cas, pour trouver un optimum, une recherche exhaustive explicite ou implicite est donc nécessaire. De manière générale, ce type d'approches permet de résoudre les problèmes d'ordonnancement de petite taille, mais il reste difficile de fournir une résolution exacte en un temps raisonnable lorsque les problèmes sont de taille importante (ou industrielle). D'où la nécessité d'adopter des méthodes de résolution approchée proposant des solutions acceptables (c'est-à-dire non nécessairement optimales) en un temps de calcul réduit.

Nous allons passer en revue les méthodes les plus connues en les répertoriant en méthodes exactes et méthodes approchées.

2.2.1 Méthodes de résolution exacte

Rappelons que les approches exactes ont pour but de rechercher un ordonnancement optimal. Elles se caractérisent par l'augmentation du temps de calcul de façon exponentielle avec la taille du problème, ce qui explique leurs utilisations le plus souvent pour la résolution des problèmes de petite taille. Parmi elles, nous distinguons :

2.2.1.a La programmation linéaire

La programmation linéaire permet de modéliser les problèmes dont le critère et les contraintes sont des fonctions linéaires des variables de décision (Lopez & Roubellat, 2001). Parmi les algorithmes les plus importants pour le traitement d'un programme linéaire nous citons la méthode du simplex.

2.2.1.b La programmation dynamique

La programmation dynamique a été établie par (Bellman, 1954). Elle a été appliquée pour la résolution de certains problèmes d'optimisation à l'aide de formules de récurrence. Pour les problèmes d'ordonnancement, cette technique est applicable uniquement si le problème est décomposable en étape et si le critère d'optimisation présente certaines propriétés particulières par exemple une forme additive ou multiplicative (Lopez & Roubellat, 2001).

2.2.1.c Les procédures par séparation et évaluation

Les procédures d'optimisation par séparation et évaluation (PSE), (en anglais « Branch and Bound »), sont les méthodes génériques les plus utilisées pour la résolution de problèmes d'optimisation combinatoire. Elles explorent, par énumération implicite, l'ensemble de toutes les solutions. Le principe de la PSE repose sur une technique de séparation et d'évaluation. La séparation permet de décomposer le problème en un ensemble de sous-problèmes de taille réduite. Les séparations successives, effectuées durant la procédure, permet de construire un arbre de recherche formé d'un ensemble de nœuds, chacun d'eux représentant un sous problème. L'évaluation associe une borne du critère d'optimisation (une borne inférieure dans le cas d'une minimisation ou une borne supérieure dans le cas d'une maximisation) à chaque nœud de l'arborescence de recherche. Elle a pour but de déterminer un minorant de l'optimum de l'ensemble des solutions réalisables associé au nœud en question ou, au contraire, de prouver mathématiquement que cet ensemble ne contient pas de solutions intéressantes pour la résolution du problème. Lorsqu'un tel nœud est identifié dans l'arbre de recherche, il est

donc inutile d'effectuer la séparation de son espace de solutions. Étant donnée une arborescence en cours de construction, il faut donc choisir le prochain sommet à séparer.

2.2.2 Méthodes de résolution approchée

Les méthodes approchées, dites aussi d'approximation, constituent une alternative intéressante pour traiter les problèmes d'ordonnancement de grande taille. Elles sont définies comme étant des procédures exploitant au mieux la structure du problème considéré afin de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible (Nicholson, 1971). Ainsi, les performances de ce type d'approches dépendent de la qualité de la solution obtenue et du temps de calcul nécessaire pour sa réalisation. Plusieurs méthodes approchées ont été développées pour la résolution des problèmes d'ordonnancement (GOThA, 1993; Lopez & Roubellat, 2001).

Lorsque ces méthodes sont conçues de manière simple, rapide et ciblée sur un problème particulier, on les appelle ***heuristique***. Leur principe de base repose généralement sur l'utilisation de règles empiriques simples qui exploitent les propriétés structurelles d'une solution et tente de retrouver une solution admissible par des critères de décision déduits de la connaissance du problème. Cependant, lorsque celles-ci sont générales, adaptables et applicables à plusieurs catégories de problèmes d'optimisation combinatoire, elles portent le nom de ***méta-heuristique***. Les méta-heuristiques sont généralement des algorithmes stochastiques, qui progressent vers un optimum par échantillonnage d'une fonction objectif. Ainsi, elles sont le plus souvent, guidées par l'exploration aléatoire de l'espace de recherche. Dans ce qui suit, nous présentons quelques unes de ces méthodes approchées.

2.2.2.a Les méthodes par construction

Il s'agit de construire de manière itérative une solution en complétant étape par étape un ordonnancement partiel (Nawaz et al., 1983). De nombreux algorithmes constructifs existent, à savoir :

- les algorithmes gloutons. Les décisions faites durant le processus de recherche ne sont jamais remises en cause. Les algorithmes de liste sont des exemples d'algorithmes gloutons. Ils utilisent des règles de priorité (exemples : priorité au travail ayant le plus court temps opératoire, priorité au travail ayant la plus proche date au plus tard,…) pour ordonnancer la liste des travaux. Une autre manière de construire des algorithmes gloutons consiste à choisir des décisions qui correspondent localement à des optima

partiels locaux.

2.2.2.b Les méthodes par décomposition

Elles se basent sur la division du problème initial en plusieurs sous-problèmes de taille réduite, facilitant ainsi la résolution de ces sous-problèmes et permettant de construire une solution globale à partir des solutions partielles obtenues par chaque sous-problème. Il existe plusieurs techniques de décomposition selon que le type est : ***hiérarchique***, ***structurel***, ***spatial***, ***temporel***, ou ***de l'ensemble des solutions*** (Portmann, 1988).

2.2.2.c Les méthodes par voisinage ou de recherche locale

Les approches par voisinage, appelées également les méthodes de recherche locale, sont des algorithmes itératifs initialisés par une solution réalisable (obtenue soit de manière aléatoire, soit à l'aide d'une autre heuristique, par exemple une heuristique constructive). Ils cherchent à améliorer, à chaque itération, la solution courante par des modifications locales. La recherche consiste donc à se déplacer successivement dans l'ensemble des solutions en passant, à chaque étape, d'une solution à une autre voisine, jusqu'à la vérification d'une condition d'arrêt. Les modalités des déplacements conduisent à de nombreuses méthodes telles que :

- ***La méthode de descente*** (« Hill Climbing » en anglais) qui consiste à partir d'une solution courante, de choisir à chaque itération, une nouvelle solution dans le voisinage. Cette solution est acceptée si elle est meilleure que la solution courante et devient ainsi la nouvelle solution courante. Ce processus recommence jusqu'à ce qu'il n'y ait plus aucune solution améliorante dans le voisinage. Dans le cas contraire, si aucune solution parmi les solutions voisines n'est strictement meilleure que la solution courante, alors celle-ci est un optimum local et on arrête l'algorithme.
- ***La méthode de recherche tabou*** (TS pour « Tabu Search ») est une procédure de recherche locale qui utilise un mécanisme de mémoire. Elle a été présentée par Glover (1987). L'idée principale est de pouvoir mémoriser l'historique de l'exploration de solutions afin de conserver les informations sur les solutions déjà visitées. Comme la méthode de la descente, nous partons d'une solution initiale, puis à chaque itération, la recherche tabou va examiner les voisinages. La meilleure solution devient la nouvelle solution courante. Le processus recommence jusqu'à ce qu'une condition d'arrêt soit satisfaite. Pour empêcher le blocage de la recherche sur un optimum local (en interdisant de revenir sur des solutions précédemment visitées de l'espace d'état), la recherche tabou

est dotée d'une mémoire de mouvements et d'un mécanisme d'échappement (appelé aussi mécanisme d'aspiration). La mémoire de mouvements constitue la liste tabou de mouvements interdits qui va servir à empêcher l'accès aux dernières solutions visitées.

- ***La méthode du recuit simulé*** (SA pour « Simulated Annealing ») tire son origine de la physique statistique. Elle a été introduite par analogie avec des phénomènes de la thermodynamique (Kirkpatrick et al., 1983). Son principe de fonctionnement repose sur une imitation du phénomène de recuit en science des matériaux en s'appuyant sur les travaux de Metropolis et al. (1953). Le recuit est une opération consistant à laisser refroidir lentement un métal d'une manière contrôlée pour améliorer ses qualités et obtenir une structure régulière, comme dans les cristaux et l'acier. L'idée physique est qu'un refroidissement trop brutal peut bloquer le métal dans un état peu favorable présentant ainsi des défauts : c'est l'équivalent d'un optimum local pour un problème d'optimisation combinatoire. Un refroidissement lent permettra aux molécules de s'agencer au mieux dans une configuration stable, équivalent à un optimum global. Ainsi, la température est un paramètre indispensable pour modifier l'état du matériau. Le recuit est donc une stratégie de contrôle de la température pour avoir un système physique dans un état de basse énergie. De manière analogue, la méthode de recuit simulé est une procédure itérative qui consiste à engendrer, à chaque itération, une nouvelle solution dans le voisinage de la solution courante. Si la nouvelle solution est meilleure, elle est acceptée, sinon elle est acceptée selon le critère de Metropolis, qui utilise une probabilité d'acceptation dépendante de la variable de température du matériau.
- ***La méthode de recherche à voisinage variable*** (VNS pour « Variable Neighbourhood Search ») est basée sur la performance des approches de descente. Elle a été proposée par Mladenović et Hansen (1997). L'idée de base est de changer la structure de voisinages durant la recherche. Elle utilise plusieurs types de voisinages permettant ainsi de diversifier l'exploration de l'espace de solutions.
- ***La méthode de recherche locale guidée*** (GLS pour « Guided Local Search ») est une procédure de recherche locale qui a été proposée par Voudouris et Tsang (1995). Elle repose sur une modification dynamique de la fonction à optimiser en ajoutant une pénalité ou un ensemble de terme de pénalité, dans le but de rendre les minima locaux déjà visités moins attractifs. La recherche locale se trouve donc orientée par les termes de pénalité et concentre la recherche sur des régions différentes de l'espace de solutions. Le processus est réitéré en procédant à des appels itératifs de la recherche locale. Si durant le processus

de recherche, cette dernière est piégée autour d'un optimum local, alors un changement des termes de pénalité est appliquée, puis la recherche est de nouveau relancée.

- ***La méthode GRASP*** (pour « Greedy Randomized Adaptive Search Procedure») a été développée par Feo et Resende (1995). Elle répète un processus composé de deux étapes : la construction d'une solution suivie par une phase d'amélioration de la solution construite. Pour l'étape de la construction de la solution initiale, le GRASP utilise des procédures gloutonnes aléatoires. L'utilisation d'une composante aléatoire permet la diversification des solutions dans l'espace de recherche. A partir de cette solution, une recherche locale est ensuite appliquée pour l'améliorer. Le processus est itéré jusqu'à la satisfaction d'un critère d'arrêt et retourne à la fin la meilleure solution trouvée.
- ***La méthode de recherche locale réitérée*** (ILS pour « Iterated Local Search ») est une variante de la méthode de descente qui combine une procédure de recherche locale et des perturbations (Lourenço et al., 2003). A partir d'une solution initiale, cette approche répète un processus composé de deux phases : une perturbation aléatoire basée sur l'utilisation d'un voisinage plus large permettant de modifier une grande partie de la solution courante et une recherche locale qui permet d'améliorer cette solution.

2.2.2.d Les méthodes évolutives

Contrairement aux méthodes par construction et par voisinage qui font intervenir une solution unique, les méthodes évolutives, dites aussi à base de population, considèrent un ensemble de solutions, appelé population. Ils font évoluer une population de solutions à chaque étape du processus de recherche permettant d'identifier et d'explorer les caractéristiques que les solutions ont en commun. Parmi les méthodes de cette catégorie, on peut citer les algorithmes évolutionnaires, les algorithmes de colonies de fourmis, les méthodes par essaims particulaires et les algorithmes à estimation de distribution. Ces méthodes se différencient par leur manière de représenter les problèmes à résoudre et par leur façon de faire évoluer la population d'une génération à l'autre.

- ***Les algorithmes évolutionnaires*** (« Evolutionary Computation » en anglais) sont des techniques d'optimisation itératives et stochastiques inspirées de la théorie de l'évolution. Cette théorie a été élaborée par Charles Darwin et fondée sur le modèle biologique. Un algorithme évolutionnaire simule le comportement d'évolution des êtres vivants. Dans ce cas, on considère que l'évolution tend à produire des organismes plus adéquat à leur environnement en ayant recourt à des phénomènes comme la sélection naturelle et

l'héritage génétique. Au cours du cycle de simulation, trois opérateurs interviennent généralement : la recombinaison (ou croisement), la mutation et la sélection. La recombinaison et la mutation créent de nouvelles solutions candidates, tandis que la sélection élimine les candidats les moins prometteurs. Plusieurs types d'évolution ont été développés, parmi lesquelles nous citons : les stratégies d'évolution (ES pour « Evolution Strategies »), la programmation évolutionnaire (EP pour « Evolutionary Programming ») et les algorithmes génétiques (GA pour « Genetic Algorithms ») (Schoenauer & Michalewicz, 1997). Ces derniers sont particulièrement connus et les plus utilisés en optimisation et en ordonnancement (voir par exemple (Ghedjati, 1994)).

- ***Les algorithmes de colonies de fourmis*** ont été introduits par Colorni et al. (1991a). L'origine de cette approche repose sur une métaphore où le comportement des fourmis réelles lors de la quête de recherche de nourriture est reproduit. En effet, bien qu'ayant individuellement des capacités cognitives très limitées, les fourmis sont capables de sélectionner collectivement le plus court chemin pour aller de leur nid à une source de nourriture. Pour cela, elles utilisent l'environnement comme support de communication grâce au dépôt et au suivi de pistes de phéromone (une substance chimique volatile). En imitant ce comportement, une famille d'algorithmes d'optimisation a été proposée (Dorigo & Stützle, 2004). L'idée de base consiste à travailler sur une population de solutions, chacune correspondant à une fourmi en utilisant une structure de donnée commune (partagée par toutes les fourmis) qui contient des informations sur les quantités de phéromones accumulées dans l'espace des solutions pour guider leurs recherches.
- ***La méthode par essaims particulaires*** (PSO pour « Particle Swarm Optimization ») est une technique d'optimisation stochastique développée par Eberhart et Kennedy (1995). elle s'inspire des déplacements collectifs observés chez certains animaux sociaux tels que les poissons et les oiseaux. En effet, ces groupes d'animaux montre des dynamiques de déplacements relativement complexes, alors qu'individuellement, ils n'ont accès qu'à des informations limitées, comme la position et la vitesse de leurs plus proches voisins. Ce type d'approche utilise une population de solutions potentielles connue sous l'appellation d'essaim, les individus sont appelés particules. Elle se base sur la collaboration des particules entre elles en échangeant des informations permettant ainsi l'émergence de comportements complexes. Chaque particule est caractérisée par sa position courante et un vecteur de changement de position (appelé vélocité). Pour influencer leurs évolutions, ces particules sont dotées d'une mémoire structurée au niveau local (c'est à dire entre

particules voisines). Chaque particule n'évolue, à chaque itération, qu'en fonction de ses proches voisins, et non pas selon l'état global de la population à l'itération précédente.

- ***Algorithmes à estimation de distribution*** (EDA pour « Estimation of distribution algorithm » ou encore « Probabilistic Model Building Genetic Algorithms »). Il s'agit d'approches basées sur des modèles probabilistes (voir (Larrañaga et al., 1999; Larrañaga & Lozano, 2002; Pelikan et al., 2002)) et inspirées des algorithmes génétiques. Ils travaillent sur une population de solutions qu'ils cherchent à améliorer en générant d'autres populations. La construction des nouvelles populations se fait grâce à l'estimation d'une distribution de probabilité (issue de la fonction objectif), associée à chaque individu de la population de l'itération précédente pour décrire la qualité des solutions. La distribution doit faire apparaître les zones prometteuses de manière à y concentrer la recherche. La méthode commence avec un échantillonnage aléatoire des solutions. Une partie de ces solutions va permettre de faire une première estimation de la distribution de la fonction de coût. A partir de cette estimation, un nouvel échantillon de solutions est créé de manière probabiliste de façon à concentrer l'échantillon dans les zones prometteuses identifiées par l'estimation de distribution. Ce processus est réitéré afin d'affiner l'estimation de distribution et de trouver de meilleures solutions.

2.2.3 Quelques propriétés associées aux méthodes approchées

La plupart des méthodes approchées ne sont pas déterministes, c'est-à-dire que deux exécutions successives utilisant les mêmes données peuvent retourner deux résultats différents. Ils reposent généralement sur un ensemble de concepts à savoir :

- ***la mémoire***, qui est un support d'apprentissage permettant de sauvegarder les informations recueillies par l'algorithme, à mesure que la recherche avance afin d'en faire usage ultérieurement ;
- ***l'intensification***, qui permet de rechercher des solutions a priori de plus grande qualité en exploitant les informations rassemblées par le système durant la recherche ;
- ***la diversification***, qui permet d'explorer de nouvelles régions de l'espace de recherche non encore visitées.

Les associations multiples de ces concepts peuvent conduire à une grande variété d'autres méthodes.

2.2.4 Autres méthodes de résolution

Il existe bien d'autres méthodes pour résoudre les problèmes d'ordonnancement. Parmi celles-ci, on retrouve les approches hybrides qui combinent au moins deux procédures de recherche différentes, y compris les méthodes exactes et/ou approchées (sus-décrites). L'hybridation est une technique qui essaie de tirer profit des points forts de chaque approche de résolution utilisée pour améliorer le comportement global de la méthode hybride. Elle semble constituer une technique intéressante qui permet d'élargir le spectre d'application d'une méthode de recherche et/ou d'augmenter ses performances (Duvivier, 2000). Cependant, le choix des approches à combiner est primordial pour obtenir une bonne coopération entre les constituants de l'approche hybride. Ainsi il est important de savoir dégager les points forts et également les points faibles de ces dernières. Nous citons à titre d'exemple le cas des algorithmes mémétiques, connus aussi sous le nom d'algorithmes génétiques hybrides, qui combinent une recherche locale (qui assure l'intensification de la recherche) et un algorithme génétique (qui renforce la diversification). Plusieurs types d'hybridations sont possibles. Pour une classification des différents niveaux possibles de combinaison pour les approches hybrides voir (Duvivier, 2000; Talbi, 2002).

Une autre méthodologie de recherche, portant le nom d'hyper-heuristique, a été récemment utilisée pour résoudre des problèmes d'optimisation. Ce type d'approche propose de regrouper un ensemble d'heuristiques et/ou de méta-heuristiques et de mettre en place un mécanisme capable d'identifier et de sélectionner l'heuristique la plus performante au cours du processus d'optimisation parmi l'ensemble des heuristiques considérées (Burke et al., 2003). La technique de sélection des heuristiques est une caractéristique très importante pour ce type d'approches.

2.2.5 Logique floue et ordonnancement

La « logique floue » recouvre un ensemble de modèles et de techniques mathématiques qui sont basés sur la notion des ensembles flous (Dubois & Prade, 1995). Elle est classée parmi les techniques de l'intelligence artificielle. La notion d'ensemble flou permet de modéliser la représentation humaine des connaissances, et améliorer les performances des systèmes de décision qui utilisent cette modélisation. Une fonction d'appartenance est ainsi employée pour la description d'un ensemble flou. La logique floue a été introduite et élaborée par Zadeh à partir de 1965 (Zadeh, 1965). Son champ d'application est vaste, à savoir la commande des processus industriels (tels que la robotique, l'énergie, le transport,…), les systèmes experts,

l'apprentissage, les systèmes automatiques, la productique (tels que la gestion de projet, l'ordonnancement,…), l'aide à la décision, etc.

L'utilisation de la logique floue en *ordonnancement* n'a pas pour but de proposer une nouvelle méthode, mais d'apporter des techniques qui vont permettre d'être plus proche de la réalité (Letouzey, 2001). Elle a été employée pour modéliser les environnements incertains (Bouchon-Meunier, 1995), ou encore des paramètres flous des travaux (ou opérations) comme les durées d'exécution. De même, elle a été appliquée à l'aide à la décision multicritère.

2.3 *STRUCTURES DES PROBLÈMES D'ORDONNANCEMENT D'ATELIER*

Nous utiliserons désormais le terme machine pour désigner une ressource. Différents types de problèmes d'ordonnancement d'atelier existent selon le nombre et la nature des machines ainsi que l'ordre d'enchaînement des opérations (Graham et al., 1979; Lawler et al., 1993; Blazewicz et al., 1996). Rappelons, qu'une gamme de fabrication précise l'ordre de réalisation des opérations d'un travail (produit ou pièce) sur l'ensemble des machines ainsi que le nombre d'opérations relatives à chaque travail. Elle permet ainsi de préciser les relations de précédence entre les opérations. Elle fournit, en outre, les conditions de fabrication comme les durées opératoires. Si le choix des machines pour les opérations n'est pas imposé, la gamme est dite ***logique***. Dans le cas inverse, on parle de gamme ***technique***. Pour les gammes techniques, nous distinguons suivant l'ordre d'exécution des opérations pour achever un produit :

- **la gamme libre :** si l'ordre est totalement libre ;
- **la gamme linéaire :** si l'ordre est entièrement imposé et déterminé ;
- **la gamme mixte :** si l'ordre est partiellement déterminé ;

Cette notion de gamme de fabrication est souvent utilisée comme critère de classification pour les problèmes d'ordonnancement. Deux grandes familles de problèmes d'ordonnancement se présentent. La première famille regroupe les problèmes pour lesquels chaque travail nécessite une seule opération. La deuxième regroupe ceux dont les travaux nécessitent plusieurs opérations. En se basant sur la configuration des machines, nous distinguons pour les premières catégories :

- ***Les problèmes à machine unique :*** où tous les travaux sont effectués sur l'unique machine.

- ***Les problèmes à machines parallèles :*** il s'agit d'une généralisation des problèmes à machine unique. Dans ce cas, l'atelier dispose de machines en plusieurs exemplaires pour le traitement d'un ensemble de travaux. Ainsi, le choix de la machine est à déterminer. Ces dernières peuvent être de différentes natures :
  - **machines parallèles identiques (P) :** c'est le cas des machines interchangeables pouvant exécuter tous les travaux. Le temps opératoire reste le même quelque soit la machine.
  - **machines parallèles uniformes (Q) :** dans ce cas, chaque machine a une vitesse de traitement propre, qui est identique à tous les travaux. Ainsi, la durée d'exécution d'un travail diffère selon la machine employée.
  - **machines parallèles indépendantes (R) :** il s'agit de machines ayant des vitesses de traitement propre qui dépendent des travaux à exécuter. Ainsi, la durée d'exécution d'un travail diffère selon la machine employée.

Pour les problèmes de la deuxième famille, trois ateliers spécialisées sont différenciés, à savoir :

- ***Les problèmes flow-shop (Atelier à cheminement unique)*** : l'atelier consiste à ordonnancer un ensemble de *n* travaux sur un ensemble de *m* machines disposées en série. Le cheminement des travaux est unique, c'est-à-dire que l'ordre d'exécution des opérations est le même pour tous les travaux. Tout travail étant composé de *m* opérations et chaque opération doit être exécutée par une machine selon un ordre de passage fixée à l'avance. Le flow-shop est dit de *permutation* s'il existe une contrainte selon laquelle la séquence des opérations des différents travaux est la même sur chaque machine (Potts et al., 1991).
- ***Les problèmes job-shop (Atelier à cheminements multiples)*** : il s'agit d'ordonnancer un ensemble de *n* travaux sur un ensemble de *m* machines tel que l'ordre d'exécution des opérations de chaque travail est fixé à l'avance, mais non nécessairement identique pour tous les travaux (donc chaque travail possède une gamme opératoire spécifique).
- ***Les problèmes open-shop (Atelier à cheminements libres)*** : l'atelier à cheminements libres consiste à ordonnancer un ensemble de *n* travaux sur un ensemble de *m* machines où l'ordre d'exécution des opérations élémentaires de chaque travail est totalement libre.

A partir de ces ateliers de base, de nombreuses extensions ont été proposées afin de traiter des problèmes industriels spécifiques. L'une des principales extensions consiste à proposer plus

d'une machine pour la réalisation d'une opération. Dans le cas d'un atelier flow-shop, l'ajout d'un ensemble de machines en parallèle sur chaque centre (ou étage) pour exécuter les opérations est appelé atelier ***flow-shop hybride***. De même pour les ateliers de type job-shop et open-shop qui donnent respectivement le job-shop flexible et l'open-shop généralisé. Ces types d'atelier offrent ainsi plus de flexibilité par rapport aux modèles classiques et nécessitent la détermination des affectations adéquates des opérations aux machines en plus de l'établissement des ordres de passages des différentes opérations sur les machines. Les machines en parallèle peuvent être identiques, uniformes, indépendantes ou encore une combinaison entre ces dernières. Le schéma (Figure 1.1) présente, d'une manière non exhaustive, une typologie de ces problèmes d'ordonnancement d'atelier qui a été adaptée de MacCarthy et Liu (1993). Elle résume les différents modèles des ateliers théoriques exposés ci-dessus.

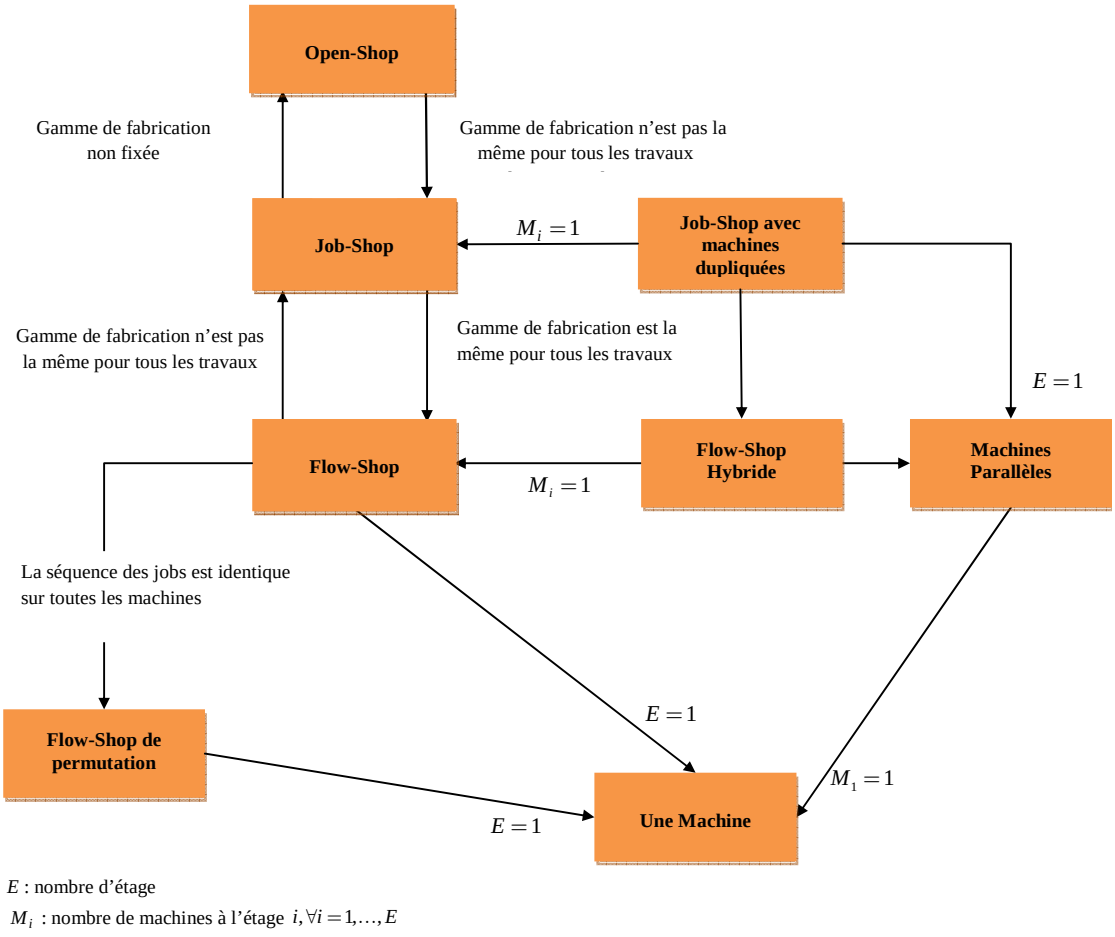

Figure 1.1 – Typologie des problèmes d'ordonnancement

En se basant sur cette typologie, une notation proposée par (Rinnooy Kan, 1976) (Voir aussi (Graham et al., 1979; Blazewicz et al., 1996)) est couramment utilisée pour distinguer un problème d'ordonnancement de manière synthétique et précise. Elle est composée de trois champs d'identification, qui sont notés par le triplet $\alpha / \beta / \gamma$. Le champ α permet de décrire l'environnement des machines. Il est constitué par deux paramètres α_1 et α_2 : le premier sert à représenter les caractéristiques des machines ou la nature de l'atelier (machine unique, flow-shop, …) et le second permet d'indiquer le nombre de machines utilisées. Le champ β indique les caractéristiques et les contraintes des travaux. Le dernier champ γ permet de spécifier le critère d'optimisation considéré. Quelques exemples de valeurs classiques des champs α_1, β et γ sont donnés dans le Tableau 1.1.

Tableau 1.1 – Quelques exemples des champs α_1, β et γ

Champ	***Valeur***	***Descriptif***
α_1	*Ø*	machine unique
	P	machines parallèles identiques
	Q	machines parallèles uniformes
	R	machines parallèles indépendantes (non reliées)
	F	les machines dédiées fonctionnent en flow-shop
	J	les machines dédiées fonctionnent en job-shop
	O	les machines dédiées fonctionnent en open-shop
β	*Pmtn*	autorisation de préemption des travaux
	Prec	existence de contraintes de précédence entre les travaux
	r_j	une date de début est associée à chaque travail *j*
	d_j	une date d'échéance est associée à chaque travail *j*
γ	C_{max}	la durée totale de l'ordonnancement (makespan)
	T_{max}	le plus grand retard vrai

Nous nous intéressons dans cette thèse aux problèmes d'atelier flow-shop hybride. Ainsi, quatre problèmes sont étudiés. Ils différent par les critères que l'on souhaite minimiser, qui sont :

- la durée totale de l'ordonnancement ;
- la somme totale et pondérée des avances/retards ;

– l'association simultanée des critères précédents (cas multicritère) ;

2.4 REPRÉSENTATION D'UN ORDONNANCEMENT

En ordonnancement, il est utile de disposer d'une représentation graphique plus précise des solutions établies. Ce but est atteint par l'utilisation de diagramme de Gantt. Ce diagramme constitue un formalisme graphique qui a été mis au point par Henry Gantt en 1910. Il s'agit d'un outil qui est couramment utilisé pour représenter la solution d'un problème d'ordonnancement. Il permet de visualiser l'utilisation des machines, l'avancement de l'exécution des opérations sur celles-ci ainsi que les dates de début et de fin de chaque opération. La Figure 1.2 représente le diagramme de Gantt associé à l'ordonnancement de trois travaux (J_1, J_2, J_3) dans un atelier flow-shop à deux machines. Les durées opératoires sont données dans le Tableau 1.2.

Tableau 1.2 – Temps opératoires pour l'exemple d'un flow-shop

	J_1	J_2	J_3
m_1	4	2	4
m_2	2	6	4

À partir de ce diagramme on peut déterminer par exemple la valeur de la date d'achèvement du travail le plus tardif.

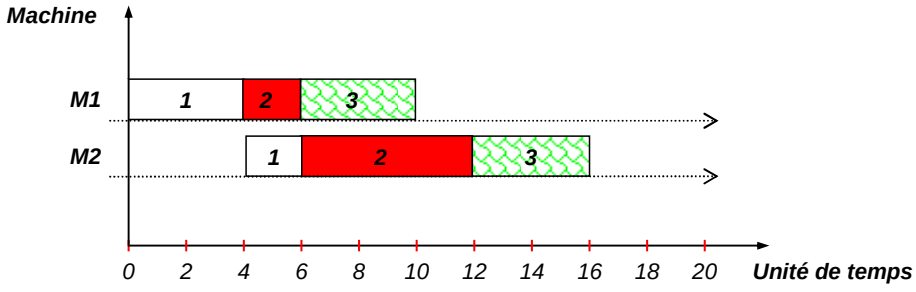

Figure 1.2 – Exemple d'un diagramme de Gantt

3 CONCLUSION

Dans ce chapitre introductif, nous avons rappelé les principales définitions et notations utilisées dans la résolution des problèmes d'ordonnancement en précisant les différents éléments qui les déterminent ainsi que leur classification en se basant sur les caractéristiques des problèmes comme l'organisation de l'atelier. Parmi ces ateliers, nous nous intéressons à l'étude du flow-shop hybride.

De même, ce chapitre nous a permis de passer en revue un ensemble de méthodes existantes pour résoudre les différents problèmes ordonnancement. Elles sont divisées en deux grandes classes, selon qu'elles les résolvent de manière exacte ou approchée. La première classe requière des ressources informatiques importantes dès que la taille du problème augmente. La deuxième permet d'offrir le plus souvent un bon compromis entre la qualité des solutions et le temps de calcul. En effet, ces dernières sont largement utilisées en ordonnancement, et figurent parmi les plus performantes. Dans cette thèse, nous nous focalisons sur ce type de méthodes, et plus particulièrement, aux méthodes dites à base de modèles pour la résolution de problèmes d'ordonnancement de type flow-shop hybride.

Dans le prochain chapitre, nous définissons le problème d'ordonnancement flow-shop hybride, puis nous détaillerons les méthodes à base de modèles.

Chapitre 2

ÉTAT DE L'ART : PROBLÈME FLOW-SHOP HYBRIDE ET MÉTA-HEURISTIQUES À BASE DE MODÈLES

Dans ce chapitre, nous décrivons le problème d'ordonnancement que nous traitons dans cette thèse : le flow-shop hybride. Nous évoquons les notions, notations, ainsi que les principales méthodes existantes pour sa résolution. Puis nous présentons, de manière approfondie les approches méta-heuristiques à base de modèles. Il s'agit de méthodes qui génèrent des solutions, non pas à partir des précédentes solutions mais à partir d'informations réunies dans un modèle. Une modélisation organisationnelle de ce type de méta-heuristiques est ainsi proposée.

1 INTRODUCTION

Dans cette thèse, nous nous intéressons à la résolution du problème d'ordonnancement de type flow-shop hybride (FSH). Il s'agit d'une extension de l'atelier de production flow-shop classique. L'éventail des différents problèmes du flow-shop hybride est très large et se rapproche des applications réelles. En effet, il constitue un modèle théorique pour résoudre des problèmes issus directement du monde industriel (Gourgand et al., 2001; Lopez & Roubellat, 2001). Ainsi, nous rencontrons dans la littérature, plusieurs problèmes des systèmes industriels modélisables par ce type d'atelier. A titre indicatif nous citons le cas de l'industrie du verre (Paul, 1979; Chevalier et al., 1996; Richard et al., 1998), l'industrie métallurgique (Narasimhan & Panwalkar, 1984; Narasimhan & Mangiameli, 1987), l'industrie du papier (Sherali et al., 1990), l'industrie chimique (Fortemps et al., 1996), l'industrie textile (Aghezzaf et al., 1995), l'industrie du bois (Riane et al., 1998) et l'industrie aérospatiale (Li, 1997) (pour plus de détails voir (Lopez & Roubellat, 2001)).

Les différents problèmes d'ordonnancement de type FSH sont classés parmi les problèmes NP-difficiles. Par conséquent et comme beaucoup de problèmes dits difficiles, les méthodes mathématiques restent limitées. Le recours aux méthodes approchées est plus fréquent pour la résolution de ces problèmes, notamment les méta-heuristiques. Ces dernières représentent un domaine en pleine effervescence. Elles sont souvent inspirées de processus biologiques ou physiques naturels qui doivent être adaptés selon le problème à résoudre. De même, elles utilisent des techniques génériques pouvant optimiser une large gamme de problèmes différents, sans changements majeurs dans les algorithmes de base. L'avantage de ce type de méthodes est de construire un grand nombre de solutions possibles en un temps raisonnable, et une réutilisation des informations colletées durant la recherche via l'évaluation des solutions générées. D'après Laguna (Laguna, 2002), une méta-heuristique est une stratégie principale qui guide d'autres heuristiques. Leurs principes communs et essentiels sont exprimés par les concepts classiques d'intensification et de diversification, d'une part, et par les notions de stratégies de recherche et d'adaptation et/ou d'auto-adaptation des mécanismes de recherche appliqués. Elles utilisent des processus de génération itératifs qui guident une heuristique sous-jacente en nuançant l'intensification et la diversification de l'espace de recherche. De plus, les méta-heuristiques induisent un aspect de multiplicité pour certains cas (comme pour le cas des algorithmes génétiques), par l'utilisation dans la recherche d'une population de solutions. Parmi les différentes méta-heuristiques rencontrées dans la littérature

(cf. Chapitre1), nous nous intéressons aux méta-heuristiques dites à base de modèles (Zlochin et al., 2004) et plus particulièrement à l'optimisation par colonie de fourmis. Il s'agit d'approches qui génèrent de nouvelles solutions, à partir d'informations réunies des précédentes solutions dans un modèle.

Dans la section 2, une modélisation des ateliers de production FSH est présentée. L'objectif sera de donner quelques définitions, notations et termes utiles pour la suite. Nous proposerons, ensuite un panorama des études faites sur ce type de problèmes dans le cas monocritères et multicritères. La section 3 porte sur une présentation des méthodes à base de modèles et son domaine d'application en optimisation combinatoire et plus particulièrement en ordonnancement.

2 PRÉSENTATION DU FLOW-SHOP HYDRIDE

2.1 *DÉFINITIONS*

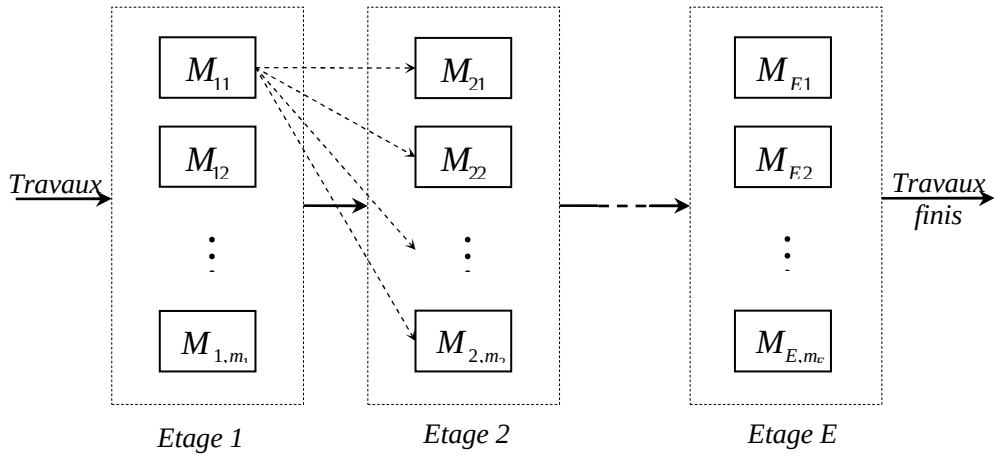

Figure 2.1 – Structure d'un atelier flow-shop hydride multi-étage

Un atelier flow-shop hybride, également appelé flow-shop multiprocesseur, représente un problème dans lequel, d'une part, le cheminement de l'atelier est le même pour tous les travaux (le flot des travaux à travers l'atelier est unidirectionnel) et d'autre part, les machines sont en exemplaire multiples et regroupées en étages (dites aussi centres ou cellules) disposées en séries de production (Linn & Zhang, 1999; Lopez & Roubellat, 2001) (voir Figure 2.1). Les machines sur chaque étage peuvent être identiques, uniformes, ou non reliées.

Dans notre travail, nous considérons que les machines sur chaque étage sont identiques. Dans la littérature, le flow-shop hybride est référé par FSH.

2.2 *MODÉLISATION*

Les différentes notations (voir Tableau 2.1) que nous utilisons sont celles présentées dans la plupart de la littérature. Formellement un problème d'ordonnancement flow-shop hybride est défini comme étant un ensemble $J = \{1,2,\ldots,n\}$ de n travaux à ordonnancer sur E étages consécutifs de production dans le même ordre, en commençant à l'étage 1 jusqu'au dernier étage. Chaque étage i $(i = 1, 2, \ldots, E)$ est composé d'un ensemble $M_i = \{M_{i1}, M_{i2}, \ldots, M_{i,m_i}\}$ de m_i $(m_i \geq 1)$ machines identiques en parallèle, tel qu'au moins l'un de ces étages est composé de plus d'une machine. Chaque travail j est composé d'une suite d'opérations O_{ij} $(i = 1, 2, \ldots, E)$. Une opération O_{ij} n'est exécutée que sur une machine à choisir parmi M_i. La durée opératoire de l'opération O_{ij} sur une machine $k \in M_i$ est notée p_{ij}. L'ordre de passage des opérations pour chaque travail j est le même et l'opération $O_{i+1,j}$ ne peut commencer que si l'opération précédente O_{ij} est terminée.

Résoudre un tel problème revient à trouver un ordonnancement réalisable pour exécuter les n travaux sur les machines tout en respectant les contraintes de précédence entre les opérations d'un même travail. Le problème consiste à affecter une machine à chaque opération (problème d'affectation) et à déterminer l'ordre dans lequel les opérations doivent être séquencées sur les machines retenues (problème de séquencement) afin de minimiser un critère donné. Dans la théorie de l'ordonnancement, sont utilisés souvent les termes suivants : « séquence » et « ordonnancement » pour présenter ou décrire un problème d'ordonnancement. Ces termes sont souvent interchangeables. Pour mieux distinguer les nuances entre ces deux termes, il faut comprendre le sens de chaque mot. Ainsi, l'exécution d'une séquence de travaux se traduit par l'ordre de passage des travaux sur les machines. Dans une séquence, on n'a pas à préciser le temps de début ou de fin de chaque opération effectuée. Pour exécuter une séquence de travaux, on a besoin d'un calendrier contenant des informations concernant l'ordonnancement des travaux. Un ordonnancement donné permet de déterminer le temps de début et de fin d'exécution de chaque travail. Pour tout critère régulier, un séquencement ou ordre donné permet de déterminer les dates de début et de fin de chaque

travail (en calant l'ordonnancement au plus tôt). Dans cette thèse, lorsque nous considérons un critère calculé à partir des avances et des retards des travaux, notre critère n'est pas régulier. On peut donc passer à côté des solutions optimales en utilisant seulement des ordonnancements calés gauche. Comme la plupart des auteurs qui ont travaillé sur ce type de critères, nous ne considérons ici que des ordonnancements calés gauche. Les méthodes obtenues sont correctes lorsque les délais sont plutôt serrés et la charge des machines relativement importante, elles le seraient nettement moins si les délais étaient très dispersés sur l'horizon et la charge des machines faible par rapport à la capacité disponible sur l'horizon, car la somme des avances pourrait alors être importante.

Tableau 2.1 – Notations utilisées dans les problèmes d'ordonnancement flow-shop hybride

Variable	**Description**
i	indice de l'étage
j	indice du travail
J	ensemble des travaux
n	nombre maximal de travaux
E	nombre maximal d'étages
O_{ij}	opération du travail *j* se traitant à l'étage *i*
p_{ij}	durée opératoire de l'opération O_{ij}
M_i	ensemble de machine à l'étage *i*
m_i	nombre de machine à l'étage *i*
t_j	date de début d'exécution du travail *j*
r_j	date de disponibilité du travail *j* (release date)
d_j	date de fin souhaitée (ou date d'achèvement maximal souhaité) du travail *j* (due date)
C_{ij}	date de fin d'exécution du travail *j* se traitant à l'étage *i*
$C_{\max}$	date de fin de l'ordonnancement (makespan)
T_j	retard vrai du travail *j* (tardiness)
E_j	avance du travail *j* (earliness)

La classification des problèmes d'ordonnancement distingue les problèmes statiques versus les problèmes dynamiques d'une part, et d'autre part, déterministes versus stochastiques. Dans le cas où le problème à traiter est statique, il faut spécifier l'ensemble des *n* travaux.

Dans le cas déterministe, les données du problème, telles que les temps d'exécution des travaux, sont connues d'une manière précise. Dans cette thèse, nous restreindrons notre étude au problème du FSH statique et déterministe. Les hypothèses suivantes sont retenues :

- Les travaux sont indépendants les uns des autres ;
- Tous les travaux sont connus et disponibles dès le début de l'ordonnancement ;
- Les machines sont disponibles du début à la fin de l'ordonnancement ;
- Les machines sur chaque étage sont supposées identiques ;
- Chaque machine ne peut réaliser à un instant donnée qu'une seule opération à la fois ;
- Les durées opératoires sont déterministes et connues à l'avance ;
- Les temps de montages et de démontages sont inclus dans les temps opératoires ;
- Les temps de transport sont négligeables ;
- Les opérations peuvent attendre dans des stocks de capacité illimitée entre les étages ;
- La préemption des opérations n'est pas autorisée. Ainsi une opération ne peut être interrompue une fois que son exécution a débuté sur une machine et aucune autre opération ne pourra commencer sur cette machine tant que l'opération en cours ne soit terminée.

2.3 *NOTATIONS*

La notation $\alpha / \beta / \gamma$ (Rinnooy Kan, 1976) est insuffisante pour prendre en considération le problème du flow-shop hybride, ainsi elle a été reprise par (Vignier et al., 1999) dans le but de l'enrichir. L'extension proposée porte essentiellement sur le champ α qui décrit l'environnement des machines. Il se compose de quatre paramètres que l'on représente ainsi :

$$\alpha = \alpha_1 \alpha_2 \left(\left(\alpha_3 \alpha_4^{(i)} \right)_{i=1}^{\alpha_2} \right)$$

- α_1 représente les caractéristiques des machines, on le note FH dans le cas du flow-shop hybride ;
- α_2 indique le nombre d'étages $\left(\alpha_2 = E\right)$;
- $\alpha_3 \in \left\{ \emptyset, \mathrm{P}, \mathrm{Q}, \mathrm{R} \right\}$ précise le type des machines sur l'étage *i* : s'il s'agit de problème à une

machine ($\varnothing$) ou à machines parallèles qui sont soit identiques (P), soit uniformes (Q), ou soit encore, non reliées (R) ;

- $\alpha_4 = m_i$ est le nombre de machines à l'étage *i*.

Le champ β permet, comme dans les notations classiques, de définir l'ensemble des contraintes et le champ γ indique le critère que l'on cherche à optimiser.

L'illustration de cette notation par un exemple de flow-shop hybride à 5 étages, composé de 5 machines identiques aux deux premiers étages, de 2 machines identiques au troisième étage, et de 3 machines identiques pour les autres et dont le critère à minimiser est le $C_{\max}$, est noté de la façon suivante : $FH5\left(P5^{(1)}, P5^{(2)}, P2^{(3)}, P3^{(4)}, P3^{(5)}\right) \mid \mid C_{\max}$.

2.4 *ÉTAT DE L'ART : PROBLÈME DU FLOW-SHOP HYBRIDE*

Les problèmes d'ordonnancement de type flow-shop hybride ont fait l'objet de plusieurs études depuis les années 1970. Ils sont classés parmi les problèmes les plus difficiles et les plus rencontrés dans le domaine industriel. Les premiers résultats concernant la complexité du FSH ont été présentés dans (Gupta, 1988) et ont prouvé qu'il s'agit d'un problème NP-difficile au sens fort pour le cas à deux étages, où un étage comporte deux machines et l'autre une seule machine, et avec pour critère d'optimisation la minimisation de la durée totale de l'ordonnancement $C_{\max}$. De même Hoogeveen et al. (1996) ont démontré que ce même problème est NP-difficile dans le cas où la préemption est autorisée.

Dans un premier temps, le flow-shop hybride à deux étages a suscité l'attention de beaucoup de chercheurs. Ensuite, les travaux se sont orientées vers une généralisation au cas multi-étages ($E \geq 3$), avec un nombre de machines quelconques disponibles sur chacun des étages. Les approches de résolution sont extrêmement variées dans la littérature. Nous ne faisons pas ici un état de l'art exhaustif, mais nous nous intéressons principalement aux méthodes et techniques proposées récemment dans le domaine des problèmes d'ordonnancement de type FSH monocritère et multicritère. Des états de l'art décrivant les différentes méthodes utilisées pour résoudre ce type de problèmes ont été proposés dans (Linn & Zhang, 1999; Vignier et al., 1999; Ruiz & Vázquez-Rodríguez, 2010).

2.4.1 Problème flow-shop hybride monocritère

De nombreux travaux proposent des méthodes exactes ou approchées pour résoudre le flow-shop hybride avec comme objectif la minimisation du makespan, du maximum des retards, du temps de séjour total (ou moyen) des travaux, ou encore du nombre de travaux en retard, etc. Dans cette section, nous dressons un résumé de quelques approches conçues pour les problèmes de type FSH monocritères.

Pour résoudre le FSH de manière optimale plusieurs techniques exactes sont proposées. Parmi celles-ci nous distinguons les procédures par séparation et évaluation qui sont les plus utilisées. Ils diffèrent généralement par leur fonction d'évaluation ainsi que par leur méthode de séparation. Nous citons à titre d'exemples les travaux de (Brah & Hunsuker, 1991; Gupta et al., 1997; Vignier et al., 1997; Brockmann & Dangelmaier, 1998; Moursli, 1999; Néron, 1999; Carlier & Néron, 2000; Haouari et al., 2006). Le critère d'optimisation pour ces travaux est la minimisation du makespan. Une modélisation en programmation linéaire entière pour le problème FSH multi-étages avec machines identiques dans chacun des étages est aussi présentée dans (Guinet et al., 1996). De même Riane et al. (1998) proposent un modèle mathématique pour la résolution du même problème à trois étages composé d'une seule machine sur le premier et le dernier étage et de deux machines sur le second étage. Dans leur cas d'étude, l'affectation des pièces sur les machines dépend de la dimension de la pièce à exécuter. Pour plus de détails sur les méthodes exactes utilisées pour résoudre le FSH voir (Kis & Pesch, 2005). Ces méthodes exactes ont été généralement utilisées pour résoudre des problèmes FSH de tailles petites ou moyennes.

Les méthodes approchées sont plus appropriées pour traiter les problèmes FSH de grandes tailles, et peuvent générer des solutions acceptables avec des temps raisonnables. De nombreux algorithmes approchés sont recensés dans la littérature pour le FSH monocritère. Ainsi, nous trouvons une variété d'études basées sur l'utilisation de différentes heuristiques, méta-heuristiques, ainsi que des approches hybrides et hyper-heuristiques.

Johnson (1954) était le premier à proposer un algorithme exact pour le problème flow-shop classique à deux machines pour la minimisation du makespan. Son algorithme a, en grande partie, influencé l'étude du FSH à deux étages. Il a permis de définir le séquencement des travaux qui minimise la durée totale de l'ordonnancement. La résolution du FSH considère aussi des décisions concernant l'affectation des opérations aux machines. Ainsi, plusieurs auteurs comme dans (Gupta, 1988; Sriskandarajah & Sethi, 1989; Gupta & Tunc, 1991; Deal

& Hunsucker, 1991; Lee et al., 1993) proposent des règles de priorités et/ou des heuristiques tout en mettant en œuvre l'algorithme de Johnson. Dans la plus part des cas, les heuristiques utilisées pour la résolution de ce type d'atelier sont souvent inspirées ou adaptées à partir d'heuristiques proposées initialement pour résoudre des problèmes flow-shop classiques. Dans (Lee & Vairaktarakis, 1994), les auteurs proposent une heuristique qui utilise un algorithme de liste appelé **FAM** (en anglais First Available Machine) pour le premier étage et un algorithme **LBM** (Last Busy Machine) au second. Santos et al. (1996) proposent pour le FSH à plusieurs étages avec pour critère la minimisation du makespan, une adaptation des heuristiques proposées dans (Palmer, 1965; Campbell et al., 1970; Gupta, 1971; Dannenbring, 1977) pour la résolution du flow-shop classique. De même, Guinet et al. (1996) s'intéressent à l'étude du même problème. Pour cela, ils le transforment en un problème de type flow-shop classique à *E* machines, puis ils appliquent l'heuristique de (Campbell et al., 1970) pour le résoudre. Dans (Vignier et al., 1996; Vignier, 1997), les auteurs proposent pour la résolution du problème FSH à deux étages des méthodes heuristiques en tenant compte de la disponibilité ou non des machines au premier étage. La fonction objectif étant la minimisation du plus grand retard (dans ce cas, des périodes d'inactivité des machines sont prises en compte). En outre, Soewandi et Elmaghraby (2001) développent trois heuristiques pour résoudre le flow-shop à trois étages avec minimisation du makespan, les machines sont supposées identiques sur chaque étage. Les deux premières méthodes subdivisent le problème en deux sous problèmes : un flow-shop hybride à deux étages et un problème d'ordonnancement à machines parallèles. L'autre approche se base sur le séquencement des travaux en premier, puis leur affectation aux différentes machines en second.

Santos et al. (1995) proposent une borne inférieure globale pour le flow-shop hybride avec machines identiques et minimisation du makespan. Cette borne est utilisée pour évaluer la qualité des solutions obtenues par des méthodes approchées lorsque la solution optimale est inconnue. Ainsi, pour chaque étage *i* s'exprime une borne inférieure de la manière suivante :

$$LB(i) = \frac{1}{m_i}\left\{\sum_{y=1}^{m_i} LSA(i,y) + \sum_{j=1}^{n} p_{ij} + \sum_{y=1}^{m_i} RSA(i,y)\right\} \qquad (2.1)$$

où $LSA(i,y)$ est la liste ordonnée des valeurs croissantes $LS(i,y) = \sum_{i'=1}^{i-1} p_{i'y}$ et $RSA(i,y)$ est la liste ordonnée des valeurs croissantes $RS(i,y) = \sum_{i'=i+1}^{i-1} p_{i'y}$.

Une autre borne inférieure en relation avec le makespan est donnée par la relation (2.2).

$$LB(0) = \max_{1 \le j \le n} \left\{ \sum_{i=1}^{E} p_{ij} \right\} \tag{2.2}$$

Ainsi, la borne inférieure globale pour le FSH est donnée par la relation :

$$LBMAX = \max\left\{ LB(0), \max_{i} \left\{ LB(i) \right\} \right\} \tag{2.3}$$

De nombreuses méta-heuristiques ont été également proposées pour la résolution du FSH. Citons par exemple les travaux de (Haouari & M'Hallah, 1997) dans lesquels sont présentés deux méta-heuristiques : le recuit simulé et la recherche taboue pour l'étude du flow-shop hybride à deux étages avec machines identiques sur chaque étage et pour objectif la minimisation du makespan. Ils supposent que le nombre de machines par étage est supérieur à un. Les résultats prouvent que leur méthode tabou est meilleure que celle du recuit simulé. De même, Gourgand et al. (1999) développent des méta-heuristiques à base du recuit simulé. Deux approches à base du recuit simulé, pour minimiser la durée totale de l'ordonnancement dans un atelier flow-shop hybride à *E* étages sont présentées dans (Jin et al., 2006). Les auteurs proposent également une borne inférieure pour le makespan. Cette borne a été reprise et rectifiée par la suite dans (Haouari & Hidri, 2008) pour proposer une borne inférieure valide. Pour résoudre ce même problème, une recherche tabou est appliquée dans (Nowicki & Smutnicki, 1998). Pour diminuer le temps de calcul, ils utilisent la notion du concept de blocs critiques. Pour l'ordonnancement d'un atelier FSH d'imprimerie de cartes électroniques à trois étages, Jin et al. (2002) proposent un algorithme génétique. Une autre approche basée sur un système immunisé artificiel (artificial immune system, AIS) est présentée dans (Engin & Döyen, 2004) pour résoudre le FSH multi-étages avec pour critère la minimisation du makespan. De même, Ben Hmida et al. (2007) proposent une recherche locale appelée « Climbing Depth-bounded Discrepancy Search » (CDDS) et basée sur des méthodes à divergences limitées. De même, un algorithme de fourmis a été présenté dans (Alaykýran et al., 2007). Cette dernière approche se base sur l'algorithme système de fourmis. Elle repose sur l'utilisation d'une solution de départ qui affecte la fonction de probabilité et, par conséquent, la qualité de la solution.

Les méthodes hybrides ont fait l'objet d'une multitude d'études pour la résolution de ce type d'atelier. Par exemple, Portmann et al. (1998) proposent un algorithme génétique hybride qui consiste à croiser un algorithme génétique dans une procédure par séparation et évaluation pour le cas le plus général (avec *E* étages). Leur méthode permet d'améliorer la recherche par

séparation et évaluation PSE proposé par (Brah & Hunsuker, 1991). Dans (Yang et al., 2000), les auteurs proposent trois méthodes approximatives pour résoudre le flow-shop hybride avec machines identiques sur chaque étage et comme critère de minimiser le retard total pondéré des travaux. La première approche est une méthode par décomposition, la deuxième utilise une recherche locale, et la troisième hybride ces deux premières approches. Les résultats prouvent que la méthode hybride surpasse les deux autres.

Selon les contraintes considérées et/ou les objectifs à optimiser, les approches de résolution du flow-shop hybride varient. Nous citons à titre d'exemple, l'étude du flow-shop hybride à opérations multiprocesseur[3]. Pour ce cas de problème, Ying et Lin (2006) proposent une adaptation de l'algorithme système de colonie de fourmis (ACS) qui construisent un ordonnancement en progressant dans le sens de l'axe des temps (« forward ») et un autre en remontant dans le temps (« backward »). Le critère d'optimisation est la minimisation de la durée totale de l'ordonnancement. La méthode utilise le ACS pour trouver un ordonnancement possible des travaux au premier étage, puis utilise la règle « premier arrivé, premier servi » pour les séquencer sur les étages suivants. Une fois que les fourmis artificielles ont terminé de construire les ordonnancements en commençant par le début, elles réalisent le chemin inverse pour obtenir les ordonnancements en partant de la fin. Plusieurs heuristiques constructives sont considérées pour exprimer l'heuristique de visibilité. Les tests sur des instances issues de la littérature ont démontré son efficacité par rapport à d'autres méthodes telles que la recherche tabou et les algorithmes génétiques.

D'autres part, dans (Bertel & Billaut, 2004), les auteurs étudient le cas d'un flow-shop hybride industriel à trois étages avec recirculation et ont présenté une heuristique pour minimiser le nombre moyen des travaux tardifs. De même, Ruiz et Maroto (2006) s'intéressent à un problème FSH réel minimisant le makespan d'un atelier de céramique avec des temps de latence dépendants de la séquence d'entrée. Les machines sont supposées non reliées et les travaux ne peuvent être traités sur toutes les machines disponibles, mais sur ce qu'on appelle une machine admissible (en anglais « machine eligibility »).

Une autre méthode combinant une méta-heuristique et une hyper-heuristique est proposée dans (Vázquez-Rodríguez & Salhi, 2007) pour résoudre le FSH sous différente formes, chacun avec un critère d'optimisation différent.

[3] c'est-à-dire les opérations des différents travaux peuvent être exécutées simultanément par plusieurs machines à la fois sur un même étage

2.4.2 Problème flow-shop hybride multicritère

L'optimisation multicritère et plus particulièrement l'ordonnancement multicritère est, sans aucun doute à l'heure actuelle, un domaine de recherche en plein essor tant du point de vue de la recherche théorique que des applications industrielles. En effet, la grande majorité des problèmes issus du monde réel sont de nature multicritère. Dans la littérature, un certain nombre de travaux de recherche s'est intéressé à l'étude des problèmes d'ordonnancement flow-shop hybride multicritère. Parmi ces travaux, nous citons ceux de Gupta et al. (2002), dans lesquels ils adaptent, pour le problème d'ordonnancement flow-shop hybride avec des durées opératoires contrôlables associées à chaque travail, des heuristiques existantes pour la résolution du flow-shop classique. Ils proposent de minimiser les sommes pondérées des avances, des retards, du makespan et des coûts associés aux dates d'échéances. De même, Janiak et al. (2007) développent trois méthodes : une recherche tabou, un recuit simulé et une approche hybride combinant les deux premières pour minimiser la somme totale moyenne des retards, des avances et des temps d'attente. D'autres travaux traitent le FSH avec machines parallèles identiques pour minimiser la somme des coûts de l'avance/retard. Ainsi dans (Fakhrzad & Heydari, 2008), les auteurs proposent une heuristique employant la règle de priorité EDD et une approche hybridant une recherche tabou et un recuit simulé en supposant que le nombre de machines sur chaque étage est le même. Dans le cadre d'une application industrielle, un modèle FSH simplifié est aussi présenté dans (Finke et al., 2007), supposant que les différentes affectations sont connues à l'avance. Dans ce cas les auteurs proposent une procédure de recherche tabou pour minimiser la somme totale des avances et des retards. Pour résoudre ce même problème avec temps de préparation dépendant de la séquence, Behnamian et al. (2010) proposent une approche hybride qui intègre une méthode Max-Min système de fourmis, une recherche à voisinage variable et le recuit simulé. Trois méta-heuristiques (un algorithme génétique, un recuit simulé et une recherche tabou) sont aussi développées pour résoudre le flow-shop hybride avec machines non reliées sur chaque étage (Jungwattanakit et al., 2009). La fonction objectif considère deux critères : le makespan et le nombre des travaux en retard sous la forme d'une somme convexe.

3 MÉTA-HEURISTIQUE À BASE DE MODÈLES : OPTIMISATION PAR COLONIE DE FOURMIS

3.1 *ORIGINE*

Depuis plusieurs années, les entomologistes se sont beaucoup intéressés à l'étude des comportements collectifs chez les insectes sociaux, en particulier chez les fourmis. En effet, ces insectes vivent et s'organisent en colonies. Ils montrent une grande capacité à solutionner des problèmes complexes d'une manière très flexible, en s'adaptant aux brusques changements de leurs environnements. Les études, menées par Deneubourg et al. (1983), ont permis de comprendre les mécanismes intervenant dans la coordination des activités collectives chez ces insectes. De même, ces études ont permis d'identifier les comportements individuels, ainsi que les interactions entre ces individus, qui produisent les comportements collectifs observés au sein de ces sociétés.

Des constatations ont montré que les fourmis, bien qu'elles aient individuellement des capacités cognitives très limitées et qu'elles soient pour certaines espèces aveugles (aucune information visuelle du monde environnant ne leur parvient), sont capables de sélectionner collectivement le plus court chemin entre une source de nourriture et leur nid. La coordination des activités collectives chez les fourmis est due à des mécanismes de communications indirectes via une substance volatile chimique appelée « phéromone », à laquelle les fourmis sont très sensibles. En effet, en se déplaçant, ces dernières communiquent entre elles de façon locale et indirecte en déposant sur le sol des traces de phéromones. Par ce marquage naturel, elles incitent ses congénères à suivre le même trajet. Les fourmis qui empruntent les sentiers les plus courts arrivent rapidement à se procurer de la nourriture et rentrent au nid en déposant de la phéromone sur le même chemin augmentant ainsi les quantités de phéromones comme le montre la Figure 2.2. De plus, comme la phéromone s'évapore avec le temps, le chemin le plus long est de moins en moins emprunté et sa trace disparaît presque complètement. Par conséquent, les sentiers les plus courts seront les plus concentrés en phéromones.

Au début, les fourmis explorent différents chemins en effectuant des déplacements aléatoires. Une fois qu'un chemin menant à une source de nourriture est découvert, elles y déposent une quantité de phéromone renforçant ainsi son importance et la probabilité d'être choisi par d'autres fourmis de la colonie. D'un autre côté, les mauvais chemins auront tendance à être oubliés, voire même, disparaître avec l'évaporation de la phéromone. Ce procédé est basé sur

le mécanisme de rétroaction positive. Il assure que les fourmis utilisent la voie d'accès la plus courte car elle sera la plus imprégnée par la phéromone.

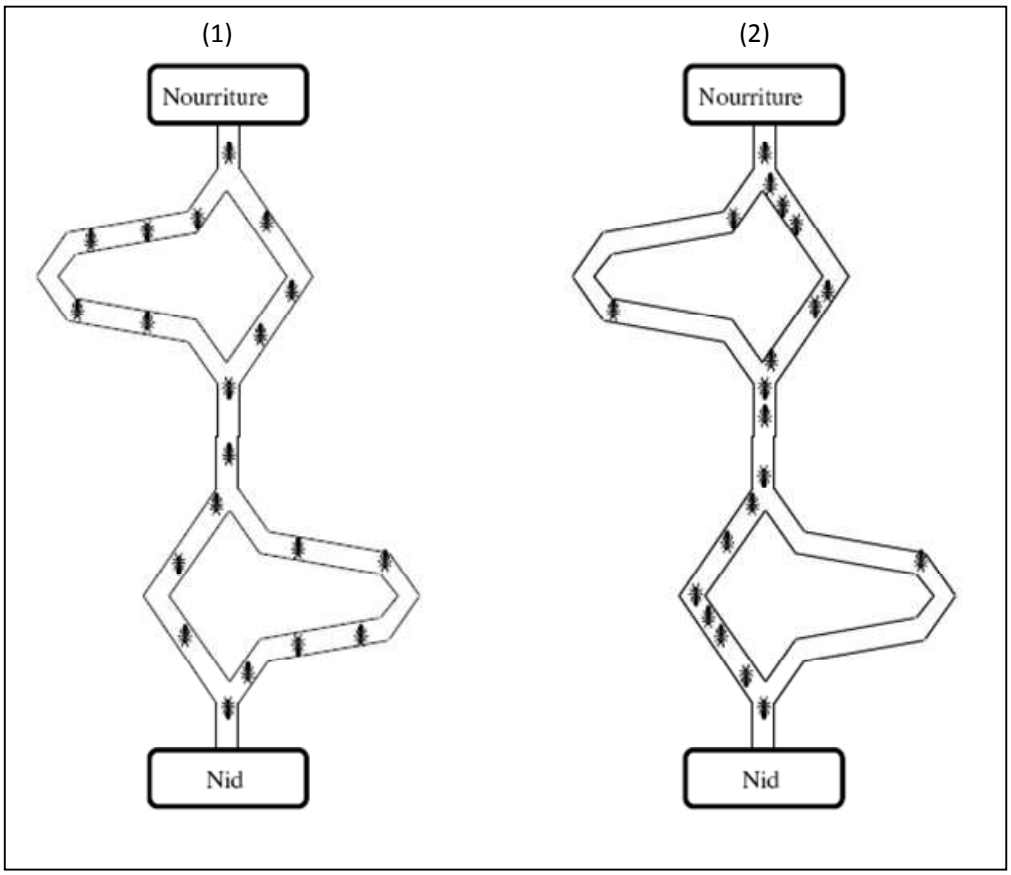

Figure 2.2 – Expérience de sélection du plus court chemin

(1) au début de l'expérience les fourmis explorent indifféremment les deux branches du pont (2) à la fin de l'expérience les fourmis ont tendance à emprunter le même chemin (qui est le plus court).

3.2 *OPTIMISATION PAR COLONIE DE FOURMIS*

Les algorithmes de colonie de fourmis ont été introduits pour la première fois pour résoudre des problèmes d'optimisation combinatoire (Colorni et al., 1991a; Colorni et al., 1991b; Colorni et al., 1992) au début des années 90 suite aux travaux de Deneubourg et al. (1983). Ces algorithmes ont été regroupés sous le nom générique d'« optimisation par colonies de fourmis » (ACO pour Ant Colony Optimisation). A l'origine, l'optimisation par colonie de fourmis a été conçue pour résoudre le problème du voyageur de commerce (PVC). L'objectif de ce problème est de trouver la tournée la plus courte en passant une seule fois par un

nombre donné de villes. Pour ce faire, un algorithme imitant le comportement de fourmis réelles, lors de la quête de nourriture, pour trouver le plus court chemin est proposé. L'ACO consiste à travailler sur une population de solutions, chacune correspondant à une fourmi artificielle. Une structure de donnée commune, qui renferme des informations sur les quantités de phéromones artificielles accumulées dans l'espace de recherche, est utilisée pour coordonner le fonctionnement de la population. Ainsi, par analogie aux comportements des fourmis réelles, les algorithmes de colonie de fourmis sont basés sur la communication indirecte d'une colonie de fourmis artificielles. La quantité de phéromones représente une information utilisée par les fourmis pour former une solution de manière stochastique à un problème donné. Il s'agit donc d'une méta-heuristique de construction aléatoire biaisée par un ensemble de données globales représentant une mémoire sur la qualité des solutions via les traces (ou quantités) de phéromones. Cet ensemble est régulièrement mis à jour par un mécanisme simulant l'évaporation de la phéromone. Il effectue des décisions probabilistes fonction des quantités de phéromones artificielles et d'une vision locale du problème via une heuristique d'information dite aussi visibilité.

D'après Dorigo et Stützle (2004), l'ACO peut être défini comme une extension d'une heuristique de construction traditionnelle, qui en plus, possède une adaptation de la quantité de phéromones durant l'exécution de l'algorithme pour prendre en considération l'expérience de la recherche. Les méthodes de recherche par colonie de fourmis sont évolutives. A la différence des algorithmes génétiques, elles ont la particularité de rechercher des solutions en se basant sur la totalité de la population et non pas à partir de quelques meilleurs individus et en s'appuyant sur leurs expériences collectives.

Pour mieux comprendre le fonctionnement de ces algorithmes, nous proposons, dans ce qui suit, un aperçu non exhaustif sur les premières approches reproduisant le comportement naturel des fourmis lors de la quête de nourriture pour traiter le PVC. Le Tableau 2.2 résume les termes de la notation utilisés pour ces algorithmes par la suite.

Dans ces approches, chaque fourmi artificielle construit des solutions en se basant sur un modèle graphique représentant le problème. Dans ce cas, les sommets (ou nœuds) du graphe représentent les *n* villes et les arêtes indiquent les trajets entre ces dernières. L'objectif est de chercher, grâce à un ensemble de *nbAnt* fourmis artificielles, les chemins les plus courts reliant ces villes, et telle que chaque ville ne soit visitée qu'une seule fois. Une fourmi artificielle se distingue par les caractéristiques suivantes (Dorigo et al., 1996).

Tableau 2.2 – Notations pour les algorithmes de fourmis

Variable	**Description**
τ_{ij}	quantité de phéromone présente entre deux nœuds *i* et *j*
η_{ij}	visibilité locale entre *i* et *j*
$\Pr_{ij}$	probabilité de choisir un élément *j*, connaissant *i*
$\Delta\tau_{ij}$	quantité de phéromone ajouté à τ_{ij}
L_k	liste des éléments déjà placés pour la fourmi *k* (*liste tabou*)
S_k	liste des éléments non encore placés par la fourmi *k*
t	compteur pour les itérations
Paramètre	**Description**
nbAnt	nombre de fourmis
$t_{\max}$	nombre maximal d'itérations
τ_0	quantité de phéromone initiale
α	paramètre modulant la quantité de phéromone
β	paramètre modulant la visibilité
q_0	variable aléatoire uniformément distribuée sur $[0,1]$
q	nombre aléatoire uniformément distribuée sur $[0,1]$
ρ	coefficient d'évaporation des quantités de phéromone
ρ_ℓ	coefficient d'évaporation locale des quantités de phéromone
ρ_g	coefficient d'évaporation globale des quantités de phéromone

- Elle n'est pas totalement aveugle. Elle est dotée d'un champ de vision lui permettant de voir un peu plus loin que son entourage direct ; cela est réalisé grâce à une valeur appelée la visibilité (ou encore valeur heuristique ou heuristique d'information). Cette valeur peut influencer le choix fait par une fourmi. Il s'agit généralement d'une donnée propre au problème.
- Elle évolue dans un univers où le temps est discret.
- Elle dispose d'une certaine mémoire utilisée pour stocker le trajet effectué permettant ainsi de retenir la solution qu'elle a construit.
- Elle gère la quantité de phéromones déposée en fonction de la qualité de la solution ; de plus il est possible de procéder à différents types de dépôts de phéromones. Cette procédure permet de mettre en place une mémoire adaptative décrivant l'état du système.

3.2.1 Algorithme système de fourmis

Le premier algorithme qui a été conçu pour le PVC porte le nom de système de fourmis (AS pour « Ant System »)(Colorni et al., 1991a). Il s'agit d'un algorithme d'optimisation distribué qui à chaque itération *t*, chaque fourmi *k* parcourt les arêtes du graphe en se déplaçant d'un nœud à un autre construisant ainsi un trajet complet. Pour effectuer un déplacement, on doit définir :

- ***La quantité de phéromone*** $\tau_{ij}(t)$ sur l'arête (i, j) reliant deux villes *i* et *j*. Elle donne l'intensité de la trace de phéromone artificielle sur le chemin (i, j) qui permet d'informer les fourmis sur l'importance de ce trajet. Ce paramètre dynamique définit l'attractivité ou encore la désirabilité de sélectionner la ville *j* comme une prochaine destination après la ville *i*. C'est, en quelque sorte, une mémoire globale du système, qui évolue par apprentissage ;
- ***La visibilité*** η_{ij} qui représente l'inverse de la distance entre les villes. Elle permet de diriger le choix des fourmis vers des villes proches et d'éviter les villes lointaines. Il s'agit d'une heuristique d'information pour choisir la ville *j* à partir de la ville *i* ;
- Une liste, dite ***liste tabou*** $L_k(i)$, constituant une mémoire des villes déjà visitées pour éviter que la fourmi *k*, située au sommet *i*, revisite une même ville. De plus, elle permet de sauvegarder le chemin parcouru par cette fourmi.

A chaque itération *t*, chaque fourmi *k* choisit sa prochaine destination *j* suivant une probabilité qui dépend de la distance séparant cette ville et sa position *i* et de la quantité de phéromone présente sur cette arête. La règle de transition d'une ville *i* vers une ville *j* est donnée par la relation (2.4) suivante :

$$\Pr_{ij}(t) = \begin{cases} \dfrac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}\right]^{\beta}}{\sum_{\ell \notin L_k(i)} \left[\tau_{i\ell}(t)\right]^{\alpha} \cdot \left[\eta_{i\ell}\right]^{\beta}}, & \text{si} \quad j \notin L_k(i) \\ 0, & \text{sinon} \end{cases} \tag{2.4}$$

Où α et β sont deux paramètres modulant l'importance relative entre l'intensité de la quantité de phéromone τ_{ij} et la visibilité η_{ij}. Le choix des valeurs de ces deux paramètres permet de régler l'influence des comportements de diversification et d'intensification.

Algorithme 2.1 – Algorithme AS pour le PVC

Initialisation

placer aléatoirement chaque fourmi *k* sur un nœud *i*

initialiser les traces de phéromone à τ_0

initialiser $L_k(i)$

déterminer les différents paramètres de l'algorithme

Pour $t=1$ à $t_{\max}$ **Faire**

Pour chaque fourmi $k=1$ à $nbAnt$ **Faire**

Pour chaque ville non visitée **Faire**

sélectionner une ville $j \notin L_k(i)$ selon la formule (2.4)

ranger *j* dans $L_k(i)$

Fin Pour

calculer la longueur L^k du chemin décrit par la fourmi *k*

déposer une piste $\Delta\tau_{ij}^k$ sur le parcours T_k de la fourmi *k* selon la formule (2.5)

Fin Pour

Mise à jour des traces de phéromone selon la formule (2.6)

Fin Pour

Trois types d'algorithmes AS ont été proposés : le « ant-density », le « ant-quantity » et le « ant-cycle ». Selon le type de l'algorithme, le dépôt de phéromones se fait pendant la construction de la solution pour les deux premières approches. Dans l'autre cas, c'est-à-dire le « ant-cycle », chaque fourmi *k*, après avoir terminé la construction de sa solution, laisse une quantité de phéromones $\Delta\tau_{ij}^k$ sur le chemin choisit. Cette quantité dépend de la qualité de la solution construite. La valeur de $\Delta\tau_{ij}^k$ diffère d'un algorithme à l'autre. D'après les résultats donnés dans (Colorni et al., 1991a), l'algorithme « ant-cycle » est le meilleur. La quantité de phéromones est donnée dans ce cas par la relation suivante (2.5).

$$\Delta\tau_{ij}^k = \begin{cases} \dfrac{Q}{L^k}, & \text{si } (i,j) \in T_k \\ 0, & \text{sinon} \end{cases} \qquad (2.5)$$

avec *Q* une constante positive donnée, L^k la longueur de la solution (qui est la somme des distances d'une ville à une autre) et T_k le trajet effectué par la fourmi *k*.

Lorsque toutes les fourmis terminent leurs tournées, il est important de mettre au point un processus d'évaporation de la quantité de phéromone afin de donner de l'importance aux derniers choix réalisés. Ceci permet d'oublier les mauvaises solutions. La règle de mise à jour est donnée par la relation suivante :

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{k=1}^{nbAnt}\Delta\tau_{ij}^{k} \tag{2.6}$$

où ρ est un coefficient d'évaporation des traces de phéromone.

La valeur initiale des τ_{ij} est $\tau_0 \geq 0$ (la valeur de τ_0 est généralement très petite) et le nombre de fourmis est proposé égale au nombre des villes. L'Algorithme 2.1 donne la structure du pseudo-code de l'algorithme AS pour le PVC.

A partir de cette approche AS plusieurs autres extensions ont été présentées dans le but d'améliorer cette approche. Parmi celles-ci, nous citons les deux variantes les plus connues :

3.2.2 Algorithme système de colonie de fourmis

L'algorithme système de colonie de fourmis (ACS pour « Ant Colony System ») est une variante du premier algorithme AS proposé pour résoudre le PVC de grandes tailles. Il a été proposé par Dorigo et Gambardella (1997). Cette méthode repose sur les principes suivants :

- ***Une règle de transition*** (dite aussi règle proportionnelle pseudo-aléatoire) dépendant d'un paramètre $q_0\,(0 \leq q_0 \leq 1)$ qui définit une balance diversification/intensification. Ainsi, une fourmi *k* se trouvant sur une ville *i* choisira une ville *j* selon la relation :

$$j = \begin{cases} \arg\max_{u \notin L_k(i)} \left\{ \left[\tau_{iu}(t)\right] \cdot \left[\eta_{iu}\right]^{\beta} \right\}, & \text{si } q \leq q_0 \\ J, & \text{si } q > q_0 \end{cases} \tag{2.7}$$

  β est un paramètre modulant l'importance relative entre l'intensité de la quantité de phéromones τ_{ij} et la visibilité η_{ij}. La variable q est choisie de manière aléatoire uniformément distribuée sur $[0,1]$. Si $q \leq q_0$, le système tend vers une intensification, c'est-à-dire exploiter plus les informations récoltées par le système. Ainsi, le choix de trajet non exploré est faible. Dans le cas contraire, où $q > q_0$, le choix se fait de la même façon que pour l'algorithme AS, et le système tend à effectuer une diversification. Ainsi, J est sectionné aléatoirement selon la probabilité (2.8) :

$$\Pr_{iJ}(t) = \frac{\left[\tau_{iJ}(t)\right]\cdot\left[\eta_{iJ}\right]^{\beta}}{\sum_{u\notin L_k(i)}\left[\tau_{iu}(t)\right]\cdot\left[\eta_{iu}\right]^{\beta}} \tag{2.8}$$

- ***Une mise à jour locale*** : en construisant son tour, chaque fourmi *k* va modifier la quantité de phéromone sur chaque arête traversée en utilisant la relation (2.9) :

$$\tau_{ij}(t+1) = (1-\rho_{\ell}).\tau_{ij}(t) + \rho_{\ell}.\tau_0 \tag{2.9}$$

où τ_0 est la valeur initiale de la piste de phéromone et ρ_{ℓ} est le coefficient d'évaporation locale de phéromone. A chaque passage, la quantité de phéromones sur les arêtes visitées diminue, favorisant ainsi la diversification par la prise en compte des trajets non explorés.

- ***Une mise à jour globale*** : qui s'effectue à chaque itération de la façon suivante :

$$\tau_{ij}(t+1) = (1-\rho_g).\tau_{ij}(t) + \rho_g.\Delta\tau_{ij}(t) \tag{2.10}$$

$$\text{avec } \Delta\tau_{ij}(t) = \begin{cases} \left(L_{gb}\right)^{-1}, & \text{si } (i,j) \in \text{au meilleur tour de longueur } L_{gb} \\ 0, & \text{sinon} \end{cases}$$

Dans ce cas, seule la meilleure solution, de longueur L_{gb}, obtenue est mise à jour, ce qui participe à une intensification par sélection de la meilleure solution.

- Le système utilise ***une liste de candidat*** pour chaque fourmi *k*, notée S_k, représentant la liste des choix possibles à partir d'une ville. Elle permet de restreindre la liste des villes les plus proches non encore sélectionnées pour accélérer le processus de construction d'un chemin, et réduire le temps nécessaire au calcul des probabilités.
- Le système dispose aussi d'une liste tabou L_k, pour sauvegarder le chemin parcouru par chaque fourmi.

La description de cette approche est décrite dans Algorithme 2.2.

3.2.1 Algorithme MAX-MIN système de fourmis

Le Max-Min système de fourmis (MMAS pour MAX-MIN ant system) est une méthode dérivée de l'AS. Elle a été proposée par Stützle et Hoos (1997). Cette variante utilise deux constantes $\tau_{\min}$ et $\tau_{\max}$ comme borne inférieure et supérieure du taux de phéromones sur chaque arête du graphe. Ceci permet de ne pas favoriser certains chemins au profit d'autres.

Initialement la quantité de phéromones sur les chemins est égale à $\tau_{\max}$. Pour cette variante, seule la meilleure solution est mise à jour.

Algorithme 2.2 – Algorithme ACS pour le PVC

Initialisation

placer aléatoirement chaque fourmi sur un nœud *i*

Initialiser les traces de phéromone à τ_0, les listes $S_k(i)$ et $L_k(i)$

Déterminer les différents paramètres de l'algorithme

Pour $t = 1$ à $t_{\max}$ **Faire**

  Pour chaque fourmi $k = 1$ à $nbAnt$ **Faire**

    Pour chaque ville non visitée **Faire**

      sélectionner une ville $j \in S_k(i)$ selon la formule (2.7)

      ranger *j* dans $L_k(i)$

      Mettre à jour localement la quantité de phéromone selon la formule (2.9)

    Fin Pour

    Evaluer L^k

  Fin Pour

  Mettre à jour globalement les quantités de phéromone de la meilleure solution obtenue selon la formule (2.10)

Fin Pour

3.3 *DOMAINES D'APPLICATIONS*

Les algorithmes de fourmis ont subi de nombreuses modifications pour les adapter et les appliquer à des problèmes distincts, et pour améliorer leurs performances. Parmi ces modifications nous citons :

- l'hybridation avec d'autres heuristiques notamment les recherches locales, qui a souvent prouvé une meilleure efficacité ;
- la proposition de nouvelles règles de transition ;
- les modifications relatives aux choix d'un modèle pour la gestion des quantités de phéromones ;
- La modélisation d'heuristiques pour la visibilité, qui permet de mieux diriger la recherche.

Ainsi, la simplicité d'implantation de ces algorithmes et les bons résultats obtenus pour la résolution du PVC ont suscité l'attention d'autres chercheurs à les appliquer pour la résolution d'autres problèmes d'optimisation. Parmi ces problèmes, nous pouvons citer à titre indicatif : les problèmes de *tournées de véhicules, d'affectation quadratique, de coloration de graphes, de routages, de satisfaction de contraintes, du sac à dos,* etc. De même, plusieurs algorithmes de colonie de fourmis ont été développés pour résoudre différents problèmes d'ordonnancement.

Un résumé présentant quelques travaux récents issus de la littérature pour résoudre des problèmes d'ordonnancement avec des algorithmes de fourmis est présenté dans Tableau 2.3.

Tableau 2.3 – Applications des algorithmes ACO aux problèmes d'ordonnancement

Problème d'ordonnancement	**Références**
Machine unique	(Den Besten et al., 2000) (Gagné et al., 2002) (Ying & Liao, 2003) (Merkle & Middendorf, 2005) (Liao & Juan, 2007) (M'Hallah, 2007)
Machines parallèles	(Sankar et al., 2005) (Srinivasa Raghavan & Venkataramana, 2009)
Flow-shop	(Stützle, 1998) (T'kindt et al., 2002) (Shyu et al., 2004) (Ying & Liao, 2004) (Gajpal et al., 2006) (Ying & Lin, 2007) (Yagmahan & Yenisey, 2008)
Flow-shop hybride	(Ying & Lin, 2006) (Alaykýran et al., 2007)
Job-shop	(Colorni et al., 1994) (Van Der Zwaan & Marques, 1999) (Zhang et al., 2006)
Job-shop hybride et ou flexible	(Huang & Yang, 2008) (Rossi & Dini, 2007) (Rossi & Boschi, 2009)
Open-shop	(Blum & Sampels, 2002) (Blum, 2005)
Group-shop	(Blum & Sampels, 2004)

L'application de ce type d'algorithmes s'avère peu utilisé notamment pour les problèmes à machines parallèles, le flow-shop hybride, ainsi que le job-shop flexible. Ceci peut

s'expliquer par le fait que ces algorithmes ont la capacité de résoudre des problèmes d'ordonnancement liée uniquement à la détermination des séquencements possibles (comme pour le cas des machines uniques ou du flow-shop de permutation pour lesquelles les machines traitant chacune des opérations sont fixées dès le départ), par analogie à la résolution du PVC. Dans le cadre de cette thèse, notre intérêt porte sur l'ordonnancement d'atelier flow-shop hybride. Par rapport aux algorithmes de fourmis proposés dans la littérature pour la résolution du flow-shop hybride, comme dans (Alaykýran et al., 2007; Ying & Lin, 2006), notre contribution se manifeste par l'utilisation de modèles de construction différents. De même, nous proposons deux manières différentes concernant les décisions liées à l'affectation et au séquencement pour la résolution des problèmes considérés. Les critères, que nous traitons, sont le makespan comme dans (Alaykýran et al., 2007; Ying & Lin, 2006) et l'avance/retard. Notons également que Ying et Lin (2006) considèrent l'étude d'un atelier flow-shop hybride où les opérations peuvent être exécutées par plusieurs machines à la fois, ce qui n'est pas le cas dans cette thèse. Ils utilisent l'algorithme de fourmis pour résoudre le problème d'ordonnancement uniquement au premier étage. Une étude expérimentale et notamment une comparaison de nos résultats avec ceux de Alaykýran et al. (2007) pour le problème FSH avec minimisation du makespan sera présenté dans le Chapitre 3.

4 CONCLUSION

Ce chapitre nous a permis de passer en revue l'ensemble des méthodes de résolution existantes pour le flow-shop hybride. Il ressort de cet état de l'art que les méthodes exactes semblent réellement être limitées dans la résolution de ce problème. Devant ce constat, nous nous sommes focalisés sur des méthodes approchées permettant d'obtenir de bonnes solutions en un temps acceptable. Les méta-heuristiques représentent dans la plupart des cas des méthodes intéressantes pour les problèmes d'ordonnancement. Ainsi, nous nous sommes intéressées dans cette thèse à l'application des algorithmes dits à base de modèles aux problèmes flow-shop hybride dans les cas monocritère et multicritère.

Chapitre 3

MÉTA-HEURISTIQUES À BASE DE MODÈLES POUR LA RÉSOLUTION DU FLOW-SHOP HYBRIDE MONOCRITÈRE

Dans ce chapitre, nous étudions le problème d'ordonnancement flow-shop hybride avec machines parallèles identiques sur chaque étage. L'objectif est la minimisation de la durée totale de l'ordonnancement. Ce chapitre constitue la première partie de notre contribution. Il présente de manière approfondie des méthodes méta-heuristiques à base de modèles pour la résolution du flow-shop hybride monocritère. Ainsi, nous détaillerons les différentes méthodologies basées sur ces méta-heuristiques. Nous terminons par une série de tests pour évaluer les méthodes de résolution proposées.

1 INTRODUCTION

L'objectif de ce chapitre est d'explorer et de justifier l'application des ACO pour l'ordonnancement des ateliers flow-shop hybride. Ainsi, nous présentons les travaux que nous avons réalisés pour résoudre ce problème avec pour fonction objectif la minimisation de la durée totale de l'ordonnancement. Nos méthodes décrites ci-après ont fait l'objet des communications internationales suivantes : (Khalouli et al., 2007; Khalouli et al., 2008a; Khalouli et al., 2009).

2 APPLICATION DES ALGORITHMES DE FOURMIS AU PROBLÈME FLOW-SHOP HYBRIDE

Nous présentons dans cette partie, la conception et l'application de trois variétés d'approches à base d'ACO pour la résolution du problème $FHE,\left(Pm_i\right)_{i=1}^{E}\left|\right|C_{\max}$. La première approche, que nous appelons ***HACS-FSH***[4], est basée sur une méthode hiérarchique en deux phases. La première est consacrée à l'affectation des différentes opérations sur les machines. Dans la deuxième phase, le problème de séqencement est résolu en proposant des ordres de passage possibles aux opérations sur chacune des machines. Les décisions concernant l'affectation et le séquencement se font donc indépendamment l'une de l'autre. Quant, aux deux autres approches, nommées respectivement ***IMOACS-FSH***[5] et ***IOMACS-FSH***[6], elles utilisent une stratégie d'ordonnancement qui, en une seule phase, prend une décision concernant l'affectation et une décision concernant le séquencement (Dauzère-Pérès & Paulli, 1997). La distinction entre ces deux dernières approches se manifeste par l'ordre de disposition des étapes concernant la sélection d'une machine et celle d'une opération.

La conception de nos algorithmes est basée sur la structure de l'algorithme ACS (Dorigo & Gambardella, 1997). Ainsi, un certain nombre de décisions doit être pris en considération (Blum & Sampels, 2002) pour résoudre le problème à l'aide de cette technique, à savoir la détermination :

- d'une représentation adéquate du modèle de phéromones pour le problème considéré ;
- d'un mécanisme de mise à jour de la quantité de phéromones ;

[4] signifie méthode hiérarchique à base d'une approche ACS pour résoudre le FSH
[5] signifie méthode intégrée à base de ACS qui sélectionne une machine (***M***) puis lui attribue une opération (***O***) pour résoudre le FSH
[6] signifie méthode intégrée à base de ACS qui sélectionne une opération (***O***) puis l'affecte sur une machine (***M***) pour résoudre le FSH

- d'une représentation de la visibilité sous forme d'heuristiques d'information, qui permet de donner une vision locale au problème considéré.

Dans ce qui suit, nous exposons en détails les méthodes proposées.

2.1 *DESCRIPTION DE L'APPROCHE HIÉRARCHIQUE HACS-FSH*

L'approche *HACS-FSH* est une adaptation de l'algorithme ACS (Algorithme 2.2) (Dorigo & Gambardella, 1997) utilisée initialement pour le PVC. Le but est de définir une stratégie de recherche de façon à prendre en considération la modélisation d'un atelier FSH. L'idée de base consiste donc à représenter le problème sous forme de la recherche du meilleur chemin dans un graphe et en le résolvant de manière hiérarchique en deux temps. La première étape permet l'affectation des opérations aux machines. La détermination des décisions concernant le séquencement fait l'objet de la deuxième étape. Nous présentons, dans ce qui suit, l'algorithme général de cette méthode hiérarchique *HACS-FSH* (Algorithme 3.1) que nous avons conçu et développé et nous détaillerons les différents modules le constituant.

2.1.1 Représentation du problème

Nous représentons le problème d'ordonnancement flow-shop hybride sous la forme d'un graphe de recherche défini par le triplet $G = \{O, C, D\}$ dans lequel :

- O représente l'ensemble des sommets associés aux opérations O_{ij} de tous les travaux à exécuter sur les différents étages auxquelles sont rajoutées deux opérations fictives O_D^* et O_F^*, qui indiquent respectivement le début et l'achèvement de tous les travaux ;
- C étant l'ensemble des arcs conjonctifs (orientés) représentant les règles de précédence de la gamme opératoire entre les différentes opérations d'un même travail. $C = \left\{ \left(O_{ij}, O_{i+1,j} \right) \middle| \; O_{ij} \to O_{i+1,j} \right\} \cup \left\{ O_D^*, O_{1,j} \right\} \cup \left\{ O_{E,j}, O_F^* \right\}$ est l'ordre de passage des opérations du même travail j $(1 \le j \le n)$ sur les différents étages i $(1 \le i \le E)$. Ainsi, chaque couple d'opérations consécutives d'un même travail est représenté par un arc orienté.
- D représente l'ensemble des arcs disjonctifs (non orientés) qui décrit les contraintes d'utilisation des machines. L'orientation de ces arcs permet de déterminer un ordonnancement possible des opérations sur les machines.

Soit à titre d'exemple un problème FSH constitué de trois travaux et trois étages. Le nombre de machines par étage est respectivement égal à : $m_1 = 2,\ m_2 = 1$ et $m_3 = 2$. L'arc orienté entre les opérations O_{11} et O_{21} exprime une contrainte de précédence entre ces deux opérations du premier travail. La figure du graphe « partiellement arbitré » suivante (Figure 3.1) représente un ordonnancement possible à l'exemple considéré :

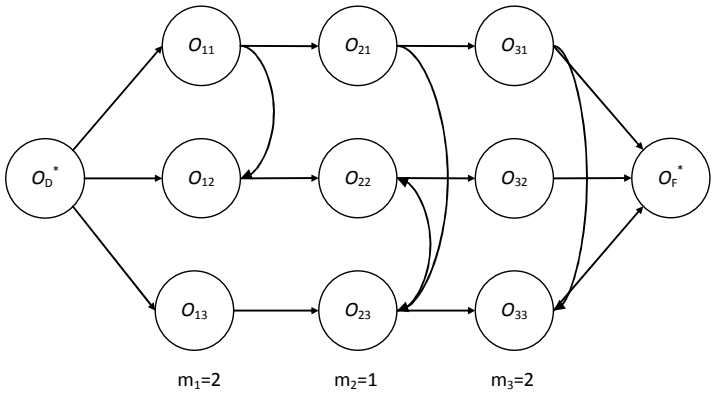

Figure 3.1 – Un ordonnancement possible associé au graphe de recherche

2.1.2 Affectation et séquencement des travaux

Pour résoudre un tel problème, il faut déterminer l'affectation de chaque opération à l'une des machines éligibles (vue la nature de l'atelier FSH un ensemble de machines est associé à chaque opération mais une seule machine doit l'exécuter). Il reste ensuite à trouver la meilleure séquence d'opérations s'exécutant sur chaque machine. Ainsi, l'algorithme que nous proposons consiste à structurer notre problème en deux phases : une première pour l'affection et une autre pour le séquencement.

Toute opération candidate à un placement se voit attribuer une machine selon une technique spécifique du générateur d'ordonnancement (Ghedjati & Portmann, 2009). En effet, si le choix des machines est multiple, l'affectation des opérations peut se faire par des règles de priorité statiques ou dynamiques. Concernant les premières règles, l'affectation est réalisée avant le processus de séquencement et reste donc invariante le long du processus. Quant aux règles dynamiques, elles utilisent une connaissance instantanée du système. Elles déterminent donc les priorités des opérations durant le processus d'ordonnancement. Dans cette approche, nous utilisons des heuristiques de construction statiques pour l'affectation des opérations sur

les machines. Ces heuristiques sont basées essentiellement sur des règles de priorité. Une affectation est caractérisée par un ensemble :

$$A = \left\{ A_{i,j,k} \in \{0,1\} \,\middle|\, 1 \le i \le E, 1 \le j \le n, 1 \le k \le m_i \right\}$$

tel que : $A_{i,j,k} = \begin{cases} 1 & \text{si } O_{ij} \text{ est affectée à } M_k \\ 0 & \text{sinon} \end{cases}$ et $\sum_{k=1}^{m_i} A_{i,j,k} = 1.$

Pour déterminer une affectation, nous appliquons des règles de priorité statiques qui permettent de trier les travaux sur le premier étage dans une liste *L*. Après avoir rangé les opérations, l'assignation de ces dernières est faite sur la première machine libre et éligible en utilisant la règle FAM (« first available machine »). Pour les étages restants, la règle « premier arrivé-premier servi » (FCFS pour « first come-first served ») est utilisée pour trier la liste des opérations par rapport à la séquence des opérations obtenue à l'étage précédent. Les opérations sont ensuite affectées à une machine en utilisant la règle FAM. Les différentes heuristiques utilisées pour cette phase reposent sur des règles de priorité et/ou des heuristiques de construction issues de la littérature pour lesquelles nous présentons dans ce qui suit un rappel sur leur principe de fonctionnement.

❖ ***Règles de priorité***

Les règles de priorité appliquées à l'affectation au premier étage sont les suivantes : SPT, LPT, LWKR, MWKR, SRM et LRM (Baker, 1974) (voir Tableau 3.1).

❖ ***Heuristiques de construction***

Comme leur nom l'indique, les heuristiques constructives construisent une solution de manière itérative en la complétant étape par étape tout en utilisant des règles simples pour trier la liste des travaux. Nous utilisons dans cette partie les heuristiques constructives suivantes : PAL, CDS, GUP, DAN et NEH. Elles ont été initialement proposées pour résoudre des problèmes d'ordonnancement de type flow-shop de permutation. Certaines de ces heuristiques utilisent l'algorithme de Johnson (Johnson, 1954) pour générer des solutions. Dans ce qui suit nous proposons un bref rappel sur leur principe de fonctionnement.

- ***Algorithme de Johnson.*** Pour résoudre le problème flow-shop classique à deux étages où le critère est la minimisation de la date d'achèvement de tous les travaux, Johnson (Johnson, 1954) a proposé un algorithme optimal qui consiste à :

- rechercher l'opération O_{ij}, où $(i=1,2)$ et $(j=1,\cdots,n)$, dont le temps d'exécution sur la machine i est le plus petit possible ;
- si $i=1$, l'opération est placée à la première place disponible en début de séquence de l'ordonnancement, et si $i=2$, cette opération est placée à la dernière position disponible ;
- supprimer l'opération O_{ij} de la liste des travaux non encore séquencés

Cette procédure est répétée jusqu'au placement de toutes les opérations.

Tableau 3.1 – Description des règles de priorité utilisées

Règles de priorité	Description
SPT	Le plus court temps opératoire (« Shortest Processing Time ») : $\min_j \left(p_{ij}\right)$
LPT	Le plus long temps opératoire (« Longest Processing Time ») : $\min_j \left(-p_{ij}\right)$
LWKR	Le plus court temps opératoire restant (« Least Work Remaining ») : $\min_j \left(\sum_{i=\ell}^{E} p_{ij}\right)$
MWKR	Le plus long temps opératoire restant (« Most Work Remaining ») : $\min_j \left(-\sum_{i=\ell}^{E} p_{ij}\right)$
SRM	Le plus court temps opératoire restant, excluant l'opération en cours d'exécution (« Shortest Remaining Work, excluding the operation under consideration ») : $\min_j \left(\sum_{i=\ell+1}^{E} p_{ij}\right)$
LRM	le plus long temps opératoire restant, excluant l'opération en cours d'exécution (« Longest Remaining Work, excluding the operation under consideration ») : $\min_j \left(-\sum_{i=\ell+1}^{E} p_{ij}\right)$

- ***Heuristique de Palmer (PAL).*** L'idée de base de l'heuristique proposée dans (Palmer, 1965) pour résoudre le flow-shop classique, est de donner une priorité aux opérations ayant des temps d'exécution croissants dans leurs gammes opératoires (Widmer, 1991). Ainsi, les travaux sont classés dans l'ordre croissant de l'indice de pente $S(j)$ calculé, pour chaque travail $j=1,\cdots,n,$ de la façon suivante :

$$S(j)=\sum_{i=1}^{E}\left(E-(2i-1)\right)\times p_{ij}$$

- ***Heuristique de Campbell, Dudek et Smith (CDS).*** L'heuristique proposée dans (Campbell et al., 1970) consiste à générer $(E-1)$ solutions en appliquant l'algorithme de Johnson sur deux machines fictives. La première regroupe les *k* premières machines et la deuxième regroupe les *k* dernières machines. Les durées d'exécution de chaque travail $(j=1,\cdots,n)$ sur ces deux machines fictives sont la somme des durées opératoires sur les machines qu'ils regroupent. Pour *k* variant entre 1 et $E-1,$ ces durées sont définies par :

$$a(k,j)=\sum_{i=1}^{k} p_{ij} \text{ et } b(k,j)=\sum_{i=E-k+1}^{E} p_{ij}$$

  La meilleure solution, ayant la plus petite date d'achèvement de tous les travaux sera ainsi retenue.

- ***Heuristique de Gupta (GUP).*** Comme pour la méthode de Palmer, Gupta (1971) propose un indice de pente pour ordonnancer les travaux. Ainsi, ces dernières sont classées par ordre croissant de leur indice de pente $G(j)$ calculés de la façon suivante :

$$G(j)=\frac{e(j)}{\min\limits_{1\le i\le E-1}\left(p_{ij}-p_{i+1,j}\right)} \quad \text{pour } j=1,\cdots,n$$

$$\text{avec } e(j)=\begin{cases} 1 & \text{si } \left(p_{1j}-p_{Ej}\right)\ge 0 \\ -1 & \text{sinon} \end{cases}$$

- ***Heuristique de Dannenbring (DAN).*** Dannenbring (1977) a tenté de combiner les avantages de l'heuristique PAL et de l'heuristique CDS. Cette méthode applique l'algorithme de Johnson sur deux machines fictives. Les durées opératoires sont déterminées en reflétant le comportement de l'indice de pente de Palmer. Le calcul des temps opératoires pour les deux machines est comme suit :

$$a(j)=\sum_{i=1}^{E}(E-i+1)\times p_{ij} \text{ et } b(j)=\sum_{i=1}^{E}(i\times p_{ij}) \quad \text{pour } j=1,\cdots,n.$$

- ***Heuristique de Nawaz, Enscore et Ham (NEH).*** L'heuristique de NEH (Nawaz et al., 1983) permet de résoudre des problèmes flow-shop à *m* machines en minimisant la plus grande date d'achèvement. Elle est basée sur l'hypothèse que le travail ayant une durée totale d'exécution élevée est prioritaire par rapport à un travail ayant un temps total d'exécution plus faible. Cette technique construit une solution par insertions successives des travaux, selon la priorité du travail, dans une séquence partielle. Elle commence par ordonner les deux travaux ayant les plus grands temps d'exécutions.

Ensuite, elle sélectionne le travail non ordonné ayant le plus grand temps d'exécution et l'insère entre les travaux déjà placés de manière à minimiser la plus grande date d'achèvement. L'étape d'insertion est répétée jusqu'à ce que tous les travaux soient ordonnés.

Pour adapter les heuristiques PAL, CDS, GUP, DAN au FSH, nous utilisons les mêmes processus d'adaptation proposées dans (Santos et al., 1996). Quant à l'heuristique de NEH, nous proposons de diviser la durée opératoire de chaque opération s'exécutant sur un étage *i* par le nombre de machine m_i sur cet étage pour rendre cette heuristique adaptable à ce même problème.

L'affectation des opérations sur les ressources étant fixée, le problème devient donc un problème flow-shop classique. La modélisation du problème sous forme d'un graphe de recherche, nous permet de le résoudre à l'aide de l'algorithme ACS. Pour cela, il est primordial de choisir un modèle pour la représentation des quantités de phéromones et de définir une heuristique d'information exprimant la visibilité. De plus, il faut tenir compte, durant le processus de construction d'une solution, des contraintes de précédence entre les opérations de chaque travail.

2.1.2.a Modélisation de la phéromone

La phéromone permet de mettre en place une mémoire adaptative décrivant l'état du système de recherche. Plusieurs modèles de phéromones ont été proposés dans la littérature (Blum & Sampels, 2002) pour résoudre divers problèmes d'ordonnancement. Ils sont essentiellement basés sur l'attribution d'une quantité de phéromones $\tau(O_{ij}, O_{hl})$ sur chaque paire d'opérations $O_{ij}, O_{hl} \in O$ (pour $1 \le i, h \le E$ et $1 \le j, l \le n$) comme dans le modèle proposé initialement par Colorni et al. (1994). Dans cette représentation, la quantité de phéromones $\tau(O_{ij}, O_{hl})$ influence le choix d'une fourmi traversant deux opérations séquencées, quelles que soient les machines auxquelles elles ont été affectées. D'autres modélisations de phéromones ont été présentées dans Blum et Sampels (2002). Dans leur modèle, nommé modèle de relation d'apprentissage (en anglais « relation-learning model »), des quantités de phéromones sont assignées aux paires d'opérations liées. Dans ce cas, deux opérations sont liées si elles sont traitées sur la même machine. Ainsi, la phéromone influence l'ordre relatif des opérations s'exécutant sur une même machine.

Le modèle de phéromones, que nous proposons, consiste à attribuer la phéromone sur chaque paire d'opérations, mais avec des quantités différentes (Khalouli et al., 2008a) selon les machines auxquelles elles sont assignées. En fait, dans notre représentation, les quantités de phéromones attribuées aux paires d'opérations liées sont plus importantes que celles des autres paires. Ainsi, une quantité initiale de phéromones τ_0 est attribuée à chaque arc reliant une paire d'opérations.

2.1.2.b Modélisation de la visibilité

La visibilité $\eta(O_{ij}, O_{hl})$ est une donnée propre au problème qui permet d'informer les fourmis artificielles sur la nature du problème. Elle est utilisée pour estimer l'opportunité de la transition d'une opération O_{ij} à une autre opération O_{hl}. Ainsi, pour guider les fourmis, nous introduisons une information liée à l'état d'avancement partiel de l'ordonnancement des opérations candidates. Pour influencer les probabilités de transition, de nombreuses règles de priorité ont été utilisées dans la littérature (Blum & Sampels, 2004; Ying & Lin, 2006; Ying & Lin, 2007). Ces dernières sont des politiques utilisées pour sélectionner une opération dans une liste d'opérations candidates à un placement. Il s'agit de méthodes, qui construisent des séquences d'opérations en se fondant sur des règles de priorité simples. Dans cette approche, nous utilisons des heuristiques basées sur les règles : SPT, LPT, LWKR, MWKR, SRM et LRM, vues précédemment. Pour pouvoir les appliquer nous les adaptons comme le montre le Tableau 3.2.

Tableau 3.2 – Heuristiques de visibilité

Règle de priorité	**Heuristique de visibilité associée**
SPT	$\left(p_{ij}\right)^{-1}$
LPT	p_{ij}
LWKR	$\left(\sum_{i=\ell}^{E} p_{ij}\right)^{-1}$
MWKR	$\sum_{i=\ell}^{E} p_{ij}$
SRM	$\left(\sum_{i=\ell+1}^{E} p_{ij}\right)^{-1}$
LRM	$\sum_{i=\ell+1}^{E} p_{ij}$

Pour mieux comprendre ces heuristiques, reprenons l'exemple de la règle SPT. Rappelons que

la définition de la règle SPT est d'ordonnancer les opérations dans l'ordre croissant de leurs durées opératoires. Ainsi, l'heuristique de visibilité associée à SPT permet d'attribuer une importance à l'opération ayant la durée opératoire la plus petite.

2.1.2.c Processus de construction d'une solution

Initialement, toutes les fourmis sont positionnées sur le nœud source O_D^* qui est le point de départ de nos ordonnancements. Pour trouver des solutions possibles au problème considéré, chaque fourmi *k* construit, indépendamment, une séquence d'opérations. Pour ce faire, elle doit choisir une opération parmi la liste des opérations « candidates » S_k (qui est l'ensemble des opérations successeurs restant non encore séquencées par la fourmi *k*). Ainsi, si cette fourmi est positionnée sur un sommet O_{ij} à un instant donné, elle choisit à partir de S_k, l'opération *y* en utilisant la règle de transition suivante (3.1) :

$$y = \begin{cases} \arg\max_{O_{hl} \in S_k} \left\{ \left[\tau(O_{ij}, O_{hl}) \right] \cdot \left[\eta(O_{ij}, O_{hl}) \right]^{\beta} \right\}, & \text{si} \quad q \le q_0 \\ Y, & \text{sinon} \end{cases} \tag{3.1}$$

Dans ce cas les mêmes paramètres β, q_0 et q de l'algorithme ACS (voir Chapitre 2) sont utilisés. Rappelons que β permet de moduler l'importance relative entre la quantité de phéromones et la visibilité. L'importance relative entre intensification et diversification est déterminée grâce à la constante paramétrable q_0 de valeur située comprise entre 0 et 1. Ainsi pour choisir une nouvelle opération, on attribue une valeur aléatoire uniformément distribuée entre 0 et 1 à la variable q, qui selon sa valeur par rapport à q_0 permet de se focaliser sur les meilleures solutions trouvées ou bien sur des solutions non encore explorées. En effet, si $q \le q_0$, le système tend vers une intensification en utilisant au mieux les informations recueillies par le système. Ainsi la sélection de la nouvelle opération se fonde sur les meilleures configurations obtenues et mémorisées au cours des recherches précédentes. Dans le cas contraire où $q > q_0$, le système tend à effectuer une diversification en sélectionnant aléatoirement l'opération *Y* selon la règle (3.2) :

$$\Pr_{O_{ij},Y} = \frac{\left[\tau(O_{ij}, Y) \right] . \left[\eta(O_{ij}, Y) \right]^{\beta}}{\sum_{O_{hl} \in S_k} \left[\tau(O_{ij}, O_{hl}) \right] . \left[\eta(O_{ij}, O_{hl}) \right]^{\beta}} \tag{3.2}$$

La procédure est répétée jusqu'à ce que toutes les opérations soient sélectionnées. Les opérations retenues sont successivement stockées dans une liste dite «liste tabou» notée L_k, qui permet de mémoriser les opérations déjà visitées par la fourmi *k*. Il est à noter que la liste tabou est différente de la méthode de recherche tabou. Son objectif est d'éviter qu'une opération ne soit visitée plus d'une fois.

2.1.2.d Mécanisme de mise à jour de la quantité de phéromones

Le mécanisme de mise à jour de la phéromone est utilisé pour simuler les changements de celle-ci au cours de la procédure de construction. En effet, cette quantité est augmentée en raison du dépôt de phéromones par les fourmis et diminuée en fonction de l'évaporation de la phéromone au cours du temps. Deux types de stratégies de mise à jour des quantités de phéromones sont ainsi proposés :

- ***Une mise à jour locale***

Afin de ne pas influencer le choix des autres fourmis, une règle de mise à jour locale est utilisée pour réduire la quantité de phéromones attribuée à une paire d'opérations (O_{ij}, O_{hl}) visitée par une fourmi *k*. Ainsi, l'opération O_{hl} sélectionnée par cette fourmi devient moins attrayante pour les autres fourmis. Cette règle est utilisée pour biaiser le tour des autres fourmis et éviter les optima locaux. En construisant une solution, chaque fourmi *k* modifie la quantité de phéromones à chaque transition comme le montre la relation (3.3) :

$$\tau(O_{ij}, O_{hl}) = (1-\rho_\ell).\tau(O_{ij}, O_{hl}) + \rho_\ell.\tau_0 \qquad (3.3)$$

telle que τ_0 est la valeur initiale de la trace de phéromones et ρ_ℓ ($0 \le \rho_\ell \le 1$) représente le coefficient d'évaporation de phéromones.

- ***Une mise à jour globale***

La quantité de phéromone est également mise à jour à la fin de chaque itération. Seule la meilleure solution générée est mise à jour en appliquant la relation (3.4), ce qui participe à une intensification par sélection de la meilleure solution.

$$\tau(O_{ij}, O_{hl}) = (1-\rho_g).\tau(O_{ij}, O_{hl}) + \rho_g.\Delta\tau(O_{ij}, O_{hl}) \qquad (3.4)$$

avec $\Delta\tau(O_{ij},O_{hl}) = \begin{cases} \left(C_{gb}\right)^{-1} & \text{si } (O_{ij},O_{hl}) \in \text{ la meilleure solution} \\ 0, & \text{sinon} \end{cases}$

$\rho_g \ \left(0 \le \rho_g \le 1\right)$ représente le coefficient d'évaporation de phéromones et C_{gb} la meilleure solution trouvée.

Algorithme 3.1 – Algorithme *HACS-FSH*

Affecter les opérations aux machines

Initialiser les traces de phéromone à τ_0 et les listes $S_k(i)$ et $L_k(i)$

Déterminer les différents paramètres de l'algorithme

Positionner une colonie de $nbAnts$ fourmis au sommet source O_D^*

Répéter

Pour chaque fourmi $k=1$ **à** $nbAnts$ **Faire**

Tant que $S_k \neq \emptyset$ **Faire**

Appliquer la règle de transition (3.1) pour sélectionner l'opération suivante $O_{hl} \in S_k$

Ranger l'opération dans la liste taboue $L_k \leftarrow L_k \cup O_{hl}$

Appliquer la mise à jour locale en utilisant la relation (3.3)

Fin Tant que (un tour complet est réalisé)

Évaluer les solutions obtenues

Fin pour (toutes les fourmis ont terminé leur ordonnancement)

Appliquer (avec une certaine probabilité) une recherche locale sur la meilleure solution obtenue

Appliquer la mise à jour globale en utilisant la relation (3.4)

Jusqu'à nombre maximal des itérations atteint ou solution optimale trouvée

2.2 *DESCRIPTION DE L'APPROCHE INTÉGRÉE IMOACS-FSH*

Nous présentons dans cette section, l'approche que nous avons développée pour le $FHE,\left(Pm_i\right)_{i=1}^{E}\left|\ \right|C_{\max}$ sans dissocier les décisions concernant la phase d'affectation et celle du séquencement. Dans cette optique de résolution nous parlons d'approches intégrées. L'idée de cette méthode s'inspire de la méthode proposée dans (Srinivasa Raghavan & Venkataramana, 2009) pour la résolution d'un système de production à machines parallèles. Notre approche

est décrite dans l'Algorithme 3.2. Ce travail a fait l'objet de la communication internationale (Khalouli et al., 2009).

2.2.1 Procédure de construction d'une solution

Le processus de construction d'une solution, pour cette approche, doit tenir compte simultanément de deux décisions : l'affectation et le séquencement. Ainsi, une fourmi *k* doit prendre, durant le processus de construction, deux décisions qui forment une composante regroupant la sélection d'une machine et la sélection d'une opération parmi la liste des opérations candidates S_k. D'autre part et afin de respecter les contraintes de précédence du problème à chaque étape de la construction, nous proposons une résolution du problème étage par étage, c'est-à-dire, qu'on commence par l'étage 1, puis l'étage 2, jusqu'à l'étage final *E*. Par conséquent, la procédure est répétée pour chaque étage *i*, jusqu'à ce que toutes les opérations soient sélectionnées et affectées à une machine éligible. Les opérations retenues, ainsi que les machines auxquelles elles sont assignées sont successivement stockées dans une liste tabou. Les étapes de la procédure sont les suivantes :

2.2.1.a La sélection d'une machine

Pour sélectionner une machine correspondant à un étage *i*, les fourmis utilisent la règle de transition (3.5).

$$m = \begin{cases} \arg\max_{m \in M_i} \{\eta_m\}, & \text{si} \quad q_m \le q_{m0} \\ M, & \text{sinon} \end{cases} \tag{3.5}$$

η_m représente une heuristique d'information qui indique la désirabilité de choisir la première machine éligible traduite par l'équation (3.6).

$$\eta_m = \frac{1}{t_m + 1} \tag{3.6}$$

où t_m représente la somme des dates d'achèvement de toutes les opérations déjà affectées à la machine *m*. Si aucune opération n'est encore placée alors η_m est supposée égale à 1. q_m est une variable aléatoire uniformément distribuée sur $[0,1]$ et q_{m0} est un paramètre compris entre 0 et 1. Selon la valeur de q_m, le choix d'une nouvelle machine se fait de deux manières différentes. La première favorise la machine la moins chargé, lorsque $q_m \le q_{m0}$. La deuxième,

à l'inverse, sélectionne de manière aléatoire une machine *M* selon la règle (3.7) :

$$\Pr(M) = \frac{\eta_M}{\sum_{y \in M_i} \eta_y}, \quad \text{si} \quad M \in M_i \tag{3.7}$$

2.2.1.b La sélection d'une opération

Une fois qu'une machine *m* est choisie, la fourmi *k* sélectionne une opération dans S_k. Le choix de l'opération dépend de la quantité de phéromone $\tau_m(x, p)$, indiquant la désirabilité de mettre l'opération *x* à la $p^{ième}$ position de la machine *m*, et une heuristique d'information η_x ayant pour but d'estimer la désirabilité du choix de l'opération *x*. La règle de transition utilisée pour la sélection d'une opération est la suivante :

$$x = \begin{cases} \arg\max_{s \in S_k} \left\{ \left[\tau_m(s, p) \right] \cdot \left[\eta_s \right]^\beta \right\}, & \text{si} \quad q \le q_0 \\ X, & \text{sinon} \end{cases} \tag{3.8}$$

Les paramètres β, q_0 et q sont repris de l'ACS standard. L'opération *X* est choisie aléatoirement selon la règle (3.9) :

$$\Pr(X) = \frac{\left[\tau_m(X, p) \right] \cdot \left[\eta_X \right]^\beta}{\sum_{r \in S_k} \left[\tau_m(r, p) \right] \cdot \left[\eta_r \right]^\beta} \tag{3.9}$$

2.2.1.c Modélisation de la phéromone

Le modèle de phéromones $\tau_m(x, p)$ proposé indique l'intérêt de placer l'opération *x* à la $p^{ième}$ position de toutes les machines éligibles au traitement de cette dernière. Ainsi, pour chaque machine *m*, nous définissons une matrice de phéromones qui permet d'attribuer une quantité de phéromones chaque fois qu'une opération est placée à la position *p* de la machine *m*. Ainsi, les quantités de phéromones influencent l'ordre relatif des opérations se traitant sur une même machine. Initialement, les quantités de phéromones sont fixées et valent τ_0.

2.2.1.d Modélisation de la visibilité

Comme pour la première méthode *HACS-FSH* proposée, des heuristiques de visibilité sont utilisées pour représenter des informations relatives au séquencement partiel des opérations.

Dans cette approche, nous utilisons une adaptation de certaines heuristiques constructives issues de la littérature en déterminant un indice de priorité pour ordonner chacune des opérations à placer comme proposé dans les articles de (Ying & Lin, 2006; Ying & Lin, 2007).

Pour cette méthode *IMOACS-FSH*, nous reprenons les heuristiques de visibilité basées sur les règles : SPT, LPT, LWKR, MWKR, SRM, LRM présentées pour la méthode *HACS-FSH* (voir Tableau 3.2). D'autres heuristiques de visibilité sont aussi considérées. Elles se basent sur les heuristiques PAL, CDS, GUP, DAN et NEH. Ainsi, nous définissons, pour la modélisation de ces visibilités, des indices de priorité RI pour les différents travaux à traiter. Ces indices se calculent pour une heuristique (H) donnée $\left(\mathrm{H} \in \{\text{PAL, CDS, GUP, DAN, NEH}\}\right)$, comme suit :

$$\mathrm{RI\_H}[j] = \frac{n - \mathrm{ordre\_H}[j] + 1}{n} \tag{3.10}$$

tel que $\mathrm{ordre\_H}[j]$ représente l'ordre (ou le rang) du travail j obtenu à partir de la solution donnée par cette heuristique H.

Par exemple, pour calcul de la visibilité pour l'heuristique NEH, nous attribuons un rang $\mathrm{ordre\_NEH}[j]$ à chaque travail à partir de la solution obtenue par NEH, puis nous déterminons l'indice de priorité de la manière suivante :

$$\mathrm{RI\_NEH}[j] = \frac{n - \mathrm{ordre\_NEH}[j] + 1}{n}$$

Comme nous procédons, étage par étage, pour la construction d'une solution au problème, nous utilisons les différentes visibilités pour le choix des opérations se traitant au premier étage. Quant aux $(E-1)$ étages restants, nous proposons une visibilité qui utilise l'heuristique FCFS (premier arrivé-premier servi). Cette dernière permet d'organiser la succession des opérations sur chaque étage en fonction de leurs temps opératoires et de leurs priorités. Ainsi, nous attribuons à chaque travail ayant achevé son traitement sur l'étage précédent une priorité d'être traité sur l'étage courant le plus tôt possible. Les travaux à placer à l'étage i sont ordonnés dans l'ordre croissant de leurs dates d'achèvement $C_{i-1,j}$ avec $1 \leq j \leq n$. Les indices de priorité sont déterminés de la même manière (voir relation (3.10)).

2.2.1 Mécanisme de mise à jour des quantités de phéromones

Les mêmes stratégies de mise à jour utilisées pour l'algorithme *HACS-FSH* sont considérées. La mise à jour locale est réalisée en appliquant la règle (3.11).

$$\tau_m(x, p) = (1-\rho_\ell)\,\tau_m(x, p) + \rho_\ell \cdot \tau_0 \qquad (3.11)$$

Lorsque toutes les fourmis terminent la construction de leur solution, une mise à jour globale est appliquée à la meilleure solution obtenue en utilisant la relation (3.12).

$$\tau_m(x, p) = (1-\rho_g)\,\tau_m(x, p) + \rho_g \cdot \Delta\tau_m(x, p) \qquad (3.12)$$

avec $\Delta\tau_m(x, p) = \begin{cases} \left(C_{gb}\right)^{-1}, & \text{si } (x, p) \in \text{ la meilleure solution} \\ 0, & \text{sinon} \end{cases}$

Algorithme 3.2 – Algorithme *IMOACS-FSH*

Initialiser les traces de phéromone à τ_0 et les listes $S_k(i)$ et $L_k(i)$

Déterminer les différents paramètres de l'algorithme

Répéter

Pour chaque fourmi $k = 1$ **à** *nbAnt* **Faire**

Pour chaque étage $i = 1$ **à** E **Faire**

Tant que $S_k \neq \emptyset$ **Faire**

Appliquer la règle de transition (3.5) pour sélectionner une machine

Appliquer la règle de transition (3.8) pour sélectionner l'opération suivante $O_{hl} \in S_k$

Ranger l'opération dans la liste tabou $L_k \leftarrow L_k \cup O_{hl}$

Appliquer la mise à jour locale en utilisant la relation (3.11)

Fin Tant que (un tour complet est réalisé)

Fin Pour

Evaluer les solutions obtenues

Fin Pour (toutes les fourmis ont terminé leur ordonnancement)

Appliquer (avec une certaine probabilité) une recherche locale sur la meilleure solution obtenue

Appliquer la mise à jour globale (3.12)

Jusqu'à nombre maximale des itérations atteint ou solution optimale trouvée

2.3 DESCRIPTION DE L'APPROCHE INTÉGRÉE IOMACS-FSH

Dans cette section, nous proposons une troisième approche pour solutionner le $FHE,(Pm_i)_{i=1}^{E}\,|\,|C_{\max}$. Nous utilisons le même principe que la deuxième approche sauf que dans ce cas le choix de la machine est effectué après le choix de l'opération. Cette approche *IOMACS-FSH* est décrite par l'Algorithme 3.3.

2.3.1 Procédure de construction d'une solution

Durant le processus de construction, chaque fourmi *k* choisit une opération à ordonnancer et lui attribue une machine pour son traitement de la manière suivante :

2.3.1.a La sélection d'une opération

Chaque fourmi *k* sélectionne, pour chaque étage *i*, une opération parmi la liste des opérations candidates S_k. La règle de transition utilisée pour la sélection d'une opération est la suivante :

$$x=\begin{cases}\arg\max\limits_{O_{hl}\in S_k}\left\{\left[\tau(O_{ij},O_{hl})\right]\cdot\left[\eta(O_{ij},O_{hl})\right]^{\beta}\right\}, & \text{si} \quad q\le q_0\\ X, & \text{sinon}\end{cases} \tag{3.13}$$

Puisque nous procédons étage par étage, les opérations O_{ij} et O_{hl} se trouvent donc sur le même étage ($i = h$). Les mêmes paramètres β, q_0 et q sont utilisés. L'opération *X* est définie selon la règle (3.14) :

$$\Pr_{O_{ij},X}=\frac{\left[\tau(O_{ij},X)\right].\left[\eta(O_{ij},X)\right]^{\beta}}{\sum\limits_{O_{hl}\in S_k}\left[\tau(O_{ij},O_{hl})\right].\left[\eta(O_{ij},O_{hl})\right]^{\beta}} \tag{3.14}$$

Les deux équations (3.13) et (3.14) sont établies en fonction de la quantité de phéromones τ et de la visibilité η qui joue le rôle d'une règle heuristique favorisant le choix d'une opération grâce à une vue partielle de l'ordonnancement du problème considéré. Les modèles représentant ces deux facteurs sont comme suit :

❖ ***Modélisation de la phéromone***

Le modèle de phéromones que nous proposons permet d'exprimer la désirabilité de placer une opération *y* après l'opération O_{ij}. Le choix de cette représentation a pour objectif d'influencer

le choix des séquencements des opérations se traitant sur un même étage.

❖ ***Modélisation de la visibilité***

Pour influencer les probabilités de transition, les visibilités proposées pour l'approche *IMOACS-FSH* (voir §2.2.1.d) sont reprises. Ceci permet de fournir une information liée à l'état d'avancement partiel de l'ordonnancement des opérations candidates.

2.3.1.b La sélection d'une machine

Pour chaque opération sélectionnée, nous lui attribuons une machine parmi la liste des machines éligibles pour son traitement. Ainsi, nous proposons d'affecter chaque opération sélectionnée à la machine libre le plus tôt. La date minimale pour chaque machine éligible est calculée au fur et à mesure de la construction de la solution. Le choix des machines varie donc en fonction des scénarios des solutions du processus de recherche.

La procédure de construction est répétée jusqu'à ce que toutes les opérations soient sélectionnées. Les solutions retenues sont successivement stockées dans une liste tabou L_k.

2.3.2 Mécanisme de mise à jour de phéromones

Comme pour les deux approches précédentes, nous utilisons les mêmes stratégies de mise à jour. En construisant une solution, chaque fourmi *k* modifie la quantité de phéromones à chaque transition en utilisant la règle (3.15) réalisant ainsi une mise à jour locale.

$$\tau(O_{ij}, O_{hl}) = (1-\rho_\ell).\tau(O_{ij}, O_{hl}) + \rho_\ell.\tau_0 \qquad (3.15)$$

De même, une mise à jour globale est effectuée à la fin de chaque cycle. Par conséquent, nous appliquons la relation (3.16) à la meilleure solution obtenue, ce qui participe à son intensification.

$$\tau(O_{ij}, O_{hl}) = (1-\rho_g).\tau(O_{ij}, O_{hl}) + \rho_g.\Delta\tau(O_{ij}, O_{hl}) \qquad (3.16)$$

avec $\Delta\tau(O_{ij}, O_{hl}) = \begin{cases} \left(C_{gb}\right)^{-1} & \text{si } (O_{ij}, O_{hl}) \in \text{ la meilleure solution} \\ 0, & \text{sinon} \end{cases}$

Algorithme 3.3 – Algorithme *IOMACS-FSH*

Initialiser les traces de phéromone à τ_0 et les listes $S_k(i)$ et $L_k(i)$

Déterminer les différents paramètres de l'algorithme

Répéter

Pour chaque fourmi $k = 1$ à $nbAnt$ **Faire**

Pour chaque étage $i = 1$ à E **Faire**

Tant que $S_k \neq \emptyset$ **Faire**

Sélectionner une opération $O_{hl} \in S_k$ en Appliquant la règle de transition (3.13)

Ranger l'opération dans la liste taboue $L_k \leftarrow L_k \cup O_{hl}$

Affecter l'opération sélectionnée à une machine éligible

Appliquer la mise à jour locale en utilisant la relation (3.15)

Fin Tant que (un tour complet est réalisé)

Fin Pour

Évaluer les solutions obtenues

Fin Pour (toutes les fourmis ont terminé leur ordonnancement)

Appliquer (avec une certaine probabilité) une recherche locale sur la meilleure solution obtenue

Appliquer la mise à jour globale

Jusqu'à nombre maximale des itérations atteint ou solution optimale trouvée

2.4 *RECHERCHES LOCALES*

Concernant l'algorithme *HACS-FSH*, rappelons que nous avons traité séparément l'affectation et le séquencement pour résoudre le problème d'ordonnancement FSH. Le fait de fixer les affectations ne permet pas de garantir des solutions pertinentes pour le problème pris dans son ensemble. L'ajout de recherche locale permet donc de remédier à cet inconvénient et essayer de changer les affectations au bout de quelques itérations pour diversifier l'espace de recherche des solutions. Ainsi, lorsqu'une recherche locale est appliquée, la nouvelle affectation (si elle existe) des opérations sur les machines est utilisée pour les prochaines itérations. Pour les méthodes *IMOACS-FSH* et *IOMACS-FSH*, l'ajout d'une recherche locale a permis d'améliorer le processus de recherche et par conséquent le temps d'exécution de nos algorithmes. En effet, d'après Dorigo et Gambardella (1997), l'addition d'une recherche locale aux algorithmes de fourmis permet d'améliorer leurs performances.

Pour apporter des améliorations à la solution obtenue, trois stratégies de recherche locale sont ajoutées à nos approches.

- La première stratégie utilise la structure de voisinages par blocs proposée par Nowicki et Smutnicki (1998) pour la résolution du FSH. Il s'agit d'une technique qui permet de réaffecter une ou plusieurs opérations sur d'autres machines éligibles et/ou de permuter leurs positions sur les machines auxquelles elles sont assignées.
- La seconde repose sur l'heuristique de NEH (Nawaz et al., 1983). Pour améliorer la meilleure solution obtenue, nous proposons une heuristique se basant sur le même principe que NEH (nous nommons cette recherche locale *« RL_ANEH »*). Ainsi, cette méthode commence par ordonner les deux premiers travaux de la solution obtenue. Ensuite, elle ajoute progressivement, les autres travaux, en les insérant entre ceux déjà placés jusqu'au placement de tous les travaux, de manière à minimiser la plus grande date d'achèvement sur le dernier étage.
- La troisième stratégie permet de générer (*n*-1) solutions en permutant les paires d'opérations adjacentes des travaux s'exécutant aux premiers étages de la solution obtenue. Nous évaluons par la suite ces permutations et retenons la meilleure. Nous appelons cette méthode « *RL_SWAP* ».

Après quelques tests préliminaires, nous proposons d'appliquer les deux dernières recherches locales à nos trois méthodes avec une probabilité $1/2$ de la manière suivantes :

Générer un nombre aléatoire Proba, compris entre 0 et 1

Si $\text{Proba} > 1/2$ **alors**

Appliquer *RL_ANEH*

Sinon

Appliquer *RL_SWAP*

Evaluer la nouvelle solution obtenue

Si la nouvelle solution obtenue est meilleure **alors** mettre à jour la valeur du $C_{\max}$ et sauvegarder cette nouvelle solution.

Dans nos algorithmes, les recherches locales sont appliquées avec une certaines probabilité, notée P_{RL}.

3 EXPÉRIMENTATIONS ET RÉSULTATS

Dans cette section, nous présentons l'ensemble des tests expérimentaux que nous avons réalisés pour l'évaluation et la validation des méthodes proposées précédemment pour la résolution du problème d'ordonnancement flow-shop hybride. Les expérimentations sont mises en œuvre sur un PC équipé d'un Intel (R) Core (TM) 2 DUO CPU condensé à 2.27 GHz et ayant 3 Go de mémoire vive (RAM). Les différents algorithmes sont codés en JAVA. Afin d'évaluer correctement nos méthodes, nous les avons testés sur des instances issues de la littérature.

3.1 *DESCRIPTION DES INSTANCES*

Nos expérimentations ont été effectuées sur un jeu de 77 instances proposées par Néron et al. (2001) pour le problème d'atelier FSH et dont la taille est de l'ordre de :

- $10 \text{travaux} \times 5 \text{étages}$;
- $10 \text{travaux} \times 10 \text{étages}$;
- $15 \text{travaux} \times 5 \text{étages}$;
- $15 \text{travaux} \times 10 \text{étages}$.

Le nombre de machines par étage varie entre 1 et 3 machine(s). Selon leurs caractéristiques, les instances peuvent être classées en 13 groupes. Chaque instance est représentée par les lettres : (*j*) indiquant le nombre de travaux, (*s*) le nombre d'étages et (*a*, *b*, *c* ou *d*) correspondant à la configuration des machines sur les étages. Par exemple l'instance $j10s5d2$ indique que l'on a 10 travaux, 5 étages et *d* comme configuration de machine. Une configuration de machine est une liste de nombres représentant le nombre de machines dans chacun des étages. Quatre structures sont ainsi proposées :

- La structure *a* est composée d'une machine à l'étage milieu et de trois machines sur les autres étages ;
- La structure *b* est composée d'une machine au premier étage et de trois machines sur les autres étages ;
- La structure *c* est composée de deux machines à l'étage milieu et de trois machines sur les autres étages ;
- La structure *d* est composée de trois machines sur tous les étages ;

Les structures de machine $d1$ et $d2$ à titre d'exemple, se différencient par les durées opératoires de leurs opérations. Ces durées ont été générées aléatoirement selon une loi de distribution discrète uniforme comprise entre 3 et 25. Les instances ayant les structures a, b et $j10s10c$ sont identifiées comme étant des problèmes faciles à résoudre (53 problèmes). Les autres instances (les 24 problèmes restants) sont classées comme étant des problèmes difficiles (Néron et al., 2001).

Ces jeux d'instances ont été résolus en utilisant une procédure par séparation et évaluation (B&B) (Néron et al., 2001). Des bornes inférieures (*LB*) sont ainsi calculées et le temps maximum alloué à la recherche est de 1600 secondes (*s*). Plusieurs instances n'ont pas été résolues de manière optimale au bout de ces 1600 *s* et restent donc ouverts. Ces instances ont aussi été utilisées pour tester l'efficacité d'autres méthodes de résolution pour le FSH et parmi lesquelles nous citons :

- l'approche AIS basée sur un système immunisé artificiel de Engin et Döyen (2004) ;
- l'algorithme génétique proposé par Besbes et al. (2006) ;
- l'algorithme de fourmis (noté « *Improved-AS* ») de Alaykýran et al.(2007).

3.2 *EXPÉRIMENTATIONS PRÉLIMINAIRES*

Des expérimentations préliminaires ont été réalisées sur quelques instances pour trouver les meilleurs paramètres pour nos algorithmes. Ces paramètres incluent le nombre de fourmis $nbAnt$, le nombre d'itérations $t_{\max}$, la quantité de phéromone initiale τ_0, les paramètres β, q_0 et q_{m0}, ainsi que les coefficients d'évaporation de phéromones locale ρ_ℓ et globale ρ_g. D'après la littérature, les coefficients d'évaporation de phéromones sont généralement choisis égaux. De même, nous supposons dans ce travail que ρ_ℓ est égal à ρ_g.

Comme dans (Alaykýran et al., 2007), nous utilisons les instances $j15s5a2$, $j15s5b1$, $j15s5c5$, et $j15s5d1$ pour déterminer les meilleurs paramètres. Quant au nombre de fourmis, nous le supposons égal à 5. Pour ces expérimentations préliminaires, nous utilisons les plages de valeurs suivantes :

- $\beta = \{0,1,2,3,4,5,6,7,8,9,10\}$
- $q_0, q_{m0} = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$
- $\rho_\ell, \rho_g = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$

- le nombre d'itérations pour ces tests préliminaires est fixé à 100. Pour chaque combinaison des paramètres, 25 essais sont réalisés.

Tableau 3.3 – Meilleurs paramètres pour nos algorithmes

		Meilleurs Paramètres		
Paramètres	**Plages des valeurs**	***HACS-FSH***	***IMOACS-FSH***	***IOMACS-FSH***
$nbAnt$	−−	5	5	5
$t_{\max}$	−−	2000	2000	2000
τ_0	−−	$(n \times LB)^{-1}$	$(n \times LB)^{-1}$	$(n \times LB)^{-1}$
β	$0 \le \beta \le 10$	2	2	2
q_0	$0 \le q_0 \le 1$	0.8	$\log(t)/\log(t_{\max})$	$\log(t)/\log(t_{\max})$
q_{m0}	$0 \le q_{m0} \le 1$	−−	$\frac{t}{t_{\max}}$	−−
ρ_ℓ	$0 \le \rho_\ell \le 1$	0.5	0.1	0.5
ρ_g	$0 \le \rho_g \le 1$	0.5	0.1	0.5
P_{RL}	$0 \le P_{RL} \le 1$	0.25	1	1

Comme dans (T'kindt et al., 2002), la valeur de q_0 a été choisie variable durant le processus de recherche pour les méthodes *IMOACS-FSH* et *IOMACS-FSH*. En effet, comme aucune meilleure solution n'est encore explorée, ce choix de q_0 permet de favoriser l'exploration de l'espace de recherche au départ. Ainsi, la sélection des solutions a tendance à avantager celles obtenues de manière aléatoire. En augmentant le nombre d'itérations, la valeur de q_0 augmente permettant ainsi d'intensifier progressivement la recherche vers les meilleures solutions déjà obtenues. Pour l'algorithme *IMOACS-FSH*, nous déterminons valeur de q_{m0} de la même manière. Le but est de choisir une machine parmi la liste des machines éligibles de façon aléatoire au début de l'algorithme, vu que toutes les machines sont libres initialement. Puis, au fur et à mesure de l'augmentation du nombre d'itérations, le choix de la machine qui est libre le plus tôt possible est avantagé.

Afin de déterminer les valeurs les plus favorables, la solution moyenne des 25 réplications, la meilleure solution ainsi que la pire ont été évaluées pour déterminer les meilleurs paramètres. Les résultats sont donnés dans le Tableau 3.3.

En outre, la probabilité d'appliquer une recherche locale pour les deux algorithmes *IMOACS-FSH* et *IOMACS-FSH* vaut 1 c'est-à-dire qu'à chaque itération la recherche locale est appliquée pour améliorer la meilleure solution obtenue. Concernant l'algorithme *HACS-FSH*, l'application d'une recherche locale dépend d'une variable $0 \leq pr \leq 1$ tirée aléatoirement. Si la valeur de pr est supérieure à P_{RL} alors la recherche locale est réalisée.

- **Quelques remarques :**

D'après les résultats obtenus, nous avons remarqué pour les instances considérées que :

- La valeur de β a une influence sur le temps d'exécution. Ainsi, plus la valeur de β augmente, plus le temps d'exécution augmente.
- Lorsque les valeurs de q_0, ρ_ℓ et ρ_g sont grandes, les solutions sont généralement médiocres.

3.3 *COMPARAISON DES MÉTHODES PROPOSÉES*

Dans cette section, nous étudions les performances des méthodes proposées pour résoudre le FSH avec pour critère d'optimisation le makespan. Cette partie expérimentale est organisée en quatre sous-parties. La première concerne les expérimentations réalisées pour l'évaluation de l'algorithme *HACS-FSH* en testant différentes variantes de cette méthode avec une multitude d'heuristiques pour la visibilité ainsi que pour les affectations. La seconde partie, concerne l'étude des résultats obtenus par l'approche *IMOACS-FSH*. La troisième partie, s'intéresse à la validation de la méthode *IOMACS-FSH*. Enfin, nous terminerons par une comparaison entre ces différentes approches pour déterminer la ou les plus performantes.

Pour chaque instance nous effectuons 25 essais. Les valeurs moyennes, minimales et maximales du $C_{\max}$ obtenues expérimentalement sont relevées pour chaque algorithme. Nous utilisons les notations suivantes :

- $BestC_{\max}$: valeur de la meilleure solution obtenue ;
- $MoyC_{\max}$: valeur moyenne des solutions obtenues ;

- $WorstC_{\max}$: valeur de la plus mauvaise solution ;
- CM : valeur moyenne des solutions obtenues par l'emploi d'une heuristique de visibilité donnée ;
- CPU_{Min} : valeur minimale du temps d'exécution de l'algorithme en secondes (s) ;
- CPU_{Moy} : valeur moyenne du temps d'exécution de l'algorithme en secondes (s).

Pour mesurer les performances de nos algorithmes, nous avons calculé les ratios de la déviation relative, que nous appelons « $\%Erreur$ », entre les meilleures solutions obtenues et les valeurs des bornes inférieures (Néron et al., 2001) à l'aide de la formule suivante :

$$\%Erreur = \frac{BestC_{\max} - LB}{LB} \times 100 \qquad (3.17)$$

Notons que le nombre d'itérations est fixé à $t_{max} = 2000$. Dans le but d'améliorer la solution de certaines instances difficiles, notons que le nombre d'itérations peut être augmenté de $t_{max} = 5000$.

3.3.1 Résultats de l'approche *HACS-FSH*

Dans ce paragraphe, nous présentons les résultats donnés par *HACS-FSH*. Les valeurs moyennes, minimales et maximales des solutions obtenues sont reportées dans le Tableau 3.4.

Pour cet algorithme 81.82% des problèmes considérés ont été résolus :

- 47 sur les 53 problèmes faciles ont atteint le *LB* ;
- sur les 24 problèmes difficiles seules 8 n'ont pas atteint le *LB* ;
- l'erreur moyenne est de 1.81% sur l'ensemble des instances, de 0.99% pour les instances faciles et de 3.6% pour les instances difficiles.

Tableau 3.4 – Résultats obtenus par l'application de la méthode HACS-FSH

Problème	***LB***	***Solutions obtenues***					
		$BestC_{max}$	$MoyC_{max}$	$WorstC_{max}$	CPU_{Min}	CPU_{Moy}	***% Erreur***
j10s5a2	88	88	88	88	0.001	0.022	0
j10s5a3	117	117	117	117	0.001	0.024	0
j10s5a4	121	121	121	121	0.001	0.042	0
j10s5a5	122	122	122	122	0.001	0.191	0
j10s5a6	110	110	110	110	0.001	0.693	0
j10s5b1	130	130	130	130	0.001	0.006	0
j10s5b2	107	107	107	107	0.001	0.026	0
j10s5b3	109	109	109	109	0.001	0.043	0
j10s5b4	122	122	122	122	0.001	0.031	0
j10s5b5	153	153	153	153	0.000	0.002	0
j10s5b6	115	115	115	115	0.001	0.019	0
j10s5c1	**68**	**68**	**70.56**	**72**	**0.102**	**1.700**	**0**
j10s5c2	**74**	**74**	**74.84**	**75**	**0.793**	**1.779**	**0**
j10s5c3	**71**	**72**	**72.56**	**73**	**1.701**	**1.756**	**1.41**
j10s5c4	**66**	**66**	**68.2**	**69**	**0.716**	**1.802**	**0**
j10s5c5	**78**	**78**	**79.6**	**81**	**0.086**	**1.745**	**0**
j10s5c6	**69**	**69**	**70.2**	**71**	**0.482**	**1.727**	**0**
j10s5d1	**66**	**66**	**67.16**	**68**	**0.026**	**1.482**	**0**
j10s5d2	**73**	**73**	**74.76**	**76**	**0.795**	**1.766**	**0**
j10s5d3	**64**	**64**	**64.84**	**65**	**0.054**	**1.535**	**0**
j10s5d4	**70**	**70**	**71.24**	**72**	**0.037**	**1.649**	**0**
j10s5d5	**66**	**66**	**68.76**	**70**	**1.367**	**1.75**	**0**
j10s5d6	**62**	**62**	**63.04**	**65**	**0.014**	**1.335**	**0**
j10s10a1	139	139	139	139	0.005	1.752	0
j10s10a2	158	158	158.92	160	0.006	7.942	0
j10s10a3	148	148	148	148	0.000	2.158	0
j10s10a4	149	149	149	149	0.005	0.590	0
j10s10a5	148	148	148	148	0.005	0.882	0
j10s10a6	146	146	147.28	148	0.006	8.315	0
j10s10b1	163	163	163	163	0.000	0.12	0
j10s10b2	157	157	157	157	0.007	0.510	0
j10s10b3	169	169	169	169	0.000	0.155	0
j10s10b4	159	159	159	159	0.000	0.135	0
j10s10b5	165	165	165	165	0.005	0.234	0
j10s10b6	165	165	165	165	0.005	0.207	0
j10s10c1	113	115	116.08	117	9.786	9.824	1.77
j10s10c2	116	119	119.92	121	9.792	9.837	2.59
j10s10c3	98	116	117.76	119	9.814	9.858	18.37
j10s10c4	103	120	120.88	121	9.797	9.833	16.5
j10s10c5	121	126	128.04	129	9.903	9.975	4.13
j10s10c6	97	106	106.88	108	10.24	10.438	9.28
j15s5a1	178	178	178	178	0.002	0.099	0
j15s5a2	165	165	165	165	0.002	0.049	0
j15s5a3	130	130	130	130	0.002	0.089	0
j15s5a4	156	156	156	156	0.002	0.269	0
j15s5a5	164	164	164	164	0.002	0.075	0
j15s5a6	178	178	178	178	0.002	0.077	0
j15s5b1	170	170	170	170	0.002	0.047	0
j15s5b2	152	152	152	152	0.002	0.087	0
j15s5b3	157	157	157	157	0.002	0.084	0
j15s5b4	147	147	147	147	0.002	177	0
j15s5b5	166	166	166	166	0.002	0.119	0
j15s5b6	175	175	175	175	0.002	0.072	0
j15s5c1	**85**	**85**	**86.6**	**87**	**5.84 (D)**	**12.223**	**0**
j15s5c2	**90**	**90**	**92.6**	**93**	**6.063(D)**	**14.747**	**0**
j15s5c3	**87**	**87**	**88.48**	**89**	**0.175(D)**	**14.900**	**0**
j15s5c4	**89**	**90**	**92**	**93**	**4.894**	**4.941**	**1.12**
j15s5c5	**73**	**77**	**78.8**	**80**	**4.796**	**4.863**	**5.48**
j15s5c6	**91**	**91**	**92.28**	**93**	**0.864**	**4.617**	**0**
j15s5d1	**167**	**167**	**167**	**167**	**0.002**	**0.013**	**0**
j15s5d2	**82**	**85**	**87.12**	**88**	**4.888**	**4.972**	**3.66**
j15s5d3	**77**	**83**	**84.44**	**86**	**7.101**	**7.615**	**7.79**
j15s5d4	**61**	**86**	**88**	**89**	**4.842**	**4.928**	**40.98**
j15s5d5	**67**	**81**	**82.32**	**83**	**4.805**	**4.893**	**20.90**

Suite – Tableau 3.4

Problème	*LB*	*Solutions obtenues*					
		$BestC_{max}$	$MoyC_{max}$	$WorstC_{max}$	CPU_{Min}	CPU_{Moy}	*%Erreur*
j15s5d6	**79**	**83**	**84.56**	**85**	**5.886**	**6.429**	**5.06**
j15s10a1	236	236	236	236	0.015	0.336	0
j15s10a2	200	200	200	200	0.016	0.270	0
j15s10a3	198	198	198	198	0.010	0.563	0
j15s10a4	225	225	225	225	0.010	1.254	0
j15s10a5	182	182	182	182	0.016	0.679	0
j15s10a6	200	200	200	200	0.010	0.818	0
j15s10b1	222	222	222	222	0.015	0.218	0
j15s10b2	187	187	187	187	0.015	0.114	0
j15s10b3	222	222	222	222	0.015	0.078	0
j15s10b4	221	221	221	221	0.015	0.099	0
j15s10b5	200	200	200	200	0.015	0.333	0
j15s10b6	219	219	219	219	0.015	0.144	0
Moyenne	**131.6**	**133.05**	**133.66**	**134.012**	**1.51**	**4.83**	**1.81**
				% des problèmes résolus			**81.82**

Les instances en gras représentent les problèmes difficiles
(D) : indique que $t_{max} = 5000$

Tableau 3.5 – Comparaison des heuristiques de visibilité utilisées pour *HACS-FSH*

	CM	**%Erreur moyenne**	**%Problèmes résolus**	
SPT	133.34	2.07	72.72	(44/12)**
LPT	134.18	3.20	59.74	(44/2)
LWKR	133.53	2.35	63.63	(43/6)
MWKR	133.23	2.04	72.72	(47/9)
SRM	133.78	2.65	57.14	(42/2)
LRM	133.14	1.92	76.62	(47/12)

** indique, pour cette colonne, que les deux chiffres entre parenthèses représentent respectivement le nombre de problèmes faciles et difficiles qui atteignent le *LB*

Au niveau de la comparaison entre les heuristiques de visibilité, les statistiques que nous avons réalisées sur l'ensemble des instances montrent que les meilleurs résultats sont obtenus en utilisant la règle LRM avec une déviation moyenne de 1.92% par rapport au *LB* et un pourcentage de résolution de 76.62%, puis MWKR et SPT avec 72.72% de problèmes résolus comme décrit dans le Tableau 3.5. Les visibilités utilisant les heuristiques SRM et LPT sont les moins bonnes. Ces résultats prouvent l'impact de la visibilité sur la qualité des solutions.

En ce qui concerne le temps de calcul, nous constatons que les solutions sont obtenues rapidement excepté les instances difficiles (qui n'ont pas atteint le *LB*), ainsi que les instances

j10s10c pour lesquelles une dizaine de secondes est indispensable en moyenne. Ces temps tiennent compte de l'utilisation des recherches locales durant le processus de recherche.

3.3.2 Résultats de la méthode *IMOACS-FSH*

Les résultats relatifs à l'algorithme *IMOACS-FSH* sont présentés dans le Tableau 3.6. On peut y voir que :

- 47 problèmes faciles atteignent le LB sur les 53 ;
- sur les 24 problèmes difficiles, 17 atteignent le *LB* ;
- l'erreur moyenne est de 1.66% sur l'ensemble des instances.

Tableau 3.6 – Résultats obtenues par l'application de la méthode *IMOACS-FSH*

Problème	***LB***	***Solutions obtenues***					
		$BestC_{max}$	$MoyC_{max}$	$WorstC_{max}$	CPU_{Min}	CPU_{Moy}	***% Erreur***
j10s5a2	88	88	88	88	0	0.002	0
j10s5a3	117	117	117	117	0.001	0.005	0
j10s5a4	121	121	121	121	0.001	0.004	0
j10s5a5	122	122	122	122	0.001	0.027	0
j10s5a6	110	110	110	110	0.007	0.037	0
j10s5b1	130	130	130	130	0.001	0.007	0
j10s5b2	107	107	107	107	0.001	0.002	0
j10s5b3	109	109	109	109	0.001	0.008	0
j10s5b4	122	122	122	122	0.001	0.005	0
j10s5b5	153	153	153	153	0.001	0.002	0
j10s5b6	115	115	115	115	0.001	0.002	0
j10s5c1	**68**	**68**	**68.44**	**69**	**0.001**	**0.848**	**0**
j10s5c2	**74**	**74**	**74.8**	**75**	**0.006**	**1.446**	**0**
j10s5c3	**71**	**72**	**72**	**72**	**1.505**	**1.563**	**1.41**
j10s5c4	**66**	**66**	**66.08**	**67**	**0.016**	**0.423**	**0**
j10s5c5	**78**	**78**	**78**	**78**	**0.007**	**0.039**	**0**
j10s5c6	**69**	**69**	**69.24**	**70**	**0.014**	**0.701**	**0**
j10s5d1	**66**	**66**	**66**	**66**	**0.002**	**0.066**	**0**
j10s5d2	**73**	**73**	**73.92**	**75**	**0.036**	**1.410**	**0**
j10s5d3	**64**	**64**	**64.24**	**65**	**0.009**	**0.721**	**0**
j10s5d4	**70**	**70**	**70**	**70**	**0.001**	**0.068**	**0**
j10s5d5	**66**	**66**	**66.96**	**67**	**0.039**	**1.549**	**0**
j10s5d6	**62**	**62**	**62**	**62**	**0.001**	**0.023**	**0**
j10s10a1	139	139	139	139	0.003	0.072	0
j10s10a2	158	158	158	158	0.019	1.119	0
j10s10a3	148	148	148	148	0.003	0.02	0
j10s10a4	149	149	149	149	0.003	0.039	0
j10s10a5	148	148	148	148	0.007	0.055	0
j10s10a6	146	146	146	146	0.008	0.235	0
j10s10b1	163	163	163	163	0.002	0.005	0
j10s10b2	157	157	157	157	0.004	0.059	0
j10s10b3	169	169	169	169	0.002	0.030	0
j10s10b4	159	159	159	159	0.002	0.009	0
j10s10b5	165	165	165	165	0.002	0.031	0
j10s10b6	165	165	165	165	0.003	0.013	0
j10s10c1	113	115	115.04	116	4.156	4.394	1.77

Suite – Tableau 3.6

Problème	*LB*	*Solutions obtenues*					
		$BestC_{max}$	$MoyC_{max}$	$WorstC_{max}$	CPU_{Min}	CPU_{Moy}	*% Erreur*
j10s10c2	116	119	119	119	4.063	4.138	2.59
j10s10c3	98	116	117.08	118	4.115	4.238	18.37
j10s10c4	103	120	120.08	122	5.274	5.271	16.5
j10s10c5	121	126	126.8	127	4.976	5.203	4.13
j10s10c6	97	106	106.36	107	6.610	6.807	9.28
j15s5a1	178	178	178	178	0.004	0.011	0
j15s5a2	165	165	165	165	0.004	0.006	0
j15s5a3	130	130	130	130	0.001	0.013	0
j15s5a4	156	156	156	156	0.004	0.009	0
j15s5a5	164	164	164	164	0.005	0.022	0
j15s5a6	178	178	178	178	0.004	0.01	0
j15s5b1	170	170	170	170	0.004	0.006	0
j15s5b2	152	152	152	152	0.004	0.005	0
j15s5b3	157	157	157	157	0.004	0.008	0
j15s5b4	147	147	147	147	0.001	0.004	0
j15s5b5	166	166	166	166	0.004	0.046	0
j15s5b6	175	175	175	175	0.004	0.012	0
j15s5c1	**85**	**85**	**86**	**87**	**0.753**	**4.454**	**0**
j15s5c2	**90**	**90**	**91.6**	**93**	**3.209**	**4.372**	**0**
j15s5c3	**87**	**87**	**87.52**	**88**	**0.255**	**2.688**	**0**
j15s5c4	**89**	**89**	**90.32**	**91**	**4.867**	**5.154**	**0**
j15s5c5	**73**	**74**	**75.36**	**76**	**6.561 (D)**	**6.724**	**1.37**
j15s5c6	**91**	**91**	**91**	**91**	**0.023**	**0.448**	**0**
j15s5d1	**167**	**167**	**167**	**167**	**0.004**	**0.016**	**0**
j15s5d2	**82**	**85**	**85.12**	**86**	**3.292**	**3.420**	**3.66**
j15s5d3	**77**	**82**	**83.4**	**84**	**3.420**	**3.606**	**6.49**
j15s5d4	**61**	**85**	**86.2**	**87**	**10.958 (D)**	**11.153**	**39.34**
j15s5d5	**67**	**80**	**80.68**	**81**	**3.120**	**3.227**	**19.4**
j15s5d6	**79**	**82**	**82.92**	**83**	**4.113**	**4.281**	**3.8**
j15s10a1	236	236	236	236	0.008	0.017	0
j15s10a2	200	200	200	200	0.038	0.232	0
j15s10a3	198	198	198	198	0.010	0.030	0
j15s10a4	225	225	225	225	0.010	0.172	0
j15s10a5	182	182	182	183	0.013	0.136	0
j15s10a6	200	200	200	200	0.010	0.051	0
j15s10b1	222	222	222	222	0.010	0.065	0
j15s10b2	187	187	187	187	0.009	0.023	0
j15s10b3	222	222	222	222	0.010	0.015	0
j15s10b4	221	221	221	221	0.010	0.092	0
j15s10b5	200	200	200	200	0.016	0.074	0
j15s10b6	219	219	219	219	0.011	0.060	0
Moyenne	**131.60**	**132.95**	**133.16**	**133.38**	**0.86**	**1.19**	**1.66**
				% des problèmes résolus			**83.12**

Les instances en gras représentent les problèmes difficiles
(D) : indique que $t_{max} = 5000$

Quant aux heuristiques de visibilité, nous pouvons conclure d'après le Tableau 3.7 que celles utilisant l'heuristique DAN donnent les meilleurs résultats avec un pourcentage de résolution de 75.33% et une déviation par rapport au *LB* de 1.85%.

Les solutions sont obtenues rapidement notamment pour les instances faciles. Pour certaines instances une amélioration a été enregistrée en augmentant le nombre d'itérations de 2000 à 5000. Le CPU reste acceptable aux alentours de 10 s en moyenne.

Tableau 3.7 – Comparaison des heuristiques de visibilité utilisées pour *IMOACS-FSH*

	CM	**%Erreur moyenne**	**%problèmes résolus**	
SPT	133.33	2.11	71.43	(45/10)**
LPT	134,01	3.03	66.23	(45/6)
LWKR	133,47	2.29	68.83	(45/8)
MWKR	133,42	2.26	68.83	(46/7)
SRM	133.85	2.82	66.23	(46/5)
LRM	133,42	2.25	70.13	(45/9)
PAL	133.92	2.85	61.04	(42/5)
CDS	133.23	1.95	68.83	(42/11)
GUP	133.28	2.04	67.32	(44/8)
DAN	133.10	1.85	75.33	(46/12)
NEH	133.48	2.26	62.34	(41/7)

** indique, pour cette colonne, que les deux chiffres entre parenthèses représentent respectivement le nombre de problèmes faciles et difficiles qui atteignent le *LB*

3.3.3 Résultats de la méthode *IOMACS-FSH*

Le Tableau 3.8 présente les valeurs des meilleures solutions obtenues par l'utilisation de l'algorithme *IOMACS-FSH*. Les expérimentations montrent que cette méthode a permis de résoudre 84.41% des instances proposées avec un pourcentage de résolution de 88.67% pour les problèmes faciles et de 75% pour ceux difficiles. De même, nous relevons que :

- sur les 53 problèmes faciles, 47 atteignent le *LB* avec une erreur moyenne de 0.93%;
- 18 problèmes difficiles parmi les 24 trouvent le *LB*, l'erreur moyenne est de 2.98%;
- l'erreur moyenne est de 1.57% pour l'ensemble des instances.

Cette qualité de solutions s'accompagne d'un temps de calcul acceptable de l'ordre de 20 secondes au pire des cas.

La comparaison des heuristiques de visibilité proposées dans cette partie (voir Tableau 3.9) indique que les meilleures heuristiques sont celles qui utilisent la règle LRM, tandis que la règle LPT est la moins performante avec un pourcentage de résolution de 62.34%.

Tableau 3.8 – Résultats obtenus par l'application de la méthode *IOMACS-FSH*

Problème	***LB***	***Solutions obtenues***					
		$BestC_{max}$	$MoyC_{max}$	$WorstC_{max}$	CPU_{Min}	CPU_{Moy}	***% Erreur***
j10s5a2	88	88	88	88	0.000	0.003	0
j10s5a3	117	117	117	117	0.000	0.004	0
j10s5a4	121	121	121	121	0.000	0.004	0
j10s5a5	122	122	122	122	0.001	0.017	0
j10s5a6	110	110	110	110	0.001	0.019	0
j10s5b1	130	130	130	130	0.000	0.001	0
j10s5b2	107	107	107	107	0.000	0.001	0
j10s5b3	109	109	109	109	0.000	0.004	0
j10s5b4	122	122	122	122	0.000	0.004	0
j10s5b5	153	153	153	153	0.000	0.001	0
j10s5b6	115	115	115	115	0.001	0.002	0
j10s5c1	**68**	**68**	**68**	**68**	**0.018**	**0.315**	**0**
j10s5c2	**74**	**74**	**74.88**	**75**	**0.159**	**1.366**	**0**
j10s5c3	**71**	**71**	**71.88**	**72**	**0.954**	**1.277**	**0**
j10s5c4	**66**	**66**	**66.44**	**67**	**0.012**	**0.823**	**0**
j10s5c5	**78**	**78**	**78**	**78**	**0.009**	**0.084**	**0**
j10s5c6	**69**	**69**	**69.04**	**70**	**0.009**	**0.53**	**0**
j10s5d1	**66**	**66**	**66**	**66**	**0.000**	**0.062**	**0**
j10s5d2	**73**	**73**	**73.76**	**74**	**0.015**	**1.322**	**0**
j10s5d3	**64**	**64**	**64**	**64**	**0.006**	**0.240**	**0**
j10s5d4	**70**	**70**	**70**	**70**	**0.004**	**0.067**	**0**
j10s5d5	**66**	**66**	**67.12**	**68**	**0.277**	**1.159**	**0**
j10s5d6	**62**	**62**	**62**	**62**	**0.003**	**0.039**	**0**
j10s10a1	139	139	139	139	0.007	0.085	0
j10s10a2	158	158	158	158	0.038	0.461	0
j10s10a3	148	148	148	148	0.004	0.03	0
j10s10a4	149	149	149	149	0.002	0.033	0
j10s10a5	148	148	148	148	0.003	0.056	0
j10s10a6	146	146	146	146	0.006	0.139	0
j10s10b1	163	163	163	163	0.002	0.008	0
j10s10b2	157	157	157	157	0.003	0.028	0
j10s10b3	169	169	169	169	0.001	0.008	0
j10s10b4	159	159	159	159	0.002	0.007	0
j10s10b5	165	165	165	165	0.002	0.014	0
j10s10b6	165	165	165	165	0.002	0.011	0
j10s10c1	113	114	115.04	115	2.794	3.023	0.88
j10s10c2	116	117	117.96	119	2.922	3.093	0.86
j10s10c3	98	116	116.12	117	2.856	3.122	18.37
j10s10c4	103	120	120.08	121	2.763	2.867	16.5
j10s10c5	121	125	126.27	128	2.77	2.915	3.31
j10s10c6	97	106	106.04	107	2.828	2.907	9.28
j15s5a1	178	178	178	178	0.004	0.016	0
j15s5a2	165	165	165	165	0.004	0.012	0
j15s5a3	130	130	130	130	0.004	0.013	0
j15s5a4	156	156	156	156	0.005	0.033	0
j15s5a5	164	164	164	164	0.002	0.017	0
j15s5a6	178	178	178	178	0.001	0.012	0
j15s5b1	170	170	170	170	0.000	0.002	0
j15s5b2	152	152	152	152	0.000	0.004	0
j15s5b3	157	157	157	157	0.001	0.005	0
j15s5b4	147	147	147	147	0.001	0.033	0
j15s5b5	166	166	166	166	0.004	0.064	0
j15s5b6	175	175	175	175	0.001	0.010	0
j15s5c1	**85**	**85**	**85.96**	**86**	**0.386**	**2.892**	**0**
j15s5c2	**90**	**90**	**91,32**	**92**	**6.488 (D)**	**14.49**	**0**
j15s5c3	**87**	**87**	**87**	**87**	**0.018**	**2.256**	**0**
j15s5c4	**89**	**89**	**90.96**	**91**	**2.180**	**3.408**	**0**
j15s5c5	**73**	**74**	**74.09**	**75**	**18.205 (D)**	**19.21**	**1.37**
j15s5c6	**91**	**91**	**91**	**91**	**0.001**	**0.082**	**0**
j15s5d1	**167**	**167**	**167**	**167**	**0.003**	**0.016**	**0**
j15s5d2	**82**	**84**	**85.2**	**86**	**20.92 (D)**	**21.88**	**2.44**
j15s5d3	**77**	**82**	**83.04**	**84**	**2.444**	**2.699**	**6.49**
j15s5d4	**61**	**85**	**86.44**	**88**	**2.967**	**3.134**	**39.34**
j15s5d5	**67**	**80**	**80.72**	**82**	**2.342**	**2.500**	**19.4**

Suite – Tableau 3.8

Problème	*LB*	*Solutions obtenues*					
		$BestC_{max}$	$MoyC_{max}$	$WorstC_{max}$	CPU_{Min}	CPU_{Moy}	*%Erreur*
j15s5d6	**79**	**81**	**81.9**	**82**	**19.32 (D)**	**21.55**	**2.53**
j15s10a1	236	236	236	236	0.008	0.027	**0**
j15s10a2	200	200	200	200	0.010	0.077	**0**
j15s10a3	198	198	198	198	0.018	0.068	**0**
j15s10a4	225	225	225	225	0.014	0.369	**0**
j15s10a5	182	182	182	182	0.064	0.297	**0**
j15s10a6	200	200	200	200	0.010	0.062	**0**
j15s10b1	222	222	222	222	0.010	0.077	**0**
j15s10b2	187	187	187	187	0.010	0.062	**0**
j15s10b3	222	222	222	222	0.010	0.062	**0**
j15s10b4	221	221	221	221	0.013	0.085	**0**
j15s10b5	200	200	200	200	0.012	0.248	**0**
j15s10b6	219	219	219	219	0.013	0.079	**0**
Moyenne	**131.597**	**132.87**	**133.07**	**133.27**	**0.581**	**0.880**	**1.57**
				% des problèmes résolus			**84.41**

Les instances en gras représentent les problèmes difficiles
(D) : indique que $t_{max} = 5000$

Tableau 3.9 – Comparaison des heuristiques de visibilité utilisées pour *IOMACS-FSH*

	CM	**%Erreur moyenne**	**%problèmes résolus**	
SPT	133.15	2.32	71.43	(46/9)**
LPT	133.67	2.49	62.34	(43/5)
LWKR	133.09	1.86	80.51	(47/15)
MWKR	132.97	1.71	80.51	(47/15)
SRM	133.16	1.94	75.32	(45/13)
LRM	132.96	1.67	81.82	(47/15)
PAL	133.33	2.06	68.83	(44/9)
CDS	133.08	1.81	79.22	(46/15)
GUP	133.21	1.93	70.13	(44/10)
DAN	133.09	1.83	71.42	(44/11)
NEH	133.36	2.13	66.23	(42/9)

** indique, pour cette colonne, que les deux chiffres entre parenthèses représentent respectivement le nombre de problèmes faciles et difficiles qui atteignent le *LB*

3.3.4 Comparaison des méthodes proposées entres elles

Nous dressons ici une analyse plus générale des résultats obtenus. Les tableaux suivants (Tableau 3.10 et Tableau 3.11) présentent un récapitulatif de la performance de chacune des méthodes proposées. Nous constatons que la méthode *IOMACS-FSH* donne de meilleures

solutions (pour les instances faciles et difficiles) que les deux autres. En ce qui concerne les problèmes faciles, les trois méthodes trouvent le *LB* avec un même pourcentage de résolution qui est de 88.68%. Les ratios de déviation relative, pour ces instances montrent la supériorité de *IOMACS-FSH*. Pour les problèmes difficiles, nous remarquons que l'approche *IOMACS-FSH* a le meilleur pourcentage de résolution comparé aux deux autres, suivi de *IMOACS-FSH*. La méthode *IOMACS-FSH* reste la meilleure avec une erreur de 2.98% contre 3.13% pour *IMOACS-FS*H et 3.6% pour *HACS-FSH*.

Tableau 3.10 – Performances des méthodes proposées

Méthodes	***%Problèmes résolus***	***%Erreur moyenne***
HACS-FSH	81.82	1.81
IMOACS-FSH	83.12	1.66
IOMACS-FSH	84.41	1.57

Tableau 3.11 – Performances relatives des méthodes proposées

Méthodes	**Problèmes faciles**		**Problèmes difficiles**	
	%Résolus	**%Erreur**	**% Résolus**	**%Erreur**
HACS-FSH	88.68	0.99	66.67	3.6
IMOACS-FSH	88.68	0.99	70.83	3.13
IOMACS-FSH	88.68	0.93	75	2.98

Les temps de calcul pour ces trois méthodes sont acceptables (ne dépassent pas les 20 secondes). Ils incluent le temps nécessaire pour la procédure de construction des algorithmes de fourmis considérées en plus des recherches locales intégrées.

Le choix de la meilleure visibilité dépend essentiellement de la nature du problème considéré (c'est-à-dire de l'instance traitée).

3.4 *COMPARAISON AVEC D'AUTRES MÉTHODES ISSUES DE LA LITTÉRATURE*

Dans cette partie, nous comparons les performances de nos approches à base de fourmis avec celles d'une procédure par évaluation et séparation (B&B) (Néron et al., 2001), d'une approche basée sur un système immunisé artificiel AIS (Engin & Döyen, 2004), d'un algorithme génétique GA (Besbes et al., 2006) et d'un algorithme de fourmis (« *Improved-AS* ») (Alaykýran et al., 2007). Le Tableau 3.12 et Tableau 3.13 donnent respectivement

$BestC_{\max}$ et le $\%Erreur$ des résultats de cette comparaison. De même, les performances de ces différents algorithmes sont présentées dans les Tableau 3.14 et Tableau 3.15. A partir des résultats obtenus, nous constatons que :

- Pour les 77 instances, notre méthode *IOMACS-FSH* atteint le *LB* avec le meilleur pourcentage de résolution (qui est de 84.41%) et la meilleure déviation par rapport au *LB* (qui est égale à 1.57%).
- Les différentes méthodes proposées dans cette partie permettent une bonne résolution du problème FSH et plus particulièrement les instances faciles ; Ainsi toutes les méthodes (sauf *Improved-AS*) arrivent à atteindre le *LB* pour 46 problèmes faciles sur les 53.
- Concernant les problèmes classés difficiles, nous pouvons remarquer que notre algorithme *IOMACS-FSH* a réussi à atteindre le *LB* à 75% suivi de *IMOACS-FSH* et de la procédure B&B et. La déviation moyenne reste la meilleure pour notre algorithme *IOMACS-FSH*. De même, la déviation de *IMOACS-FSH* est meilleure que celle de la méthode B&B, mais elle reste légèrement supérieure à celles donnée par la méthode GA. Quant à notre méthode *HACS-FSH*, elle a pu atteindre 66.67% des LB tout comme la méthode AIS. Ce taux est supérieur à celui de l'algorithme GA mais le pourcentage de déviation est plus élevé (3.6% contre 3.08%).
- Si l'on étudie uniquement les 63 instances (voir Tableau 3.16 et Tableau 3.17) qui ont été considérées par Alaykýran et al. (2007) et testées par leur algorithme de fourmis *Improved-AS*, nous avons 51 problèmes résolus par notre algorithme *IOMACS-FSH*, 50 par *IMOACS-FSH* et 49 par *HACS-FSH* contre 41 seulement par *Improved-AS*. La déviation moyenne par rapport au *LB* de notre algorithme *IOMACS-FSH* (1.92%) est légèrement inférieure à celle de l'algorithme *Improved-AS* (1.98%). En outre, nous constatons pour les 45 problèmes faciles, notre algorithme *IOMACS-FSH* a un pourcentage de résolution de 86.67% contre 64.44% pour *Improved-AS* mais la déviation de cette dernière est légèrement meilleure avec 1% contre 1.09%. Pour les instances difficiles le taux de résolution est le même pour *IOMACS-FSH* et *Improved-AS*, mais notre méthode est meilleure en terme de déviation avec une erreur moyenne de 3.98% contre 4.1%.

Tableau 3.12 – Comparaison entre les différentes méthodes de résolution considérées

Problème	*LB*	*Best*C_{max}						
		HACS-FSH	*IMOACS-FSH*	*IOMACS-FSH*	*B&B*	*Improved-AS*	*AIS*	*GA*
j10s5a2	88	88	88	88	88	88	88	88
j10s5a3	117	117	117	117	117	117	117	117
j10s5a4	121	121	121	121	121	121	121	121
j10s5a5	122	122	122	122	122	124	122	122
j10s5a6	110	110	110	110	110	110	110	110
j10s5b1	130	130	130	130	130	131	130	130
j10s5b2	107	107	107	107	107	107	107	107
j10s5b3	109	109	109	109	109	109	109	109
j10s5b4	122	122	122	122	122	124	122	122
j10s5b5	153	153	153	153	153	153	153	153
j10s5b6	115	115	115	115	115	115	115	115
j10s5c1	**68**	**68**	**68**	**68**	**68**	**68**	**68**	**68**
j10s5c2	**74**	**74**	**74**	**74**	**74**	**76**	**74**	**74**
j10s5c3	**71**	**72**	**72**	**71**	**71**	**72**	**72**	**72**
j10s5c4	**66**	**66**	**66**	**66**	**66**	**66**	**66**	**66**
j10s5c5	**78**	**78**	**78**	**78**	**78**	**78**	**78**	**78**
j10s5c6	**69**	**69**	**69**	**69**	**69**	**69**	**69**	**69**
j10s5d1	**66**	**66**	**66**	**66**	**66**	**#**	**66**	**66**
j10s5d2	**73**	**73**	**73**	**73**	**73**	**#**	**73**	**74**
j10s5d3	**64**	**64**	**64**	**64**	**64**	**#**	**64**	**64**
j10s5d4	**70**	**70**	**70**	**70**	**70**	**#**	**70**	**70**
j10s5d5	**66**	**66**	**66**	**66**	**66**	**#**	**66**	**66**
j10s5d6	**62**	**62**	**62**	**62**	**62**	**#**	**62**	**62**
j10s10a1	139	139	139	139	139	#	139	139
j10s10a2	158	158	158	158	158	#	158	158
j10s10a3	148	148	148	148	148	#	148	148
j10s10a4	149	149	149	149	149	#	149	149
j10s10a5	148	148	148	148	148	#	148	148
j10s10a6	146	146	146	146	146	#	146	146
j10s10b1	163	163	163	163	163	163	163	163
j10s10b2	157	157	157	157	157	157	157	157
j10s10b3	169	169	169	169	169	169	169	169
j10s10b4	159	159	159	159	159	159	159	159
j10s10b5	165	165	165	165	165	165	165	165
j10s10b6	165	165	165	165	165	165	165	165
j10s10c1	113	115	115	114	127	118	115	115
j10s10c2	116	119	119	117	116	117	119	119
j10s10c3	98	116	116	116	133	108	116	116
j10s10c4	103	120	120	120	135	112	120	120
j10s10c5	121	126	126	125	145	126	126	126
j10s10c6	97	106	106	106	112	102	106	106
j15s5a1	178	178	178	178	178	178	178	178
j15s5a2	165	165	165	165	165	165	165	165
j15s5a3	130	130	130	130	130	132	130	130
j15s5a4	156	156	156	156	156	156	156	156
j15s5a5	164	164	164	164	164	166	164	164
j15s5a6	178	178	178	178	178	178	178	178
j15s5b1	170	170	170	170	170	170	170	170
j15s5b2	152	152	152	152	152	152	152	152
j15s5b3	157	157	157	157	157	157	157	157
j15s5b4	147	147	147	147	147	149	147	147
j15s5b5	166	166	166	166	166	166	166	166
j15s5b6	175	175	175	175	175	176	175	175
j15s5c1	**85**	**85**	**85**	**85**	**85**	**85**	**85**	**85**
j15s5c2	**90**	**90**	**90**	**90**	**90**	**90**	**91**	**91**
j15s5c3	**87**	**87**	**87**	**87**	**87**	**87**	**87**	**87**
j15s5c4	**89**	**90**	**89**	**89**	**90**	**89**	**89**	**89**
j15s5c5	**73**	**77**	**74**	**74**	**84**	**73**	**74**	**74**
j15s5c6	**91**	**91**	**91**	**91**	**91**	**91**	**91**	**91**
j15s5d1	**167**	**167**	**167**	**167**	**167**	**167**	**167**	**167**
j15s5d2	**82**	**85**	**85**	**84**	**85**	**86**	**84**	**84**
j15s5d3	**77**	**83**	**82**	**82**	**96**	**83**	**83**	**83**
j15s5d4	**61**	**86**	**85**	**85**	**101**	**84**	**84**	**84**
j15s5d5	**67**	**81**	**80**	**80**	**97**	**80**	**80**	**79**

Suite – Tableau 3.12

Problème	*LB*	*BestC_{max}*						
		HACS-FSH	*IMOACS-FSH*	*IOMACS-FSH*	*B&B*	*Improved-AS*	*AIS*	*GA*
j15s5d6	**79**	**83**	**82**	**81**	**87**	**79**	**82**	**81**
j15s10a1	236	236	236	236	236	236	236	236
j15s10a2	200	200	200	200	200	200	200	200
j15s10a3	198	198	198	198	198	198	198	198
j15s10a4	225	225	225	225	225	228	225	225
j15s10a5	182	182	182	182	183	182	182	182
j15s10a6	200	200	200	200	200	200	200	200
j15s10b1	222	222	222	222	222	222	222	222
j15s10b2	187	187	187	187	187	188	187	187
j15s10b3	222	222	222	222	222	224	222	222
j15s10b4	221	221	221	221	221	221	221	221
j15s10b5	200	200	200	200	200	#	200	200
j15s10b6	219	219	219	219	219	#	219	219

Les instances en gras représentent les problèmes difficiles

: indique que nous ne disposons pas de la solution pour cette méthode

Tableau 3.13 – Pourcentage de déviation des différentes méthodes de résolution considérées

Problème	*LB*	*%Erreur*						
		HACS-FSH	*IMOACS-FSH*	*IOMACS-FSH*	*B&B*	*Improved-AS*	*AIS*	*GA*
j10s5a2	88	0	0	0	0	0	0	0
j10s5a3	117	0	0	0	0	0	0	0
j10s5a4	121	0	0	0	0	0	0	0
j10s5a5	122	0	0	0	0	1.64	0	0
j10s5a6	110	0	0	0	0	0	0	0
j10s5b1	130	0	0	0	0	0.77	0	0
j10s5b2	107	0	0	0	0	0	0	0
j10s5b3	109	0	0	0	0	0	0	0
j10s5b4	122	0	0	0	0	1.64	0	0
j10s5b5	153	0	0	0	0	0	0	0
j10s5b6	115	0	0	0	0	0	0	0
j10s5c1	**68**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
j10s5c2	**74**	**0**	**0**	**0**	**0**	**2.7**	**0**	**0**
j10s5c3	**71**	**1.41**	**1.41**	**0**	**0**	**1.41**	**1.41**	**1.41**
j10s5c4	**66**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
j10s5c5	**78**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
j10s5c6	**69**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
j10s5d1	**66**	**0**	**0**	**0**	**0**	**#**	**0**	**0**
j10s5d2	**73**	**0**	**0**	**0**	**0**	**#**	**0**	**1.37**
j10s5d3	**64**	**0**	**0**	**0**	**0**	**#**	**0**	**0**
j10s5d4	**70**	**0**	**0**	**0**	**0**	**#**	**0**	**0**
j10s5d5	**66**	**0**	**0**	**0**	**0**	**#**	**0**	**0**
j10s5d6	**62**	**0**	**0**	**0**	**0**	**#**	**0**	**0**
j10s10a1	139	0	0	0	0	#	0	0
j10s10a2	158	0	0	0	0	#	0	0
j10s10a3	148	0	0	0	0	#	0	0
j10s10a4	149	0	0	0	0	#	0	0
j10s10a5	148	0	0	0	0	#	0	0
j10s10a6	146	0	0	0	0	#	0	0
j10s10b1	163	0	0	0	0	0	0	0
j10s10b2	157	0	0	0	0	0	0	0
j10s10b3	169	0	0	0	0	0	0	0
j10s10b4	159	0	0	0	0	0	0	0
j10s10b5	165	0	0	0	0	0	0	0
j10s10b6	165	0	0	0	0	0	0	0
j10s10c1	113	1.77	1.77	0.88	12.39	4.42	1.77	1.77
j10s10c2	116	2.59	2.59	0.86	0	0.86	2.59	2.59
j10s10c3	98	18.37	18.37	18.37	35.71	10.2	18.37	18.37

Suite – Tableau 3.13

Problème	***LB***	***%Erreur***						
		HACS-FSH	***IMOACS-FSH***	***IOMACS-FSH***	***B&B***	***Improved-AS***	***AIS***	***GA***
j10s10c4	103	16.5	16.5	16.5	31.07	8.74	16.5	16.5
j10s10c5	121	4.13	4.13	3.31	19.83	4.13	4.13	4.13
j10s10c6	97	9.28	9.28	9.28	15.46	5.15	9.28	9.28
j15s5a1	178	0	0	0	0	0	0	0
j15s5a2	165	0	0	0	0	0	0	0
j15s5a3	130	0	0	0	0	1.54	0	0
j15s5a4	156	0	0	0	0	0	0	0
j15s5a5	164	0	0	0	0	1.22	0	0
j15s5a6	178	0	0	0	0	0	0	0
j15s5b1	170	0	0	0	0	0	0	0
j15s5b2	152	0	0	0	0	0	0	0
j15s5b3	157	0	0	0	0	0	0	0
j15s5b4	147	0	0	0	0	1.36	0	0
j15s5b5	166	0	0	0	0	0	0	0
j15s5b6	175	0	0	0	0	0.57	0	0
j15s5c1	**85**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
j15s5c2	**90**	**0**	**0**	**0**	**0**	**0**	**1.11**	**1.11**
j15s5c3	**87**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
j15s5c4	**89**	**1.12**	**0**	**0**	**1.12**	**0**	**0**	**0**
j15s5c5	**73**	**5.48**	**1.37**	**1.37**	**15.07**	**0**	**1.37**	**1.37**
j15s5c6	**91**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
j15s5d1	**167**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
j15s5d2	**82**	**3.66**	**3.66**	**2.44**	**3.66**	**4.88**	**2.44**	**2.44**
j15s5d3	**77**	**7.79**	**6.49**	**6.49**	**24.68**	**7.79**	**7.79**	**7.79**
j15s5d4	**61**	**40.98**	**39.34**	**39.34**	**65.57**	**37.7**	**37.7**	**37.7**
j15s5d5	**67**	**20.9**	**19.4**	**19.4**	**44.78**	**19.4**	**19.4**	**17.91**
j15s5d6	**79**	**5.06**	**3.8**	**2.53**	**10.13**	**0**	**3.8**	**2.53**
j15s10a1	236	0	0	0	0	0	0	0
j15s10a2	200	0	0	0	0	0	0	0
j15s10a3	198	0	0	0	0	0	0	0
j15s10a4	225	0	0	0	0	1.33	0	0
j15s10a5	182	0	0	0	0.55	0	0	0
j15s10a6	200	0	0	0	0	0	0	0
j15s10b1	222	0	0	0	0	0	0	0
j15s10b2	187	0	0	0	0	0.53	0	0
j15s10b3	222	0	0	0	0	0.9	0	0
j15s10b4	221	0	0	0	0	0	0	0
j15s10b5	200	0	0	0	0	#	0	0
j15s10b6	219	0	0	0	0	#	0	0

Les instances en gras représentent les problèmes difficiles

: indique que nous ne disposons pas de la solution pour cette méthode

Tableau 3.14 – Performances des différentes méthodes considérées sur les 77 instances

Méthodes	**% résolus**	**%Erreur moyenne**
HACS-FSH	81.82	1.81
IMOACS-FSH	83.12	1.66
IOMACS-FSH	84.41	1.57
B&B	83.12	3.64
AIS	81.82	1.66
GA	80.52	1.64

Tableau 3.15 – Performances relatives des méthodes considérées sur les 77 instances

Méthodes	Problèmes faciles		Problèmes difficiles	
	%Résolus	%Erreur	% Résolus	%Erreur
HACS-FSH	88.68	0.99	66.67	3.60
IMOACS-FSH	88.68	0.99	70.83	3.14
IOMACS-FSH	88.68	0.93	75	2.98
B&B	88.68	2.17	70.83	6.88
AIS	88.68	0.99	66.67	3.13
GA	88.68	0.99	62.50	3.08

Tableau 3.16 – Performances des différentes méthodes considérées sur les 63 instances

Méthodes	% résolus	%erreur moyenne
HACS-FSH	77.78	2.21
IMOACS-FSH	79.37	2.03
IOMACS-FSH	80.95	1.92
B&B	79.37	4.44
Improved-AS	65.08	1.89
AIS	77.78	2.03
GA	77.78	1.98

Tableau 3.17 – Performances relatives des méthodes considérées sur les 63 instances

Méthodes	Problèmes faciles		Problèmes difficiles	
	%Résolus	%Erreur	% Résolus	%Erreur
HACS-FSH	86.67	1.17	55.56	4.8
IMOACS-FSH	86.67	1.17	61.11	4.19
IOMACS-FSH	86.67	1.09	66.67	3.98
B&B	86.67	2.56	61.11	9.17
Improved-AS	64.44	1	66.67	4.1
AIS	86.67	1.17	55.56	4.17
GA	86.67	1.17	55.56	4.01

Les temps de calcul CPUs des différentes approches (excepté ceux de *Improved-AS*) sont donnés dans le Tableau 3.18. Les résolutions des différentes méthodes ont été réalisées sur des machines différentes d'où l'impossibilité de comparer les CPUs. Mais, nous pouvons

remarquer que nos méthodes ont des CPUs acceptables. Les résultats obtenus montrent aussi que les configurations des machines ont un impact sur la qualité des solutions.

Tableau 3.18 – CPU (en secondes) des différentes méthodes de résolution considérées

Problème	***LB***	***CPU***					
		HACS-FSH	***IMOACS-FSH***	***IOMACS-FSH***	***B&B***	***AIS***	***GA***
j10s5a2	88	0.001	0	0.000	13	1	0.23
j10s5a3	117	0.001	0.001	0.000	7	1	0.04
j10s5a4	121	0.001	0.001	0.000	6	1	0.12
j10s5a5	122	0.001	0.001	0.001	11	1	0.18
j10s5a6	110	0.001	0.007	0.001	6	4	0.25
j10s5b1	130	0.001	0.001	0.000	13	1	0.03
j10s5b2	107	0.001	0.001	0.000	6	1	0.05
j10s5b3	109	0.001	0.001	0.000	9	1	0.51
j10s5b4	122	0.001	0.001	0.000	6	2	0.34
j10s5b5	153	0.000	0.001	0.000	6	1	0.28
j10s5b6	115	0.001	0.001	0.001	11	1	0.02
j10s5c1	**68**	**0.102**	**0.001**	**0.018**	**28**	**32**	**0.38**
j10s5c2	**74**	**0.793**	**0.006**	**0.159**	**19**	**4**	**4.41**
j10s5c3	**71**	**1.701**	**1.505**	**0.954**	**240**	**NA**	**NA**
j10s5c4	**66**	**0.716**	**0.016**	**0.012**	**1017**	**3**	**1.35**
j10s5c5	**78**	**0.086**	**0.007**	**0.009**	**42**	**14**	**0.57**
j10s5c6	**69**	**0.482**	**0.014**	**0.009**	**4865 (AD)**	**12**	**0.84**
j10s5d1	**66**	**0.026**	**0.002**	**0.000**	**6490 (AD)**	**5**	**0.18**
j10s5d2	**73**	**0.795**	**0.036**	**0.015**	**2617 (AD)**	**31**	**NA**
j10s5d3	**64**	**0.054**	**0.009**	**0.006**	**481**	**15**	**0.18**
j10s5d4	**70**	**0.037**	**0.001**	**0.004**	**393**	**5**	**0.19**
j10s5d5	**66**	**1.367**	**0.039**	**0.277**	**1627 (AD)**	**1446**	**3.41**
j10s5d6	**62**	**0.014**	**0.001**	**0.003**	**6861 (AD)**	**8**	**0.48**
j10s10a1	139	0.005	0.003	0.007	41	1	0.23
j10s10a2	158	0.006	0.019	0.038	21	2	0.53
j10s10a3	148	0.000	0.003	0.004	58	1	0.39
j10s10a4	149	0.005	0.003	0.002	21	4	0.32
j10s10a5	148	0.005	0.007	0.003	36	1	0.19
j10s10a6	146	0.006	0.008	0.006	20	1	0.57
j10s10b1	163	0.000	0.002	0.002	36	1	0.73
j10s10b2	157	0.007	0.004	0.003	66	1	0.22
j10s10b3	169	0.000	0.002	0.001	19	1	0.28
j10s10b4	159	0.000	0.002	0.002	20	1	0.25
j10s10b5	165	0.005	0.002	0.002	33	1	0.5
j10s10b6	165	0.005	0.003	0.002	34	1	0.1
j10s10c1	113	9.786	4.156	2.794	NAD	NA	NA
j10s10c2	116	9.792	4.063	2.922	1100	NA	NA
j10s10c3	98	9.814	4.115	2.856	NAD	NA	NA
j10s10c4	103	9.797	5.274	2.763	NAD	NA	NA
j10s10c5	121	9.903	4.976	2.77	NAD	NA	NA
j10s10c6	97	10.24	6.61	2.828	NAD	NA	NA
j15s5a1	178	0.002	0.004	0.004	18	1	0.12
j15s5a2	165	0.002	0.004	0.004	35	1	0.17
j15s5a3	130	0.002	0.001	0.004	34	1	0.12
j15s5a4	156	0.002	0.004	0.005	21	2	0.27
j15s5a5	164	0.002	0.005	0.002	34	1	0.14
j15s5a6	178	0.002	0.004	0.001	38	1	0.27
j15s5b1	170	0.002	0.004	0.000	16	1	0.07
j15s5b2	152	0.002	0.004	0.000	25	1	0.06
j15s5b3	157	0.002	0.004	0.001	15	1	0.6
j15s5b4	147	0.002	0.001	0.001	37	1	0.17
j15s5b5	166	0.002	0.004	0.004	20	2	0.89
j15s5b6	175	0.002	0.004	0.001	23	1	0.08
j15s5c1	**85**	**5.84 (D)**	**0.753**	**0.386**	**2131 (AD)**	**774**	**40.08**
j15s5c2	**90**	**6.063 (D)**	**3.209**	**6.488 (D)**	**184**	**NA**	**NA**

Suite – Tableau 3.18

Problème	*LB*	*CPU*					
		HACS-FSH	*IMOACS-FSH*	*IOMACS-FSH*	*B&B*	*AIS*	*GA*
j15s5c3	**87**	**0.175 (D)**	**0.255**	**0.018**	**202**	**16**	**3.55**
j15s5c4	**89**	**4.894**	**4.867**	**2.180**	**NAD**	**317**	**2.00**
j15s5c5	**73**	**4.796**	**6.561 (D)**	**18.205 (D)**	**NAD**	**NA**	**NA**
j15s5c6	**91**	**0.864**	**0.023**	**0.001**	**57**	**19**	**0.39**
j15s5d1	**167**	**0.002**	**0.004**	**0.003**	**24**	**1**	**1**
j15s5d2	**82**	**4.888**	**3.292**	**20.920 (D)**	**NAD**	**NA**	**NA**
j15s5d3	**77**	**7.101**	**3.42**	**2.444**	**NAD**	**NA**	**NA**
j15s5d4	**61**	**4.842**	**10.958 (D)**	**2.967**	**NAD**	**NA**	**NA**
j15s5d5	**67**	**4.805**	**3.12**	**2.342**	**NAD**	**NA**	**NA**
j15s5d6	**79**	**5.886**	**4.113**	**19.315 (D)**	**NAD**	**NA**	**NA**
j15s10a1	236	0.015	0.008	0.008	40	1	0.03
j15s10a2	200	0.016	0.038	0.010	154	30	0.91
j15s10a3	198	0.010	0.01	0.018	45	4	1.86
j15s10a4	225	0.010	0.01	0.014	78	12	0.91
j15s10a5	182	0.016	0.013	0.064	c	2	0.19
j15s10a6	200	0.010	0.01	0.010	44	2	0.93
j15s10b1	222	0.015	0.01	0.010	70	3	0.04
j15s10b2	187	0.015	0.009	0.010	80	1	0.33
j15s10b3	222	0.015	0.01	0.010	80	1	0.02
j15s10b4	221	0.015	0.01	0.013	84	1	0.03
j15s10b5	200	0.015	0.016	0.012	84	1	0.77
j15s10b6	219	0.002	0.001	0.004	67	1	0.02

Les instances en gras représentent les problèmes difficiles.
NA : AIS et GA ne peuvent pas atteindre le LB au bout de 1600s
AD : B&B atteint le LB avec un CPU>1600s
NAD : B&B ne peut pas atteindre le LB
D : indique que $t_{\max} = 5000$

4 CONCLUSION

Dans ce chapitre, nous avons proposé la résolution du problème d'ordonnancement flow-shop hybride, avec pour critère d'optimisation le makespan, en employant trois approches différentes à base d'algorithmes de fourmis. Chacune de ces dernières propose une modélisation spécifique au problème considéré.

Un nombre d'heuristiques de visibilité à base de règles de priorité a été utilisé pour recueillir des informations sur l'état d'avancement de nos solutions durant le processus de construction de nos algorithmes. La détermination d'une meilleure visibilité est difficile car elle dépend essentiellement de la nature du problème.

Nos expérimentations ont permis de comparer nos algorithmes entre eux. Il en résulte que la meilleure méthode est *IMOACS-FSH*. De même, Nous remarquons que nos résultats d'une manière générale et ceux de l'approche *IMOACS-FSH* en particulier ont de bonnes performances en comparaison avec ceux obtenus avec d'autres méthodes issues de la littérature. Ceci permet de justifier l'intérêt d'utiliser les méthodes de fourmis pour la résolution du FSH.

Chapitre 4

ORDONNANCEMENT JUSTE À TEMPS POUR LE FLOW-SHOP HYBRIDE

Ce chapitre est consacré à la présentation du problème d'ordonnancement flow-shop hybride avec pour critère la minimisation de l'avance/retard. Nous commençons d'abord par exposer les travaux existants dans la littérature concernant les problèmes juste à temps. Nous décrivons ensuite des heuristiques constructives basées sur des règles de priorité issues de la littérature que nous avons adaptées pour notre cas d'étude. Nous présentons par la suite, les approches développées pour la résolution du FSH dans le cas de la somme pondérée des avances/retards et dans le cas de la somme totale des avances/retards. Le chapitre se conclut par une synthèse des résultats de l'ensemble des problèmes considérés.

1 INTRODUTION

L'environnement économique s'est fortement modifié ces dernières décennies. D'une part les délais d'attente sont devenus relativement plus restreints incitant à produire à temps. D'autre part, les entreprises font face, à une variabilité de la demande très importante pour satisfaire sa clientèle, et à une très forte concurrence mondiale. Partant de ce constat, il convient de développer des systèmes de production capables d'offrir aux clients une grande variété de produits, au bon moment, au moindre coût, et sans engendrer de stocks et des en-cours inutiles. Ainsi est né le concept de production juste à temps (JAT) (« just-in-time » en anglais). Il a été développé pendant les années 50 avec comme motivation principale une élimination des gaspillages à tous les niveaux. Contrairement à la production de masse, qui a pour objet de fabriquer plusieurs gros lots d'un même produit puis de les entreposer jusqu'à ce qu'un client passe une commande, la philosophie de la production JAT repose sur la fabrication de plusieurs produits en petits lots afin de mieux répondre à des besoins précis.

Les problèmes d'ordonnancement reflètent généralement les environnements de production. Ainsi, ils peuvent contribuer à la modélisation des caractéristiques du JAT (Hendel, 2005). Parmi ces modèles, nous retenons celui introduisant une fonction objectif qui pénalise toutes opérations se fabriquant à l'avance ou en retard par rapport à une date de fabrication souhaitée et fixée à l'avance. Cette fonction représente le critère d'ordonnancement connue sous l'appellation d'avance/retard (« earliness/tardiness ») et qu'on note ET.

Dans ce chapitre, nous nous intéressons à l'étude des problèmes d'ordonnancement dans les ateliers de production de type flow-shop hybride avec le critère ET. Deux cas sont traités : le $FHE,\left(Pm_i\right)_{i=1}^{E}\left|\ \right|\sum\left(\omega_j^E E_j+\omega_j^T T_j\right)$ et le $FHE,\left(Pm_i\right)_{i=1}^{E}\left|\ \right|\sum\left(E_j+T_j\right)$. Pour ce faire, nous procédons là encore par des approches à base d'ACO. Ainsi, nous présentons dans ce chapitre deux approches pour résoudre ces deux types de problèmes, que nous appelons respectivement ***HACS-FSHET*** et ***IACS-FSHET***. Nous adaptons également des heuristiques constructives à base de règles de priorité. Ces dernières sont proposées dans (Rajendran & Alicke, 2007) pour solutionner le problème d'ordonnancement de type flow-shop classique avec pour critère la somme des avances/retards.

2 LE CRITERE D'OPTIMISATION AVANCE/RETARD

Pour traduire la volonté de produire selon la politique du JAT, les problèmes d'ordonnancement utilisent une fonction objectif liée à la minimisation simultanée des coûts engendrés par la production anticipée, ou tardive des produits. Deux critères sont ainsi considérés : un critère d'avance qui pénalise un travail exécuté trop tôt et un critère de retard qui pénalise un travail traité en retard. Pour chaque travail j ayant une date de fin souhaitée d_j et une date de fin réelle C_j, nous définissons les indicateurs d'avance et de retard de la manière suivante :

- $E_j = \max\left(0, d_j - C_j\right)$, représente l'avance produit par le travail *j* ;
- $T_j = \max\left(0, C_j - d_j\right)$, représente le retard produit par le travail *j*.

Nous considérons, la fonction objectif avance/retard sous forme d'une combinaison linéaire des coûts d'avance et de retard des différents travaux comme suit :

$$ET = \sum_{j=1}^{n}\left(\omega_j^E \cdot E_j + \omega_j^T \cdot T_j\right) \qquad (4.1)$$

Avec :

- ω_j^E est la pénalité d'avance par unité de temps du travail j
- ω_j^T est la pénalité de retard par unité de temps du travail j.

Ces poids représentent l'importance du travail au sein du système de production. Cette fonction pénalise tout travail se terminant avant ou après sa date d'échéance. En effet, la minimisation de l'avance permet de réduire les coûts générés par les stockages inutiles et/ou la détérioration des produits, tandis que la minimisation du retard a pour rôle de réduire les coûts engendrés par le non respect de la date de livraison et/ou la perte des clients. Il s'agit donc de trouver un compromis entre ces coûts. Ce problème est considéré comme étant un problème d'ordonnancement multicritère (Hoogeveen, 2005; T'kindt & Billaut, 2002). De plus, il est lié à une catégorie de critères d'optimisation irréguliers (Gupta et al., 2002).

Nous illustrons le problème par l'exemple d'un atelier FSH formé de 3 étages. L'objectif est d'ordonnancer l'ensemble de 7 travaux de façon à trouver une solution réalisable minimisant le critère ET (relation (4.1)). Le nombre de machines par étage est respectivement $m_1 = 3$, $m_2 = 2$ et $m_3 = 4$. Dans le Tableau 4.1, nous donnons pour chaque travail *j*, la valeur de sa

date d'échéance d_j , les poids associés à l'avance ω_j^E et au retard ω_j^T et les durées opératoires p_{ij} sur chaque étage *i*.

Tableau 4.1 – Les paramètres de l'exemple illustratif

travail	d_j	ω_j^E	ω_j^T	p_{1j}	p_{2j}	p_{3j}
1	100	2	5	30	20	50
2	100	2	5	20	10	50
3	130	2	5	30	20	80
4	160	2	5	30	20	100
5	180	2	5	40	30	50
6	110	2	5	20	20	40
7	200	2	5	30	70	50

Une solution possible est représentée à l'aide du diagramme de Gantt illustré par la Figure 4.1.

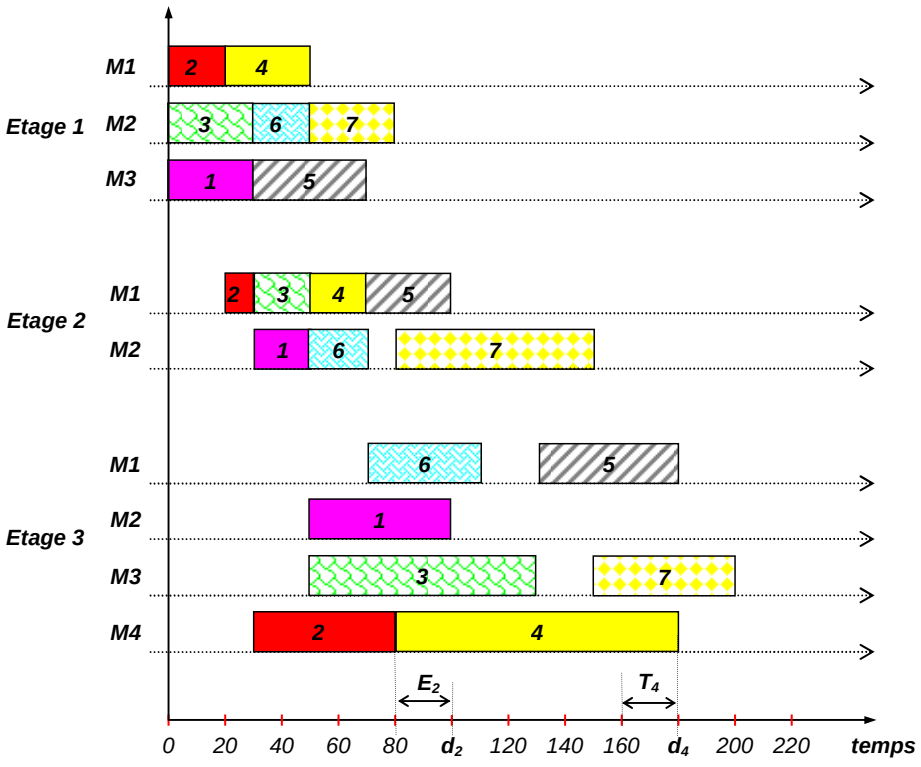

Figure 4.1 – Diagramme de Gantt correspondant à l'exemple illustratif

Nous remarquons, pour la solution obtenue au problème, que :

– le travail 4 se trouve en retard puisque sa date de fin est supérieure à sa date d'échéance ;

- le travail 2 se trouve en avance puisque sa date de fin est inférieure à sa date d'échéance ;
- les autres travaux (c'est-à-dire 1, 3, 5, 6 et 7) sont exécutés juste à temps, car la date de fin de chacun de ces travaux coïncide avec sa date d'échéance.

La fonction objectif pour cette solution vaut $ET = 2\times 20 + 5\times 20 = 140$.

3 ETAT DE L'ART

D'un point de vue pratique, les problèmes issus du JAT sont souvent difficiles (Rios-Solis, 2007). Le problème d'ordonnancement FSH monocritère, composé de deux étages où au moins deux machines sont disponibles dans l'un des étages avec la minimisation de la durée totale de l'ordonnancement, est déjà NP-difficile (Gupta, 1988). En outre, le critère d'avance/retard, avec des dates d'échéance distinctes induit généralement des problèmes NP-difficiles (Hendel & Sourd, 2007). Il est donc peu probable de trouver une solution optimale sans l'utilisation d'un algorithme énumérative mais avec un temps de calcul qui augmente exponentiellement avec la taille du problème. Par conséquent, le développement de méthodes heuristiques et/ou méta-heuristiques qui permettent de trouver des solutions acceptables en un temps raisonnable est bien justifié.

Les problèmes d'ordonnancement, qui prennent en compte les caractéristiques de la philosophie JAT, ont été intensivement étudiés dans le contexte de machines uniques. Ainsi, une variété de problèmes, considérant le cas d'une date d'échéance commune pour tous les travaux ou encore des dates d'échéance distinctes, est proposée dans (Baker & Scudder, 1990). Parmi les travaux, qui abordent ce problème, nous citons (Sourd & Kedad-Sidhoum, 2003; Merkle & Middendorf, 2005; M'Hallah, 2007; Valente & Alves, 2007). En outre, de nombreux chercheurs s'intéressent à l'étude des problèmes à machine parallèles avec minimisation des coûts d'avance/retard comme dans (Ventura & Kim, 2003; Rios-Solis, 2007). Toutefois, quelques travaux de recherche considèrent les problèmes d'ordonnancement sur des ateliers de production avec ce même critère. Rajendran et Alicke (Rajendran & Alicke, 2007) développent des règles de priorité pour résoudre le problème d'avance-retard sur un atelier flow-shop avec machines goulots d'étranglement. De même, des méthodes heuristiques sont proposées dans (Pankaj et al., 2009) pour résoudre le flow-shop de permutation avec une date d'échéance commune pour tous les travaux. Une méta-heuristique à base d'ACO est présentée dans (Huang & Yang, 2008) pour la résolution du problème ET sur un atelier job-shop. De même, Baptiste et al. (2008) proposent deux approches de

relaxation lagrangienne pour le job-shop avec pénalités d'avance/retard. Pour minimiser la somme pondérée des avances/retards dans le cas d'un atelier job-shop flexible, Wu et Weng (2005) présentent un système multi-agent qui intègre les décisions apportées à l'affectation et au séquencement. Concernant le problème FSH, nous retrouvons la notion d'avance/retard dans la fonction objectif en association avec d'autres critères dans certains travaux de recherche, comme dans (Gupta et al., 2002; Janiak et al., 2007). La plupart des travaux rencontrés dans la littérature proposent des méthodes heuristiques ou méta-heuristiques pour résoudre le flow-shop hybride avec pénalités d'avance/retard. Ainsi, dans (Fakhrzad & Heydari, 2008) une méthode heuristique employant la règle de priorité EDD (« earliest due date ») et une approche hybride composée d'une recherche tabou et d'un recuit simulé sont présentées. Dans ce cas, le nombre de machines est le même sur tous les étages. Pour résoudre ce même problème avec temps de préparation dépendant de la séquence, Behnamian et al. (2010) proposent une approche hybride qui intègre une méthode Max-Min système de fourmis, une recherche à voisinage variable et un recuit simulé. En outre, Finke et al. (2007) utilisent une approche de recherche tabou pour minimiser la somme totale des avances et des retards pour le cas particulier du flow-shop hybride avec une affectation des travaux sur les machines connue à l'avance. De même, la notion de l'ordonnancement JAT au sein de la logistique a été présentée dans les travaux de (Mouloua, 2007). Pour notre travail, nous proposons dans ce chapitre la résolution du problème flow-shop hybride avec pour critère la somme des avances/retards, en tenant compte des décisions se rapportant à l'affectation et au séquencement sans imposer que le nombre de machines soit le même sur tous les étages. Nous concevons deux approches s'inspirant de la structure de l'algorithme système de colonies de fourmis, et une approche heuristique. Une comparaison des résultats obtenus par ces différentes méthodes est présentée.

4 PRÉSENTATION DES HEURISTIQUES DE CONSTRUCTION

Dans cette partie, nous décrivons quatre heuristiques de construction à base de règles de priorité issues de la littérature, ainsi que leurs adaptations pour la résolution des deux problèmes considérés. Ces heuristiques ont été proposées pour solutionner le flow-shop classique avec comme critère la somme des avances/retards (Rajendran & Alicke, 2007). Nous présentons dans ce qui suit ces différentes règles.

- ***La règle EDD.*** Parmi les règles de priorité les plus utilisées pour les problèmes

d'ordonnancement, tenant compte des dates d'échéance d_j des travaux, nous retrouvons la règle EDD. Elle consiste à ordonner les opérations selon l'ordre croissant de d_j.

- ***La règle ODD* (« operation due date »).** Il s'agit d'une autre façon de ranger les opérations en utilisant des dates d'échéance d_{ij} relatives aux opérations et qui sont calculées à partir des dates d'échéance d_j des travaux comme suit :

$$d_{ij} = d_j \times \left(\sum_{\ell=1}^{i} p_{\ell j} \right) \Big/ \left(\sum_{\ell=1}^{E} p_{\ell j} \right) \tag{4.2}$$

Les opérations sont ainsi ordonnancées selon l'ordre croissant de ces dates d_{ij}.

A partir des deux règles EDD et ODD, Rajendran et Alicke (Rajendran & Alicke, 2007) proposent deux règles de priorité pour résoudre le flow-shop classique en présence de machines goulots qu'ils nomment « DD/BJ » et « TPT&DD/BJ ».

- ***La règle DD/BJ***, consiste à ordonner les opérations selon l'ordre croissant des indices de priorité Z_j définis comme suit :

$$Z_j = \begin{cases} d_{gj}, & \text{si } i \le g \\ d_j, & \text{si } i > g \end{cases} \tag{4.3}$$

tel que *g* représente une machine goulot d'étranglement. Le choix des opérations à un placement, se traitant sur cette machine *g* ou sur une des machines la précédant, se fait grâce à la règle ODD. Pour les opérations s'exécutant sur une des machines succédant la machine goulot, la règle EDD est utilisée. L'idée est donc de placer les opérations ayant les plus petites dates d'échéance en premier si elles se traitent avant ou sur la machine goulot. Pour les machines situées après, les opérations prioritaires sont celles qui appartiennent aux travaux ayant les plus petites dates d'échéance. Ceci permet de minimiser les avances et les retards des travaux (Rajendran & Alicke, 2007).

- ***La règle TPT&DD/BJ***, utilise les indices de priorité Z_j' pour ordonner la liste des opérations. Les indices Z_j' sont calculés comme suit :

$$Z_j^{'} = \begin{cases} d_{gj} + \sum_{\ell=i}^{g} p_{\ell j}, & \text{si} \quad i \le g \\ d_j + \sum_{\ell=i}^{E} p_{\ell j}, & \text{si} \quad i > g \end{cases} \tag{4.4}$$

Cette règle de priorité est similaire à DD/BJ sauf qu'on considère en plus la somme des durées opératoires dans le calcul des indices de priorité. L'idée est d'avoir un aperçu sur la progression du travail pour capturer les travaux ayant tendance à s'effectuer en retard.

Pour le FSH, nous adaptons ces deux dernières règles en utilisant la notion de station (ou étage) goulot d'étranglement à la place de la machine. Une telle station joue un rôle dans la limitation de la capacité de la production globalement. Nous la calculons pour chaque étage *i* le ratio entre la charge de travail et la capacité totale disponible comme suit :

$$FR_i = \frac{\sum_{j=1}^{n} p_{ij}}{m_i} \tag{4.5}$$

La station goulot est la station ayant la valeur du ratio FR_i maximale.

Après avoir rangé les opérations par l'une des règles citées précédemment, l'affectation des opérations sur chaque étage est faite sur la première machine libre.

5 ÉTUDE DU PROBLÈME $FHE,\left(Pm_i\right)_{i=1}^{E} \mid \mid \sum\left(\omega_j^E E_j + \omega_j^T T_j\right)$

Pour résoudre ce problème, nous reprenons l'algorithme *HACS-FSH* que nous avons proposé initialement pour résoudre le problème $FHE,\left(Pm_i\right)_{i=1}^{E} \mid \mid C_{\max}$ (voir Chapitre 3 Algorithme 3.1) et que nous adaptons pour ce problème. Nous appelons cet algorithme *HACS-FSHET*. Il s'agit d'une approche hiérarchique en deux temps dans laquelle une première phase s'intéresse à l'affectation des opérations aux machines. Quant, à la deuxième phase, elle détermine les décisions concernant les séquencements. Ainsi, nous modélisons ce problème par un graphe de recherche $G = \{O, C, D\}$ pour la construction de solutions possibles. Cette partie à fait l'objet d'une communication internationale (Khalouli et al., 2008b).

5.1 *DESCRIPTION DE L'APPROCHE HACS-FSHET*

5.1.1 Description de la phase d'affectation

Pour cette phase d'affectation, nous utilisons les règles décrites dans la section 4. Ainsi, les travaux sont triés selon une règle dans une liste puis, affectés à la première machine libre et éligible.

5.1.2 Description de la phase de séquencement

La phase de sélection d'opérations est réalisée grâce à la quantité de phéromones attribuée à chaque opération et aux informations représentant le problème considéré.

5.1.2.a Modélisation de la phéromone

A chaque paire d'opérations $O_{ij}, O_{hl} \in O$ (pour $1 \le i,h \le E$ et $1 \le j,l \le n$) est décernée une quantité de phéromones $\tau(O_{ij}, O_{hl})$. Une telle représentation permet d'influencer le choix d'une fourmi traversant deux opérations séquencées en tenant compte du choix des machines auxquelles elles ont été affectées.

5.1.2.b Modélisation de la visibilité

Pour mettre en évidence les coûts de pénalités d'avance et de retard (ET), nous proposons une généralisation de la règle de priorité nommée « LINET » (Ow & Morton, 1989; Valente & Alves, 2007) pour modéliser la visibilité $\eta(O_{ij}, O_{hl})$. Cette règle a été initialement proposée par (Ow & Morton, 1989) pour résoudre le problème d'ordonnancement sur machine unique. Elle attribue trois valeurs possibles à la visibilité $\eta(O_{ij}, O_{hl})$ selon la position de l'opération O_{ij} et la valeur des poids associés à l'avance et au retard comme décrite dans la relation (4.6).

$$\eta\left(O_{ij}, O_{hl}\right) = \begin{cases} W_l, & \text{si } s_l \le 0 \\ W_l - \left(s_l \times CT\right), & \text{si } 0 \le s_l \le 1/CT \\ H_l, & \text{sinon} \end{cases} \qquad (4.6)$$

où :

- $W_l = \dfrac{\omega_l^T}{p_{hl}}$, ω_l^T est la pénalité de retard du travail *l* et p_{hl} la durée opératoire de O_{hl} ;
- $H_l = \dfrac{\omega_l^E}{p_{hl}}$, ω_l^E est la pénalité d'avance du travail *l* et p_{hl} la durée opératoire de O_{hl} ;

- $s_l = d_l - t - p_{hl}$ représente la laxité du travail *l* à l'instant *t* avec d_l sa date d'échéance ;
- $CT = \dfrac{(H_l + W_l)}{max\{0, \bar{p} - p_{hl}\}}$, $\bar{p}$ étant la moyenne des durées d'exécution de tous les travaux.

Cette règle de priorité utilise une fonction de priorité reflétant le retard lorsque la valeur de s_l devient petite. Dans le cas contraire, où la valeur de s_l est importante, c'est la pénalité d'avance qui domine (Valente & Alves, 2007). L'idée étant de choisir la prochaine opération O_{hl} à l'aide de la valeur des temps morts s_l.

- Si $s_l \leq 0$, l'opération candidate a tendance à se terminer en retard si l'on tient compte de la somme des temps opératoires des opérations déjà placées. Par conséquent, le coût d'un tel travail serait plus élevé s'il est choisi plus tard. C'est pourquoi, il doit être sélectionné le plus tôt possible pour éviter que le retard n'augmente.
- Si $0 \leq s_l \leq 1/CT$, les opérations à ordonnancer dans ce cas ont tendance à ne pas se terminer en retard si on les sélectionne dès que possible. Dans le cas contraire, où ces opérations sont sélectionnées avec un certain retard en favorisant le placement d'autres travaux préalablement, ils peuvent induire des pénalités de retard. Ainsi, la priorité du choix d'un travail diminue linéairement avec l'augmentation de s_l.
- La dernière situation considère les travaux qui auront tendance à se terminer en avance.

5.1.2.c Processus de construction d'une solution

Une fourmi *k* positionnée sur un nœud O_{ij} va choisir, à partir d'une liste d'opérations candidates S_k à un instant donné, l'opération suivante *y* en utilisant la règle de transition suivante :

$$y = \begin{cases} \arg\max_{O_{hl} \in S_k} \left\{ \tau(O_{ij}, O_{hl}) \cdot \eta(O_{ij}, O_{hl})^{\beta} \right\}, & \text{si} \quad q \leq q_0 \\ Y, & \text{si} \quad q > q_0 \end{cases} \tag{4.7}$$

Rappelons que β est un paramètre qui permet de moduler l'importance relative entre τ et η ; q est une variable aléatoire uniformément distribuée sur $[0,1]$ et q_0 $(0 \leq q_0 \leq 1)$ est un

paramètre qui module les comportements associés à l'intensification et la diversification. *Y* est sélectionné aléatoirement selon la règle (4.8) :

$$\Pr_{O_{ij},Y} = \frac{\tau(O_{ij},Y).\eta(O_{ij},Y)^{\beta}}{\sum_{O_{hl}\in S_k} \tau(O_{ij},O_{hl}).\eta(O_{ij},O_{hl})^{\beta}} \quad (4.8)$$

La procédure est répétée jusqu'à ce que toutes les opérations soient sélectionnées.

5.1.2.d Mécanisme de mise à jour de phéromones

Ce mécanisme permet de simuler les changements sur les quantités de phéromones. Deux stratégies pour la mise à jour sont proposées. Ainsi, une mise à jour locale est appliquée à chaque transition comme suit :

$$\tau(O_{ij},O_{hl}) = (1-\rho_{\ell}).\tau(O_{ij},O_{hl}) + \rho_{\ell}.\tau_0 \quad (4.9)$$

De même une mise à jour globale est réalisée à la fin de chaque cycle pour intensifier les quantités de phéromones de la meilleure solution générée.

$$\tau(O_{ij},O_{hl}) = (1-\rho_g).\tau(O_{ij},O_{hl}) + \rho_g.\Delta\tau(O_{ij},O_{hl}) \quad (4.10)$$

avec

$$\Delta\tau(O_{ij},O_{hl}) = \begin{cases} 1/ET_{gb}, & \text{si } (O_{ij},O_{hl}) \in \text{à la meilleure solution} \\ 0, & \text{sinon} \end{cases}$$

ρ_g ($0 \le \rho_g \le 1$) représente le coefficient d'évaporation de phéromones et ET_{gb} est le résultat de la meilleure solution trouvée.

6 ÉTUDE DU PROBLÈME $FHE,\left(Pm_i\right)_{i=1}^{E}\left|\right.\left|\sum\left(E_j+T_j\right)\right.$

Dans cette section, nous proposons de résoudre le $FHE,\left(Pm_i\right)_{i=1}^{E}\left|\right.\left|\sum\left(E_j+T_j\right)\right.$ avec l'approche que nous appelons *IACS-FSHET*. La somme totale des avance/retard représente un cas particulier du critère présenté par la relation (4.1) où les pénalités d'avance ω_j^E et de retard ω_j^T par unité de temps sont identiques et égales à 1. Ainsi, l'avance et le retard sont pénalisés de la même manière. Notre objectif est de déterminer le meilleur ordonnancement d'un ensemble de *n* travaux candidats à l'ordonnancement caractérisés par leurs dates de fin

de fabrication au plus tard d_j et leurs durées opératoires sur un ensemble de machines regroupées en étage de production. Ce travail a fait l'objet d'une communication nationale (Khalouli et al., 2010a) et d'une publication dans une revue internationale (Khalouli et al., 2010b).

6.1 *DESCRIPTION DE L'APPROCHE IACS_FSHET*

Pour résoudre ce problème, nous proposons un algorithme à base d'ACO, dans lequel nous tenons compte des pénalités d'avance/retard.

6.1.1 Représentation d'une solution

Le codage choisi pour la représentation d'une solution illustre la succession des opérations et les machines sur lesquelles elles sont traitées sous forme d'un tableau à deux dimensions $(n\times E,2)$. Ce type de codage est parfaitement adapté à notre problème car une telle solution permet de représenter la séquence de passage de toutes les opérations de production sur chaque étage, ainsi que la machine choisie pour chaque opération parmi la liste des machines éligibles pour son traitement.

6.1.2 Processus de construction d'une solution

Nous utilisons la structure ACS pour la conception de notre approche. La génération des solutions par les fourmis artificielles se fait grâce à une fonction des propriétés locales du problème considéré via une heuristique de visibilité et grâce à l'expérience accumulée au cours de l'algorithme par le biais de la phéromone. Les modélisations associées à ces deux facteurs sont décrites ci-dessous. En outre, chaque fourmi *k* possède une mémoire sous forme d'une liste tabou L_k, lui rappelant la séquence des opérations déjà ordonnancées afin d'obliger celle-ci à respecter les contraintes de précédence et à former par conséquent une solution admissible. Il est aussi à noter, que dans cet algorithme, le séquencement des opérations s'effectue étage par étage.

6.1.2.a Modélisation de la phéromone

La phéromone permet de récolter des informations concernant l'importance du choix de la prochaine opération *y* juste après une opération *x*. Plus cette quantité est importante, plus la probabilité de choisir à nouveau l'opération *y* est forte. Le choix de cette représentation a pour

objectif d'influencer le choix des séquencements des opérations sur un même étage. Au début de l'algorithme, la quantité de phéromones se voit accorder une valeur faible et positive τ_0.

6.1.2.b Modélisation de la visibilité

L'utilisation de l'heuristique de visibilité pour diriger les fourmis à la construction probabiliste de solutions est importante du fait qu'elle donne la possibilité d'exploiter les connaissances spécifiques du problème à résoudre. Elle joue le rôle d'une règle qui favorise les opérations les plus intéressantes selon les objectifs à optimiser. Le but est d'attribuer des priorités aux opérations candidates par l'introduction d'informations liées aux caractéristiques des opérations (comme les durées opératoires, les dates d'échéances…). Pour ce faire, nous utilisons les règles de priorité EDD, ODD, DD/BJ, TPT&DD/BJ décrites dans la section 4 en les adaptant comme suit :

- ***Visibilité à base de la règle EDD.*** L'heuristique de visibilité associée à la règle EDD attribue à chaque travail un indice de rang (RI) donnant ainsi une importance à l'opération du travail selon la valeur de sa date d'échéance d_j. Dans ce cas, plus la valeur de d_j est petite, plus la valeur de l'indice du rang augmente.

$$\text{RI\_EDD}[j] = \frac{1}{d_j}$$

- ***Visibilité à base de la règle ODD.*** Si $\text{ordre\_ODD}[j]$ représente le rang du travail *j* obtenu par la règle ODD à l'étage *i*, alors nous calculons l'indice de priorité de chaque travail sur chaque étage en utilisant la relation suivante :

$$\text{RI\_ODD}[j] = \frac{n - \text{ordre\_ODD}[j] + 1}{n}$$

- ***Visibilité à base de la règle DD/BJ.*** Selon l'ordre de priorité des travaux proposé par la règle DD/BJ, la visibilité est déterminée en calculant les RI_DD/BJ comme suit :

$$\text{RI\_DD/BJ}[j] = \frac{n - \text{ordre\_DD/BJ}[j] + 1}{n}$$

- ***Visibilité à base de la règle TPT&DD/BJ.*** La solution obtenue à partir de cette règle permet de déterminer la visibilité comme suit :

$$\text{RI\_TPT\&DD/BJ}[j] = \frac{n - \text{ordre\_TPT\&DD/BJ}[j] + 1}{n}$$

6.1.2.c Construction d'une solution

Chaque fourmi construit une solution complète, composée de toutes les opérations ainsi que les machines sur lesquelles elles seront traitées. Ceci est réalisé en réitérant les deux étapes de décisions concernant la sélection d'une opération et la sélection d'une machine.

- ***La sélection d'une opération***

Pour déterminer la prochaine opération $O_{hl} \in S_k$ (S_k étant la liste des opérations candidates) à ordonnancer, chaque fourmi *k* doit appliquer la règle de transition suivante :

$$x = \begin{cases} \arg\max_{O_{hl} \in S_k} \left\{ \left[\tau(O_{ij}, O_{hl}) \right] \cdot \left[\eta(O_{ij}, O_{hl}) \right]^{\beta} \right\}, & \text{si} \quad q \le q_0 \\ X, & \text{sinon} \end{cases} \tag{4.11}$$

Les mêmes paramètres sont repris et l'opération candidate X est définie selon la règle (4.12).

$$\Pr_{O_{ij}, X} = \frac{\left[\tau(O_{ij}, X) \right] . \left[\eta(O_{ij}, X) \right]^{\beta}}{\sum_{O_{hl} \in S_k} \left[\tau(O_{ij}, O_{hl}) \right] . \left[\eta(O_{ij}, O_{hl}) \right]^{\beta}} \tag{4.12}$$

- ***La sélection d'une machine***

A chaque opération sélectionnée, nous attribuons une machine parmi la liste des machines éligibles pour son traitement. Ainsi pour chaque étage $1 \le i \le E-1$, nous proposons d'affecter chaque opération sélectionnée à la machine libre le plus tôt. Pour l'étage final, nous affectons chaque opération à la machine qui fournit la valeur minimale à la fonction objectif ET. Les règles d'affectation proposées sont des règles qui dépendent d'une connaissance instantanée sur le système. Les solutions obtenues varient donc en fonction des scénarios du processus de recherche.

La procédure de construction est répétée jusqu'à ce que toutes les opérations soient sélectionnées et affectées à une machine. Les solutions retenues sont successivement stockées dans L_k.

6.1.2.d Mécanisme de mise à jour de phéromones

Pour contrôler les quantités de phéromones durant le processus de construction de solution, nous considérons une mise à jour locale qui consiste à diminuer la valeur de la quantité de phéromones (voir relation (4.13)). Elle est effectuée dans le but de rendre moins attirante l'opération visitée pour pousser la recherche vers les autres opérations et ainsi diversifier les solutions de l'ensemble des fourmis.

$$\tau(O_{ij}, O_{hl}) = (1-\rho_\ell).\tau(O_{ij}, O_{hl}) + \rho_\ell.\tau_0 \qquad (4.13)$$

Algorithme 4.1 – Description de l'algorithme *IACS-FSHET*

Initialiser les traces de phéromone à τ_0

Déterminer les différents paramètres de l'algorithme

Répéter

Pour chaque fourmi $k=1$ **à** *nbAnts* **Faire**

Pour chaque étage $i=1$ à E **Faire**

Tant que $S_k \neq \emptyset$ **Faire**

Sélectionner une opération $O_{hl} \in S_k$ en Appliquant la règle de transition

Ranger l'opération dans la liste taboue $L_k \leftarrow L_k \cup O_{hl}$

Affecter l'opération sélectionnée à une machine éligible

Appliquer la mise à jour locale

Fin Tant que (un tour complet est réalisé)

Fin Pour

Évaluer les solutions obtenues

Fin Pour (toutes les fourmis ont terminé leur ordonnancement)

Appliquer la mise à jour globale

Jusqu'à nombre maximale des itérations atteint ou solution optimale trouvée

De même, une mise à jour globale est réalisée à la fin de chaque cycle. Ainsi, la meilleure solution trouvée met à jour les quantités de phéromones de manière proportionnelle au degré de succès obtenu pour guider les fourmis dans la construction des nouvelles solutions. Le processus consiste à évaluer la fonction objectif de chaque solution et à effectuer des

modifications sur les informations concernant les quantités de phéromones correspondant à la meilleure séquence d'opérations.

$$\tau(O_{ij},O_{hl}) = (1-\rho_g).\tau(O_{ij},O_{hl}) + \rho_g.\Delta\tau(O_{ij},O_{hl}) \quad (4.14)$$

avec $\Delta\tau(O_{ij},O_{hl}) = \begin{cases} 1/ET_{gb}, & \text{si } (O_{ij},O_{hl}) \in \text{à la meilleure solution} \\ 0, & \text{sinon} \end{cases}$

tel que ET_{gb} est le résultat de la meilleure solution trouvée pour la fonction objectif considérée.

Le fonctionnement général de l'approche *IACS-FSHET* est décrit dans Algorithme 4.1.

7 EXPÉRIMENTATIONS ET RÉSULTATS

Cette section présente les résultats des évaluations conduites par les deux approches proposées *HACS-FSHET* et *IACS-FSHET*. Pour chaque cas, nous étudions les performances des algorithmes et nous les comparons avec les résultats obtenus par l'utilisation des heuristiques constructives décrites dans la section 4. A notre connaissance et jusqu'à ce jour, il n'existe pas dans la littérature d'instances pour les problèmes que nous considérons dans cette partie. Nous avons donc généré nos propres instances. Les expérimentations ont été effectuées en utilisant ce jeu de paramètres :

- le nombre de travaux est respectivement 25, 50, 100 ;
- le nombre d'étages est respectivement 2, 5, 8 (pour le cas *IACS-FSHET*) et 2, 5, 8, 10 (pour le cas *HACS-FSHET*) ;
- le nombre de machines par étage m_i est choisi de manière aléatoire entre $[1,5]$ et tel qu'au moins un étage est composé de $m_i > 1$;
- les durées opératoires p_{ij} sont générées aléatoirement selon une loi de distribution discrète uniforme comprise entre 1 et 100 ;
- Comme dans (Gupta et al., 2002), nous attribuons une date d'échéance pour chaque travail en utilisant l'équation :

$$d_j = r_j + (1+c) \times \sum_{i=1}^{E} p_{ij} \qquad (4.15)$$

avec $c \in [0,1]$ et r_j la date de disponibilité du travail (dans notre cas d'étude la valeur de r_j est négligeable).

- les pénalités d'avance ω_j^E par unité de temps sont aléatoirement générés dans l'intervalle $[1,5]$;
- les pénalités de retard ω_j^T par unité de temps sont aléatoirement générés dans l'intervalle $[1,10]$;

Les développements informatique ont été codés en JAVA et réalisés sur un PC équipé d'un Intel (R) Core (TM) 2 DUO CPU condensé à 2.27 GHz et ayant 3 Go de mémoire vive (RAM).

7.1 CAS DU PROBLÈME $FHE, \left(Pm_i\right)_{i=1}^{E} \mid \mid \sum \left(\omega_j^E \cdot E_j + \omega_j^T \cdot T_j\right)$

Le Tableau 4.2 récapitule les meilleurs paramètres obtenus à partir des expérimentations préliminaires de l'algorithme *HACS-FSHET*. Ces paramètres incluent le nombre de fourmis $nbAnt$, le nombre d'itérations $t_{\max}$, la quantité de phéromone initiale τ_0, les paramètres β et q_0, ainsi que les coefficients d'évaporation de phéromone locale ρ_ℓ et globale ρ_g.

Pour chaque instance, une série de 10 réplications est effectuée. Nous comparons les résultats de *HACS-FSHET* avec ceux donnés par les quatre heuristiques de construction utilisant les règles de priorité EDD, ODD, DD/BJ et TPT&DD/BJ. Nous calculons la déviation de l'erreur relative de chaque méthode par rapport à la meilleure solution trouvée comme suit :

$$\%Déviation = 100 \times \frac{ET_A - ET_{Best}}{ET_{Best}} \qquad (4.16)$$

où ET_{Best} est la meilleure solution obtenue pour les méthodes EDD, ODD, DD/BJ, TPT&DD/BJ et le *HACS-FSHET*. ET_A est la solution obtenue pour chacune des méthodes $A \in \{\text{EDD, ODD, DD/BJ, TPT\&DD/BJ, } HACS\text{-}FSHET\}$.

Tableau 4.2 – Valeurs des paramètres utilisées pour l'algorithme *HACS-FSHET*

Paramètres	**Plages des valeurs**	***Meilleure valeur***
$nbAnt$	−−	20
$t_{\max}$	−−	500
τ_0	−−	0.1
β	$0 \le \beta \le 10$	3
q_0	$0 \le q_0 \le 1$	0.8
ρ_ℓ	$0 \le \rho_\ell \le 1$	0.1
ρ_g	$0 \le \rho_g \le 1$	0.1

Les résultats relatifs à la somme pondérée des avances/retards sont présentés dans le Tableau 4.3. Dans de dernier, on peut constater que l'algorithme *HACS-FSHET* donne toujours les meilleurs résultats pour les instances utilisées.

Tableau 4.3 – Résultats obtenus par l'algorithme *HACS-FSHET*

Problème		**%Déviation de l'erreur relative**					**CPU****
n	***E***	**EDD**	**ODD**	**DD/BJ**	**TPT&DD/BJ**	**HACS-FSHET**	
25	2	5.54	16.21	4.56	8.73	0	0.02
	5	15.69	74.79	34.25	52.99	0	0.08
	8	0.46	176.51	75.66	144.34	0	0.19
	10	4.59	254.15	110.92	242.56	0	0.28
Moyenne		**6.57**	**130.42**	**56.35**	**112.56**	**0**	**0.14**
50	2	12.85	2.95	7.44	7.44	0	0.1
	5	8.63	192.28	57.86	95.58	0	0.41
	8	9.35	201.15	80.98	125.83	0	1.16
	10	0	377.74	101.15	192.08	0	1.75
Moyenne		**7.71**	**193.71**	**61.91**	**105.23**	**0**	**0.86**
100	2	5.69	14.14	15.9	15.91	0	0.49
	5	4.46	356.83	190.91	210.86	0	2.58
Moyenne		**5.08**	**185.5**	**103.41**	**113.39**	**0**	**1.54**

** : Le CPU relevé est le CPU moyen par itération de l'algorithme *HACS-FSHET*
Le CPU des méthodes heuristiques EDD, ODD, DD/BJ et TPT&DDBJ ne dépasse pas 1 seconde.

7.2 *CAS DU PROBLÈME* $FHE,\left(Pm_i\right)_{i=1}^{E}\left|\ \right|\sum\left(E_j+T_j\right)$

Pour utiliser l'algorithme *IACS-FSHET*, des expérimentations préliminaires ont été conduites sur quelques instances pour trouver les meilleurs paramètres (voir Tableau 4.4).

Tableau 4.4 – Valeurs des paramètres utilisées pour l'algorithme *IACS-FSHET*

Paramètres	**Plages des valeurs**	***Meilleure valeur***
$nbAnt$	−−	10
$t_{\max}$	−−	2000
τ_0	−−	0.01
β	$0 \le \beta \le 10$	2
q_0	$0 \le q_0 \le 1$	$\frac{\log(t)}{\log(t_{\max})}$
ρ_ℓ	$0 \le \rho_\ell \le 1$	0.001
ρ_g	$0 \le \rho_g \le 1$	0.001

Pour chaque combinaison ($n \times E$), cinq instances sont créées. De plus, une série de 10 réplications est effectuée pour chaque instance et avec les différentes heuristiques de visibilités. Le Tableau 4.5 donne la solution moyenne (notée Moy) et la meilleure solution (notée Best) obtenues à partir des 10 réplications par l'application de notre algorithme *IACS-FSHET* pour chaque instance. De même, le temps moyen d'exécution CPU est relevé.

Nous remarquons que l'association de la visibilité RI_TPT&DD/BJ avec l'algorithme *IACS-FSHET*, pour la résolution les instances générées, donne les meilleures solutions dans 80% des cas.

Les meilleures solutions obtenues par l'algorithme *IACS-FSHET* et celles obtenues par les heuristiques constructives utilisant les règles de priorité EDD, ODD, DD/BJ et TPT&DD/BJ sont rassemblées dans le Tableau 4.6. Pour calculer le pourcentage de gain entre notre algorithme *IACS-FSHET* et la meilleure solution obtenue à partir des heuristiques constructives (CH) nous utilisons la formule suivante :

$$\%Gain = 100 \times \frac{ET_{CH} - ET_{IACS-FSHET}}{ET_{CH}} \qquad (4.17)$$

De même, nous pouvons conclure que les résultats obtenus, par cette approche, sont supérieurs à ceux obtenus à partir des heuristiques constructives. Ainsi, nous enregistrons une amélioration de 16.22% en moyenne de notre approche *IACS-FSHET* par rapport aux meilleurs résultats obtenus par les heuristiques.

Nous notons aussi que les temps de calcul de *IACS-FSHET* augmentent lorsque la taille du problème augmente mais restent raisonnables. Cependant, les heuristiques constructives sont rapides.

Tableau 4.5 – Résultats obtenus par l'algorithme *IACS-FSHET*

Problème			$ET_{IACS-FSHET}$			
$n \times E$			***RI_EED***	***RI_ODD***	***RI_DD/BJ***	***RI_TPT&DD/BJ***
25×3	**1**	**Moy**	4108,8	4315,2	3924,4	3928
		Best	4070	4305	**3908**	3915
		CPU	3,84	3,78	3,76	3,83
	2	**Moy**	2455.8	2344.8	2346.8	2287.8
		Best	2427	2319	2334	**2261**
		CPU	3.71	3.6	3.76	3.63
	3	**Moy**	10978.4	10618	10624.6	10625.6
		Best	10915	10604	10604	**10599**
		CPU	3.62	3.56	3.58	3.59
	4	**Moy**	870.8	1181.6	866.2	866
		Best	853	1157	**851**	856
		CPU	3.65	3.7	3.63	3.59
	5	**Moy**	3704.8	4887	3553.4	3722.2
		Best	3648	4702	**3525**	3676
		CPU	3.67	3.62	3.63	3.65
25×5	**1**	**Moy**	12319.8	10146.8	10083.2	9925.2
		Best	12225	10096	10035	**9869**
		CPU	7.04	6.97	6.97	6.96
	2	**Moy**	8661.6	10272.8	8690	8320.6
		Best	8507	10093	8619	**8191**
		CPU	7.15	7.07	7.3	7.16
	3	**Moy**	8187.8	10516.4	8022	8227.4
		Best	8095	10355	**7919**	8132
		CPU	6.94	6.93	6.93	7.01
	4	**Moy**	11213	11897.6	10858.4	10742.4
		Best	11140	11766	10678	**10473**
		CPU	6.94	6.89	6.92	6.9
	5	**Moy**	3537	4495	3509.8	3440
		Best	3503	4352	3450	**3373**
		CPU	6.96	6.94	6.93	6.93
25×8	**1**	**Moy**	8647.4	9733.2	8757.2	8088.2
		Best	8475	9632	8584	**8001**
		CPU	13.22	13.67	13.35	13.4
	2	**Moy**	4230	4965.6	3900.4	4079.8
		Best	4075	4913	**3825**	4035
		CPU	12.87	13.12	13.25	13.37
	3	**Moy**	8285.6	10292.6	8219	7909.4
		Best	8079	10187	8142	**7696**
		CPU	13.46	13.19	13.18	12.97
	4	**Moy**	10557	11016.6	9081.8	9263.6
		Best	10353	10904	**9038**	9183
		CPU	13.28	12.87	12.82	13.22
	5	**Moy**	6795.6	8223.8	6871.2	6806.6
		Best	**6717**	7984	6842	6790
		CPU	12.9	12.85	12.77	12.82

Suite – Tableau 4.5

Problème			$ET_{IACS-FSHET}$			
$n \times E$			***RI_EED***	***RI_ODD***	***RI_DD/BJ***	***RI_TPT&DD/BJ***
50×3	**1**	**Moy**	11237.6	14147.8	11183	11150.8
		Best	11207	14042	11139	**11063**
		CPU	12.42	12.4	12.46	12.44
	2	**Moy**	19203	20243.6	17885.6	17696.8
		Best	19175	20063	17846	**17657**
		CPU	12.89	12.73	12.62	12.54
	3	**Moy**	10548	11860.8	10473	10174.8
		Best	10493	11703	10404	**10124**
		CPU	14.19	14.59	13.73	13.14
	4	**Moy**	37804.2	31141.8	31135	31006.4
		Best	37212	31094	31106	**30971**
		CPU	14	13.19	13.83	13.81
	5	**Moy**	21645.2	26042	21777.4	21343.2
		Best	21538	25893	21529	**21157**
		CPU	14.97	13.61	13.22	13.43
50×5	**1**	**Moy**	20468.2	21893.6	19039.6	18563.2
		Best	20162	21711	18895	**18400**
		CPU	25.08	24.96	25.57	26.24
	2	**Moy**	13796.6	18800.2	13957.2	13317.8
		Best	13658	18326	13806	**13046**
		CPU	24.73	59.79	77.06	25.25
	3	**Moy**	56810.8	68001.6	52468.4	51643.4
		Best	56240	67567	52362	**51541**
		CPU	24.01	24.02	24.04	24.04
	4	**Moy**	59685	67889.2	57607.2	56300.4
		Best	58896	67587	56751	**56078**
		CPU	24.02	23.97	24.01	24
	5	**Moy**	44209.4	54278.4	40559.8	39968.8
		Best	43019	54064	40231	**39876**
		CPU	24.1	24.09	24.04	24.03
50×8	**1**	**Moy**	38262.6	47220.8	38076.2	37653.4
		Best	38114	46854	37790	**37375**
		CPU	47.19	47.49	46.83	46.91
	2	**Moy**	21147.8	25917.4	22220.8	22280
		Best	**20986**	25809	21957	21836
		CPU	46.92	47.48	48.44	48.27
	3	**Moy**	49891	57220.6	49534.2	48966
		Best	49222	56679	49184	**48188**
		CPU	47.02	46.78	46.7	46.56
	4	**Moy**	18276.4	18848	16833.6	16268.4
		Best	18205	18776	16766	**16236**
		CPU	46.73	47.52	47.16	46.62
	5	**Moy**	48299.4	54856	47631.2	45901
		Best	47971	54498	46905	**45671**
		CPU	47.91	47.6	47.36	46.84

Suite – Tableau 4.5

Problème			$ET_{IACS-FSHET}$			
$n \times E$			***RI_EED***	***RI_ODD***	***RI_DD/BJ***	***RI_TPT&DD/BJ***
100×3	**1**	**Moy**	188636	223805.4	188912	183741.8
		Best	187471	217368	188464	**183046**
		CPU	47.56	47.29	47.7	**47.94**
	2	**Moy**	91333.4	79771.4	79724.8	79413.2
		Best	91109	79574	79522	**79309**
		CPU	47.29	47.18	47.29	47.35
	3	**Moy**	58427.6	65114.2	58350.8	55862.6
		Best	57989	64863	57625	55561
		CPU	47.49	47.31	47.45	47.58
	4	**Moy**	51452.6	63118.8	48486.6	47793.8
		Best	51247	62906	48255	**47745**
		CPU	47.51	47.78	47.45	47.41
	5	**Moy**	174080.8	147460.2	147443.4	145092.6
		Best	172949	147345	147360	**145050**
		CPU	47.57	47.29	47.21	47.25
100×5	**1**	**Moy**	195216.4	215190.2	187804	184473.4
		Best	194222	214068	187284	**184136**
		CPU	95.06	94.98	94.94	94.99
	2	**Moy**	210893.6	173708.2	173722.4	170664.4
		Best	209854	173547	173491	**170425**
		CPU	94.94	94.59	94.51	94.55
	3	**Moy**	205714.4	241100	206802.2	203877.6
		Best	205086	238648	205877	202256
		CPU	95.12	94.77	94.78	94.94
	4	**Moy**	105337.8	117977.6	96153.4	95228.4
		Best	104514	117548	96079	**94874**
		CPU	95.29	94.97	94.78	94.73
	5	**Moy**	182734	216935.4	182595.4	180891.4
		Best	182007	215722	182319	**179937**
		CPU	94.81	94.9	94.99	95.24
100×8	**1**	**Moy**	185876.2	213428.2	169959.6	166445.2
		Best	185399	212484	169598	**166004**
		CPU	192.31	193.39	192.31	192.93
	2	**Moy**	213979.4	224857	209713.2	205235
		Best	213235	224327	208713	**204627**
		CPU	192.68	193.1	195.79	192.83
	3	**Moy**	83948.6	97587.2	77715.4	78117.4
		Best	83780	97426	**77292**	77784
		CPU	193.28	193.72	193.34	193.64
	4	**Moy**	210381.6	211082.2	192174.8	188737.8
		Best	204305	210533	191385	**188462**
		CPU	192.78	192.03	193.15	192.5
	5	**Moy**	88104	98755.2	81091	80137.6
		Best	87982	98563	80900	**79672**
		CPU	193.29	192.06	193.69	194.46

Tableau 4.6 – Résultats obtenus par l'algorithme *IACS-FSHET*

Problème		**Heuristiques constructives**	**Algorithme *IACS-FSHET***		**%Gain**
n	***E***	$BestET_{HC}$	$BestET_{ACS-FSHET}$	**CPU(s)**	
25	3	4562	3908	3.8	14.34
	3	2718	2261	3.68	16.81
	3	11703	10599	3.59	9.43
	3	1086	851	3.64	21.64
	3	4499	3525	3.64	21.65
Moyenne		**4913.6**	**4228.8**	**3.67**	**16.77**
25	5	13487	9869	6.98	26.83
	5	10288	8191	7.17	20.38
	5	9603	7919	6.95	17.54
	5	12324	10473	6.91	15.02
	5	4286	3373	6.94	21.3
Moyenne		**9997.6**	**7965**	**6.99**	**20.21**
25	8	10665	8001	13.41	24.98
	8	5279	3825	13.15	27.54
	8	11691	7696	13.2	34.17
	8	13227	9038	13.05	31.67
	8	8209	6717	12.84	18.18
Moyenne		**9814.2**	**7055.4**	**13.13**	**27.31**
50	3	12353	11063	12.43	10.44
	3	19815	17657	12.69	10.89
	3	12192	10124	13.91	16.96
	3	40619	30971	13.71	23.75
	3	23179	21157	13.81	8.72
Moyenne		**21631.6**	**18194.4**	**13.31**	**14.15**
50	5	23773	18400	25.46	22.6
	5	14782	13046	46.71	11.74
	5	59021	51541	24.03	12.67
	5	64433	56078	24	12.97
	5	47235	39876	24.06	15,58
Moyenne		**41848.8**	**35788.2**	**28.85**	**15.11**
50	8	40342	37375	47.11	7.35
	8	26220	20986	47.78	19.96
	8	55225	48188	46.77	12.74
	8	19984	16236	47.01	18.76
	8	53929	45671	47.43	15.31
Moyenne		**39140**	**33691.2**	**47.22**	**14.82**
100	3	193694	183046	47.62	5.5
	3	93559	79309	47.28	15.23
	3	63097	55561	47.46	11.94
	3	55290	47745	47.54	13.65
	3	180342	145050	47.33	19.57
Moyenne		**117196.4**	**102142.2**	**47.45**	**13.18**
100	5	204595	184136	94.99	10
	5	215189	170425	94.65	20.8
	5	213604	202256	94.9	5.31
	5	108290	94874	94.94	12.39
	5	186678	179937	94.99	3.61
Moyenne		**185671.2**	**166325.6**	**94.89**	**10.42**
100	8	192321	166004	192.73	13.68
	8	226339	204627	193.6	9.59
	8	92324	77292	193.5	16.28
	8	222633	188462	192.61	15.35
	8	93774	79672	193.41	15.04
Moyenne		**165478.2**	**143211.4**	**193.17**	**13.99**

8 CONCLUSION

Dans ce chapitre, nous nous sommes intéressés au problème flow-shop hybride avec machines identiques sur chaque étage. L'objectif est la minimisation des coûts engendrés par les pénalités d'avance-retard lorsque les dates d'échéances des travaux sont distinctes. Deux cas d'étude ont été considérés : la somme totale des avances/retards ainsi que leur somme pondérée. Pour résoudre ces problèmes, nous avons présenté deux algorithmes à base de colonie de fourmis. En outre, quatre heuristiques constructives ont été développées en utilisant des règles de priorités issues de la littérature et tenant compte des pénalités d'avance/retard pour comparer les performances de nos approches. Un ensemble de tests a été conduit en utilisant des instances générées de façon aléatoire. Les résultats obtenus montrent que la performance de nos méthodes *HACS-FSHET* et *IACS-FSHET* sont meilleures par rapport aux heuristiques constructives à base des règles EDD, ODD, DD/BJ et TPT&DD/BJ. Les approches proposées peuvent être considérées comme prometteuses pour la résolution de ce type de problème réputé difficile.

L'étude de ce problème a un intérêt pratique dans le sens où l'atelier de production que nous étudions est fréquemment rencontré dans l'industrie et fait partie des systèmes hybrides (ou encore flexibles). De plus, l'implémentation de la production JAT en ordonnancement au sein des environnements de production présente un bénéfice majeur pour les industriels et les clients. Elle incite l'augmentation de la compétitivité. Dans ce travail, cette philosophie de production est traduite par l'introduction du critère avance/retard en ordonnancement.

Chapitre 5

CONTRIBUTION À LA RÉSOLUTION D'UN FLOW-SHOP HYBRIDE MULTICRITÈRE

Ce chapitre est dédié à un problème d'ordonnancement multicritère sur un atelier flow-shop hybride considérant le makespan et la somme pondérée des avances/retards comme critères d'optimisation. Dans cette contribution, nous employons une méta-heuristique à base de colonie de fourmis et un système d'aide à la décision pour l'évaluation des solutions générées.

1 INTRODUCTION

De nombreux travaux de recherche en ordonnancement n'optimisent qu'un seul critère, alors que les problèmes industriels sont généralement de nature multicritère (T'kindt & Billaut, 2002). En effet, l'ordonnancement d'ateliers nécessite la prise en compte de plusieurs critères ou objectifs à satisfaire simultanément (minimisation du temps moyen de séjour des produits, minimisation des retards, minimisation des stockages de produits, maximisation des taux d'utilisation des équipements, …), dont certains sont contradictoires entre eux. Ainsi, il est peu probable de trouver des solutions optimales pour tous les critères simultanément. Dans ce cas, la meilleure solution, est la solution ayant obtenue la meilleure évaluation au regard des critères définis. Les différents critères intervenant dans une décision ne sont évidemment pas appréciés de la même manière. En effet, l'évaluation dépend généralement du décideur, du lieu ou du moment de prise de décision. D'où l'importance du concept d'optimum de Pareto formulé par l'économiste Vilfredo Pareto (Talbi, 1999) et qui constitue les fondements de la recherche en optimisation multicritère.

Dans ce chapitre, nous souhaitons évaluer la qualité de l'ordonnancement d'un atelier FSH, en considérant la minimisation simultanée de la date d'achèvement maximale de tous les travaux :

$$f_1 = C_{\max} = \max_{1 \le j \le n} C_j \tag{5.1}$$

et de la somme des avances et des retards de tous les travaux :

$$f_2 = \sum_{j=1}^{n} (\omega_j^E \cdot E_j + \omega_j^T \cdot T_j) \tag{5.2}$$

L'objectif de ce travail s'inscrit dans le développement d'une méthodologie d'aide à la décision multicritère (Pomerol & Barba-Romero, 2000). Il vise à proposer une démarche s'appuyant sur la résolution du problème d'ordonnancement d'un atelier FSH en utilisant une méta-heuristique d'optimisation par colonie de fourmis et d'un module de décision pour l'évaluation des solutions. L'évaluation proposée consiste à transformer le problème multicritère en un problème monocritère en combinant les différents critères sous forme d'une seule fonction objectif en utilisant l'intégrale floue de Choquet. Le choix des ACO pour la résolution du problème d'ordonnancement présente l'avantage de conduire en fin de recherche à une population diversifiée de solutions. Si plusieurs critères sont à prendre en

compte, cette diversité est un élément avantageux puisqu'elle permet de disposer de solutions variées.

Nous commençons ce chapitre par l'introduction des notions de base en rapport avec l'ordonnancement multicritère. Nous passons ensuite à la description de l'approche proposée ainsi que du module d'évaluation intégrée pour l'aide à la décision. Une partie est, par la suite, consacrée à la présentation des résultats expérimentaux pour la validation de notre approche. Ce Travail a fait l'objet d'une communication internationale (Khalouli et al., 2008c).

2 DÉFINITION DES PROBLÈMES MULTICRITÈRES

Un problème multicritère peut être défini comme un problème où l'on cherche l'action qui satisfait un ensemble de contraintes et optimise un vecteur de fonction coût. La difficulté se manifeste par le fait qu'on ne peut pas trouver une solution « optimale unique » pour tous les critères considérés à la fois, comme pour le cas des problèmes monocritères, mais un ensemble de solutions, en utilisant la notion d'optima de Pareto (Talbi, 1999). Toute solution de cet ensemble est « optimale » dans le sens où aucune amélioration ne peut s'effectuer sur une composante du vecteur fonction coût sans la dégradation d'au moins une autre composante du vecteur (Zitzler & Thiele, 1999).

Si $f_1, f_2, \cdots, f_{n_c}$ représentent les $n_c, (n_c \geq 2)$ critères à évaluer, un problème d'optimisation multicritère peut être, pour certains cas, définit formellement comme suit :

$$\underset{x \in \Omega}{\text{minimiser}}\ F(x) = \left(f_1(x), f_2(x), \cdots, f_{n_c}(x) \right) \tag{5.3}$$

où Ω représente l'espace des solutions réalisables, $x = (x_1, x_2, \cdots, x_L)$ est le vecteur des variables de décision et $F(x)$ est le vecteur des critères. Il peut aussi s'agir de maximiser le vecteur $F(x)$ pour d'autres cas.

La complexité des problèmes multicritère fait en sorte qu'il est souvent difficile d'utiliser des méthodes exactes pour les résoudre. Les méthodes approchées représentent donc une stratégie de résolution plus appropriée. Parmi ces dernières nous citons les algorithmes évolutionnaires, la recherche tabou, le recuit simulé, etc. (T'kindt & Billaut, 2002) qui ont été utilisées pour ce type de problèmes. Ces approches permettent de générer un ensemble de solutions, il reste

donc à trouver un compromis entre les différents critères considérés. C'est la mission du décideur (« decision-maker » en anglais), qui est l'intervenant principal à qui s'adresse l'aide à la décision. La notion de décideur « *désigne en dernier ressort l'entité qui apprécie le possible et les finalités, exprime les préférences et est sensée les faire prévaloir dans l'évolution du processus* » (Roy, 1985). Son intervention peut se faire de plusieurs manières dans les approches de résolution. Ainsi, un ensemble d'approches a été proposé dans la littérature. Elles peuvent être classifiées en trois catégories (Talbi, 1999; T'kindt & Billaut, 2002) :

- La première se compose des méthodes basées sur la transformation du problème multicritère en un problème monocritère. Dans ce cas le système retourne une solution au décideur. Ce dernier doit donc donner sa préférence entre les critères au préalable, sous forme de poids associés par exemple. Il s'agit d'approche dite a priori. Parmi elles, on y trouve les méthodes d'agrégation, ε-contrainte, de programmation par but.
- La seconde catégorie inclut les méthodes dites non Pareto pour lesquelles la recherche de la meilleure solution est réalisée en traitant séparément les différents critères. Dans ce cas, on parle d'approches interactives où le décideur intervient dans le système de résolution, en précisant la direction de recherche qu'il préfère.
- Concernant la dernière catégorie, elle renferme les méthodes Pareto qui utilisent la notion de dominance pour la sélection des solutions générées. Il s'agit d'approches a posteriori. Ainsi, seul un nombre restreint de ces solutions est intéressant. Pour déterminer ces solutions, il existe une relation de dominance entre chacune de ces solutions et les autres solutions qu'on définit comme suit : les solutions qui dominent les autres mais ne se dominent pas entre elles sont appelées solutions optimales au sens de Pareto (ou solutions non dominées). Par définition, un vecteur x domine le vecteur y lorsqu'on vérifie les deux conditions suivantes :
    - x est au moins aussi bon que y pour tous les critères, $\forall i : f_i(x) \leq f_i(y)$;
    - x est strictement meilleur que y pour au moins un critère, $\exists j : f_j(x) < f_j(y)$.

Dans ce manuscrit, nous nous intéressons à la première catégorie de résolution avec une conception d'aide à la décision qui utilise une méthode d'agrégation. Naturellement, aucune méthode d'agrégation ne respecte la totalité des exigences. Il faut donc décider sur quelle exigence on va céder. Un des aspects significatifs dans les problèmes d'agrégation est la prise en compte de l'importance des critères considérés en leur attribuant des poids. Ces derniers

sont modélisés généralement par l'utilisation de fonctions d'agrégation pondérées. Les opérateurs d'agrégation les plus utilisés sont les moyennes arithmétiques pondérées pour exprimer les préférences du décideur sur l'ensemble des critères (Grabisch, 1996; Marichal, 1998). Ils ont l'inconvénient de ne pas pouvoir modéliser une quelconque interaction parmi les critères, puisqu'ils considèrent leur indépendance. L'intégrale de Choquet permet de résoudre le problème de la compensation entre les critères. Il s'agit d'un opérateur d'agrégation pondéré capable de considérer l'interaction parmi les différents critères (Dubus et al., 2009). Ainsi, pour définir les préférences du décideur, nous utilisons cette intégrale floue pour l'évaluation des solutions générées.

3 MÉTHODE D'AGRÉGATION PAR L'INTÉGRALE DE CHOQUET

L'intégrale de Choquet est considérée comme un opérateur d'agrégation qui a pour objectif d'améliorer la puissance de l'analyse multicritère en généralisant la moyenne arithmétique pondérée par la prise en compte de l'interaction entre les critères (Grabisch, 1996). Cette notion d'intégrale de Choquet permet donc de modéliser les phénomènes d'interaction entre les critères telles que la corrélation et la dépendance préférentielle (Marichal, 1998). Il s'agit d'un cas particulier des intégrales floues utilisées en aide multicritère à la décision. L'intégrale de Choquet utilise des mesures floues pour prendre en compte l'importance relative de chaque critère ainsi que les interactions mutuelles entre eux. Dans ce cas, les poids dépendent des rangs et d'une mesure floue permettant d'exprimer le degré d'importance entre une combinaison de critères. Ces mesures sont donc capables de modéliser la dépendance entre les critères. Elles ont été proposées initialement par Sugeno (Sugeno, 1977) pour généraliser les mesures additives. Dans ce qui suit nous proposons quelques définitions (Grabisch, 1996), pour mieux comprendre le concept de l'intégrale de Choquet.

3.1 *DÉFINITION DE LA MESURE FLOUE*

Une mesure floue sur un ensemble $N_c = \{1,2,..,n_c\}$ est une fonction $\mu : P(N_C) \rightarrow [0,1]$, tels que :

1) $\mu(\varnothing) = 0$ et $\mu(N_C) = 1$;
2) $\forall S \subset D \subset N_c \Rightarrow \mu(S) \leq \mu(D)$ (cette relation vérifie que la fonction est monotone).

Dans le contexte de l'agrégation multicritère, le coefficient $\mu(S)$ peut être vu comme le poids ou l'importance du sous-ensemble de critères $S \subset N_c$.

3.2 *L'INTÉGRALE DE CHOQUET*

3.2.1 Définitions

Étant donnée une mesure floue μ, l'intégrale de Choquet C_μ sur un vecteur de critères $a = (a_1, a_2, \ldots, a_{n_c})$, est définie par :

$$C_\mu(a) = \sum_{i=1}^{n_c} \left(a_{\sigma(i)} - a_{\sigma(i-1)}\right).\mu\left(\sigma(i), \ldots, \sigma(n_c)\right) \tag{5.4}$$

tels que $a_{\sigma(0)} = 0 \leq a_{\sigma(1)} \leq \cdots \leq a_{\sigma(n_c)}$ représentent les indices permutés des critères.

Par exemple si :

$a = (a_1, a_2, a_3) = (0.9, 0.8, 0.5)$, alors $a_0 = 0 \leq a_{\sigma(1)=3} = 0.5 \leq a_{\sigma(2)=2} = 0.8 \leq a_{\sigma(3)=1} = 0.9$

par conséquent :

$$C_\mu(0.9, 0.8, 0.5) = \left(a_{\sigma(1)} - a_{\sigma(0)}\right).\mu\left(\sigma(1), \sigma(2), \sigma(3)\right) + \left(a_{\sigma(2)} - a_{\sigma(1)}\right).\mu\left(\sigma(2), \sigma(3)\right) + \left(a_{\sigma(3)} - a_{\sigma(2)}\right).\mu\left(\sigma(3)\right)$$

soit encore :

$$C_\mu(0.9, 0.8, 0.5) = 0.5 \times \mu(3,2,1) + (0.8 - 0.5) \times \mu(2,1) + (0.9 - 0.8) \times \mu(1).$$

La plupart des méthodes existantes en agrégation multicritère se basent sur la somme pondérée, qui met en valeur l'importance de chaque critère indépendamment. A la différence, l'intégrale de Choquet se base sur la notion d'interaction pour la résolution des problèmes multicritères, c'est-à-dire tenir compte de l'importance de chaque critère mais aussi de l'importance relative entre ces derniers. Pour cela, nous distinguons l'importance globale de chaque critère et l'importance relative due à l'interaction entre ces critères. Outre les propriétés usuelles des opérateurs d'agrégation et la modélisation de l'importance relative des critères, la famille de l'intégrale de Choquet a la distinction de permettre la représentation de phénomènes d'interaction mutuelle qui peuvent exister.

3.2.2 Importance relative des critères : indices de Shapley

L'importance globale d'un critère i n'est pas déterminée uniquement par la mesure floue $\mu(i)$, mais elle prend en compte toutes les mesures $\mu(D)$ pour un sous-ensemble $D \subset N_C$ de toutes les coalitions D pour $i \in D$. En effet, nous pouvons avoir $\mu(i)$ quasiment nulle suggérant que le critère *i* est sans importance. Cependant, nous pouvons avoir en joignant *i* à une coalition $D \subset N_C$, une valeur de $\mu(D \cup \{i\})$ qui soit plus grande que $\mu(D)$ suggérant ainsi l'importance du critère *i* dans la décision. Le calcul de l'importance globale se base ainsi sur la notion d'indice de Shapley, issue de la théorie des jeux coopératifs (Grabisch, 1996). Pour tout critère *i* l'indice de Shapley est défini par :

$$I_i = \sum_{D \subset N_c \setminus \{i\}} \frac{(n_c - |D| - 2)!|D|!}{(n_c - 1)!} . \left(\mu(D \cup \{i\}) - \mu(D)\right) \qquad (5.5)$$

où $|D|$ représente le cardinal de D.

L'indice de Shapley est représenté par le vecteur $\left[I_1, I_2, \cdots, I_{n_c}\right]$. Cet indice calcule la contribution moyenne du critère *i* dans toutes les coalitions. Une propriété fondamentale de l'indice de Shapley est que $\sum_{i=1}^{n_c} I_i = 1$.

3.2.3 L'indice d'interaction entre les critères

L'indice d'interaction entre les critères *i* et *j* est la moyenne de la quantité de synergie entre *i* et *j* en présence d'un groupe de critères D. Il est défini par la relation (5.6).

$$I_{ij} = \sum_{D \subset N_c \setminus \{i\}} \frac{(n_c - |D| - 2)!|D|!}{(n_c - 1)!} \left(\mu(D \cup \{i, j\}) - \mu(D \cup \{i\}) - \mu(D \cup \{j\}) + \mu(D)\right) \qquad (5.6)$$

Trois situations peuvent être considérées selon l'importance des critères *i* et *j* pris ensemble :

- Si $\mu(\{i, j\}) > \mu(\{i\}) + \mu(\{j\})$ alors il y a une synergie de complémentarité entre ces deux critères.
- Si $\mu(\{i, j\}) < \mu(\{i\}) + \mu(\{j\})$ alors il y a une redondance ou une synergie négative entre ces deux critères.
- Si $\mu(\{i, j\}) = \mu(\{i\}) + \mu(\{j\})$ alors les critères sont indépendants.

En se basant sur la mesure 2-additive (Grabisch, 1997) (qui nécessite $n_c \times (n_c + 1)/2$ coefficients représentés par les singletons et les paires de critères), l'intégrale de Choquet peut s'exprimer en fonction des indices de Shapley et d'interaction comme suit :

$$C_\mu(a) = \sum_{I_{i,j} \succ 0} (a_i \wedge a_j) I_{ij} + \sum_{I_{i,j} \prec 0} (a_i \vee a_j) \left| I_{ij} \right| + \sum_{i=1}^{n_c} a_i (I_i - \frac{1}{2} \sum_{i \neq j} \left| I_{ij} \right|) \qquad (5.7)$$

$\wedge$ et $\vee$ représentent respectivement les fonctions min et max.

Avec $I_i - \frac{1}{2} \sum_{i \neq j} \left| I_{ij} \right| \geq 0$ et $\sum_{i=1}^{n_c} I_i = 1.$

L'équation se décompose en une partie conjonctive, disjonctive et une partie additive correspondant respectivement aux synergies positives, négatives et nulles. Une valeur de :

- I_{ij} positive implique un comportement conjonctif entre les critères *i* et *j*. Ceci indique que la satisfaction simultanée de ces critères est significative dans l'évaluation globale, et que la satisfaction d'un seul des critères aura peu d'effet. Il s'agit d'une synergie positive entre les deux critères.
- I_{ij} négative implique un comportement disjonctif entre les critères *i* et *j*. Ainsi, la satisfaction de l'un des critères est suffisante pour avoir un effet significatif dans l'évaluation globale. Il s'agit d'une synergie négative entre les deux critères.
- I_{ij} nulle exprimant le fait que les deux critères sont indépendants et les valeurs de Shapley agissent en tant que vecteur de poids dans une moyenne pondérée arithmétique. Ceci constitue la partie linéaire de l'intégrale de Choquet.

En outre, le choix du décideur peut être modélisé par la matrice d'interaction permettant de représenter les poids des différents critères ainsi que leurs interactions comme le montre la relation (5.8).

$$I = \begin{pmatrix} I_1 & I_{12} & I_{13} & \cdots & I_{1n_c} \\ I_{12} & I_2 & I_{23} & \cdots & I_{2n_c} \\ I_{13} & I_{23} & I_3 & \cdots & I_{3n_c} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ I_{1n_c} & I_{2n_c} & I_{3n_c} & \cdots & I_{n_c} \end{pmatrix} \qquad (5.8)$$

3.3 *EXEMPLE D'APPLICATION*

Cet exemple a été exposé dans (Grabisch, 1996). Il s'agit d'un exemple dans lequel le directeur d'une école d'ingénieurs souhaite évaluer ses étudiants. L'évaluation s'appuie sur les notes obtenues en mathématique (M), physique (P) et littérature (L). L'ensemble des alternatives est donc constitué des étudiants, et les critères sont les notes obtenues dans les trois matières. Voici les notes obtenues par trois étudiants A, B et C.

Tableau 5.1 – Données de l'exemple illustratif.

		Alternatives		
		A	B	C
Critères	M	18	10	14
	P	16	12	15
	L	10	18	15

Si l'on considère la moyenne pondérée, c'est-à-dire la combinaison linéaire des critères $a=\left(a_1,a_2,\cdots,a_{n_c}\right)$, qui est définie par la relation suivante :

$$h_\omega(a)=\sum_{i=1}^{n_c}\omega_i\times a_i$$

où $\omega=\left[\omega_1,\omega_2,\cdots,\omega_{n_c}\right]$ est le vecteur des poids $\omega_i\in[0,1]$ associé à chaque critère *i* et $\sum_{i=1}^{n_c}\omega_i=1$ (pour la normalisation).

Le directeur, établit une moyenne pondérée en se basant sur l'orientation scientifique de l'école. Il attribue donc les poids suivants : $\omega_M=\omega_P=3/8$ et $\omega_L=2/8.$ On obtient alors $h_\omega(A)=15.25,\ h_\omega(B)=12.75,$ et $h_\omega(C)=14.625.$ D'après cette moyenne, nous remarquons que les étudiants sont classés de cette façon $A\succ C\succ B$. L'étudiant A est classé premier bien qu'il a la plus petite note en littérature. A partir de cette structure de préférence, on en déduit que des mauvaises notes peuvent se compenser par des bonnes. En effet, le fait d'attribuer une importance aux mathématiques et à la physique au détriment de la littérature permet de surestimer, par la moyenne pondérée, les étudiants bons en matières scientifiques.

Le directeur n'est pas satisfait du résultat, car d'après lui, l'étudiant C est bon en sciences comme en littérature, et est meilleur que l'étudiant A, qui est excellent en mathématique et

physique mais mauvais en littérature. Les nouvelles préférences du directeur tiennent donc compte du fait que :

- les matières scientifiques (M, P) sont les plus importantes ;
- les étudiants bons en mathématique sont en général bons en physique et vise versa. Par conséquent il ne faut pas les favoriser ;
- les étudiants bons en matières scientifiques et littéraires doivent être favorisés.

Ainsi l'operateur d'agrégation doit considérer l'importance de chaque critère pris à part, mais il doit aussi prendre en compte l'interaction entre eux. Pour ce faire, le directeur décide d'utiliser une intégrale de Choquet en attribuant les mesures floues suivantes pour exprimer ces préférences (décrites ci-dessus) :

- $\mu(M)=\mu(P)=0.45$ et $\mu(L)=0.3$ qui représentent les importances relatives des critères et donnant plus d'importance aux matières scientifiques ;
- $\mu(M,P)=0.5<\mu(M)+\mu(P)$ car M et P se renforcent (redondances) ; une telle valeur permet de ne pas trop les favoriser ;
- $\mu(M,L)=\mu(P,L)=0.9>\mu(M)+\mu(L)$ pour favoriser les étudiants bons en M et L et ceux en P et L (complémentarité) ;

Les résultats donnent :

- pour l'étudiant A, sachant que ses notes sont classées dans l'ordre croissant suivant $L\le P\le M$:

$$C_\mu(A)=(10-0)\cdot\mu(\{M,P,L\})+(16-10).\mu(\{M,P\})+(18-16).\mu(\{M\})$$

$$C_\mu(A)=10+6\times0.5+2\times0.45=13.9$$

- pour l'étudiant B, sachant que ses notes sont classées dans l'ordre croissant suivant $M\le P\le L$:

$$C_\mu(B)=(10-0)\cdot\mu(\{M,P,L\})+(12-10).\mu(\{L,P\})+(18-12).\mu(\{L\})$$

$$C_\mu(B)=10+2\times0.9+6\times0.3=13.6$$

- pour l'étudiant C, sachant que ses notes sont classées dans l'ordre croissant suivant $M\le P\le L$:

$$C_\mu(C) = (14-0)\cdot\mu(\{M,P,L\}) + (15-14).\mu(\{L,P\}) + (15-15).\mu(\{L\})$$

$$C_\mu(B) = 14 + 1\times 0.9 + 0 = 14.9$$

Ainsi nous obtenons le classement $C \succ A \succ B$.

4 PRÉSENTATION DE L'APPROCHE DÉVELOPPÉE

Nous proposons pour la résolution du problème FSH multicritère une approche par hybridation d'un algorithme ACO permettant de générer une multitude de solutions possibles et d'une approche floue considérant l'agrégation multicritère. Cette dernière vise à évaluer la qualité des solutions en utilisant l'intégrale de Choquet permettant ainsi d'aider le décideur lorsqu'il ne peut pas donner une préférence particulière (Grabisch, 1996).

4.1 *DESCRIPTION DE NOTRE APPROCHE ACO MULTICRITÈRE*

Dans la littérature, différentes approches ACO sont introduites pour la résolution de problèmes d'optimisation multicritère. Les points, qui différencient ses méthodes (Alaya et al., 2007), concernent essentiellement :

- le nombre de colonie de fourmis : ainsi une colonie ou plusieurs colonies peuvent être considérée(s) pour traiter les différents critères ;
- la modélisation de la phéromone pour laquelle deux structures sont possibles : une structure unique pour l'ensemble des critères, ou plusieurs structures de phéromones associées à chaque critère ;
- le mécanisme de mise à jour des quantités de phéromones ;
- la définition de la visibilité en proposant une ou plusieurs heuristique(s) d'information.

Pour la résolution d'un problème d'optimisation multicritère, Alaya et al.(2007) ont proposé quatre variantes d'algorithmes ACO avec une ou plusieurs colonies de fourmis et ont prouvé l'efficacité de celles utilisant une seule colonie. De même, nous développons, pour le problème considéré dans ce chapitre, un algorithme ACO utilisant une seule colonie de fourmis pour mettre en évidence les différents critères simultanément. Dans ce qui suit, une description détaillée des différentes étapes de notre approche est présentée.

4.1.1 Procédure de construction d'une solution

La stratégie de recherche de notre approche ACO prend en considération la nature du problème qui est multicritère. Pour la construction d'une solution, deux modules sont introduits : la désirabilité via la phéromone et l'heuristique d'information. La modélisation de ces modules pour les différents critères est réalisée comme suit :

4.1.1.a Modélisation de la phéromone

La phéromone est structurée de manière à donner l'intérêt à placer une opération suite à une autre sur une même machine. Cependant, le fait d'avoir plusieurs critères à prendre en considération simultanément a une influence sur l'évaluation de la solution. Ainsi, nous proposons l'attribution d'une structure de phéromones commune pour les différents critères qui pendant le mécanisme de mise à jour des quantités de phéromones sera modifiée de manière à donner de l'importance aux meilleures solutions pour chaque critère indépendamment. Initialement une petite valeur τ_0 est attribuée à chaque paire d'opérations.

4.1.1.b Modélisation de la visibilité

A chaque critère considéré une heuristique de visibilité est introduite. Ainsi, nous adoptons une première heuristique pour le makespan, par l'utilisation de la règle SPT comme le montre la relation suivante :

$$\eta_{f_1}\left(O_{ij},O_{hl}\right)=\frac{1}{p_{hl}} \tag{5.9}$$

Une deuxième heuristique qui prend en compte les coûts de pénalité d'avance et de retard est aussi proposée pour guider les fourmis vers des solutions meilleures pour ce critère. La description de cet heuristique a déjà été donnée dans le chapitre précédent (voir chapitre 4 §5.1.2.b). Nous rappelons dans ce qui suit la structure de cette visibilité :

$$\eta_{f_2}\left(O_{ij},O_{hl}\right)=\begin{cases}W_l, & \text{si } s_l \le 0\\ W_l-\left(s_l\times CT\right), & \text{si } 0\le s_l \le 1/CT\\ H_l, & \text{sinon}\end{cases} \tag{5.10}$$

avec :

- $W_l=\dfrac{\omega_l^T}{p_{hl}}$;

- $H_l = \dfrac{\omega_l^E}{p_{hl}}$;
- $s_l = d_l - t - p_{hl}$ représente la laxité du travail l à l'instant t ;
- $CT = \dfrac{(H_l + W_l)}{max\{0, \bar{p} - p_{hl}\}}$, $\bar{p}$ étant la moyenne des durées d'exécution de tous les travaux.

4.1.1.c Règle de transition

Une fourmi *k* positionnée sur un nœud O_{ij} va à un instant donné, choisir, à partir d'une liste d'opérations candidates S_k, l'opération suivante *y* en utilisant la règle de transition suivante :

$$y = \begin{cases} \arg\max_{O_{hl} \in S_k} \left\{ \left[\tau(O_{ij}, O_{hl}) \right] \cdot \left(\left[\eta_{f_1}(O_{ij}, O_{hl}) \right]^\beta + \left[\eta_{f_2}(O_{ij}, O_{hl}) \right]^\beta \right) \right\}, & \text{si} \quad q \le q_0 \\ Y, & \text{si} \quad q > q_0 \end{cases} \tag{5.11}$$

Les même paramètres β , q_0 et q sont repris de L'ACS standard. *Y* est une opération sélectionnée aléatoirement selon la règle :

$$p_k(O_{ij}, Y) = \frac{\tau(O_{ij}, O_{hl}) \cdot \left(\left[\eta_{f_1}(O_{ij}, O_{hl}) \right]^\beta + \left[\eta_{f_2}(O_{ij}, O_{hl}) \right]^\beta \right)}{\sum_{O_{hl} \in S_k} \tau(O_{ij}, O_{hl}) \cdot \left(\left[\eta_{f_1}(O_{ij}, O_{hl}) \right]^\beta + \left[\eta_{f_2}(O_{ij}, O_{hl}) \right]^\beta \right)} \tag{5.12}$$

La sommation introduite dans les deux relations (5.11) et (5.12) permet l'évaluation de chaque critère pour la sélection de la prochaine opération. Une même importance est attribuée à chaque critère.

4.1.1.d Mécanisme de mise à jour

Une mise à jour locale est effectuée par chaque fourmi selon la relation (5.13) permettant ainsi de diminuer la quantité de phéromones à chaque transition.

$$\tau(O_{ij}, O_{hl}) = (1 - \rho_\ell).\tau(O_{ij}, O_{hl}) + \rho_\ell.\tau_0 \tag{5.13}$$

La mise à jour globale est faite grâce à la relation (5.14), la meilleure solution générée pour chaque critère considéré indépendamment est mise à jour.

$$\tau(O_{ij}, O_{hl}) = (1 - \rho_g).\tau(O_{ij}, O_{hl}) + \rho_g.\Delta\tau(O_{ij}, O_{hl}) \tag{5.14}$$

avec

$$\Delta\tau(O_{ij},O_{hl})=\begin{cases}1/C_{gb}, & \text{si } (O_{ij},O_{hl}) \in \text{à la meilleure solution trouvée pour le } C_{max}\\ 1/ET_{gb}, & \text{si } (O_{ij},O_{hl}) \in \text{à la meilleure solution trouvée pour le ET}\\ 0, & \text{sinon}\end{cases}$$

ρ_g ($0 \le \rho_g \le 1$) représente le coefficient d'évaporation de phéromones, C_{gb} est le résultat de la meilleure solution trouvée pour le makespan et ET_{gb} est le résultat de la meilleure solution trouvée pour ET.

4.2 *APPROCHE D'ÉVALUATION MULTICRITÈRE : INTÉGRALE DE CHOQUET*

Pour évaluer les solutions obtenues par notre algorithme de fourmis et selon les critères considérés, le concept d'agrégation par les intégrales de Choquet est utilisé. Dans la plupart des problèmes multicritères, le problème d'ordre de grandeur des critères est posé. Par exemple, on peut avoir une solution où $f_1 = 1000$ et $f_2 = 10$. Dans ce cas, f_2 sera toujours dominé par f_1. De plus, il est difficile d'estimer à priori l'ordre de grandeur de chaque critère. Pour éviter que certains critères ne soient toujours dominés par d'autres, il est primordiale d'homogénéiser (ou encore normaliser) les différents critères. Ainsi, pour chaque critère, une fonction d'appartenance floue est utilisée (Zadeh, 1965). Donc, nous attribuons pour chaque critère *i* :

- Une valeur minimale $f_i^{\min}$. Pour déterminer une valeur minimale à la date d'achèvement maximale de tous les travaux $C_{\max}$, nous utilisons la borne inférieure proposée dans (Santos et al., 1995). Pour la somme pondérée des avances/retards de tous les travaux, il est difficile de construire une borne inférieure raisonnable pour le problème FSH. L'idéal pour un tel problème est de terminer tous les travaux juste à temps c'est-à-dire à la date d_j (Saad et al., 2008), ainsi nous supposons que $f_2^{\min}$ vaut zéro.
- Une valeur maximale $f_i^{\max}$. A partir des solutions trouvées par les heuristiques constructives utilisant les règles de priorité EDD, ODD, DD/BJ, TPT&DD/BJ pour chaque critè*re i*, nous déterminons une valeur maximale donnée par calcul de la valeur moyenne de ces solutions.

La méthode d'évaluation floue que nous adaptons associe à chaque solution réalisable *x* un vecteur :

$$f(x) = \left(f_1(x), f_2(x), \ldots, f_{n_c}(x) \right)^T \tag{5.15}$$

où $f(x) \in [\, f_1^{\min}(x), +\infty[\times \ldots \times [\, f_{n_c}^{\min}(x), +\infty[$, T représente la transposée d'un vecteur.

Pour chaque vecteur $f(x)$, nous proposons une fuzzification de ces composants $f_i(x)$ selon leurs positions dans les intervalles $\left[f_i^{\min}, f_i^{\max} + \varepsilon_i \right]$, tel que ε_i est une petite valeur positive conçue pour éviter un problème de division par zéro lorsque $f_i^{\min} = f_i^{\max}$. Elle est formulée comme suit :

$$\varepsilon_i = \begin{cases} 0.01 \times f_i^{\max}, & \text{si } f_i^{\min} = f_i^{\max} \\ 0, & \text{sinon} \end{cases} \tag{5.16}$$

Cette fuzzification est appliquée en utilisant la fonction d'appartenance décrite par la Figure 5.1.

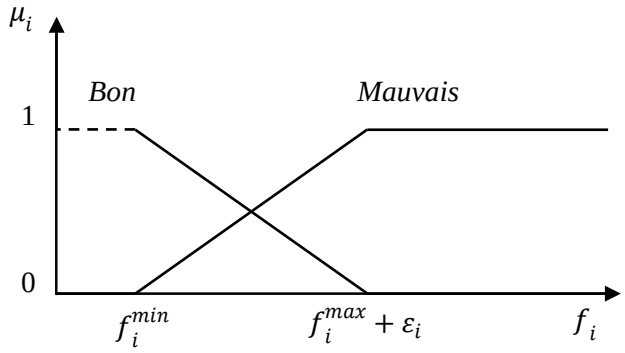

Figure 5.1 – Application floue dans la résolution du problème d'échelle

Deux sous-ensembles flous B^i et M^i sont considérés. Le premier regroupe les bonnes solutions selon la $i^{ième}$ fonction objectif et le second regroupe les mauvaises solutions selon la $i^{ième}$ fonction objectif. Les mesures floues de ces sous-ensembles sont données par les relations suivantes :

$$\mu_i^B(f_i(x)) = \begin{cases} 0, & \text{si } f_i(x) \geq f_i^{max} + \varepsilon_i \\ \dfrac{f_i^{max} - f_i(x) + \varepsilon_i}{f_i^{max} - f_i^{min} + \varepsilon_i}, & \text{si } f_i(x) \in [f_i^{min}, f_i^{max} + \varepsilon_i[\\ 1, & \text{si } f_i(x) \leq f_i^{min} \end{cases} \quad (5.17)$$

$$\mu_i^M(f_i(x)) = 1 - \mu_i^B(f_i(x))$$

La qualité de chaque solution *x* est caractérisée par le vecteur $f_B(x)$ dont toutes les composantes sont homogènes puisqu'elles appartiennent toutes au même intervalle $[0,1]$. Ce vecteur est défini par :

$$f_B(x) = (a_1, a_2, \ldots, a_{n_c})^T \quad (5.18)$$

avec $a_i = \mu_i^B(f_i(x)), \forall i = 1, \ldots, n_c$.

Dans ce cas, les préférences du décideur peuvent s'exprimer par des variables linguistiques.

4.3 *EXEMPLE D'APPLICATION*

Soit l'exemple d'un atelier où trois travaux doivent se traiter sur trois étages. Le nombre de machines sur chaque étage est respectivement $m_1 = 1, m_2 = 2$ et $m_3 = 2$. Les durées opératoires sont comme suit $p_{11} = 4$, $p_{21} = 6$, $p_{31} = 4$, $p_{12} = 2$, $p_{22} = 8$, $p_{32} = 2$, $p_{13} = 10$, $p_{23} = 4$ et $p_{33} = 4$. les dates d'échéances sont $d_1 = 23$, $d_2 = 20$, $d_3 = 22$. En outre, nous supposons que $\omega_j^E = 0.5$ et $\omega_j^T = 1$ pour chaque travail *j*. Les préférences du décideur sont données par la matrice d'interaction ci-dessous indiquant l'importance du critère f_1 par rapport à f_2.

$$I = \begin{pmatrix} 0.6 & -0.1 \\ -0.1 & 0.4 \end{pmatrix}$$

Tableau 5.2 – Valeurs des solutions obtenues par l'ACO pour chaque critère

	f_1	f_2.
e	24	6.5
f	20	4.5
g	22	6.5
h	26	4

Quatre solutions (*e*, *f*, *g* et *h*) pour cet exemple sont considérées. Nous reportons dans le

Tableau 5.2, les valeurs des solutions obtenues pour chaque critère. Une homogénéisation des critères est ensuite appliquée en utilisant les mesures floues du sous-ensemble des bonnes solutions B^i pour chaque critère *i* (ces mesures sont données par la relation $a_i = \mu_i^B\left(f_i(x)\right)$) pour pouvoir comparer les solutions obtenues selon les préférences du décideur. Les valeurs obtenues ainsi que les scores C_μ calculés pour chaque solution sont données dans le Tableau 5.3.

Tableau 5.3 – Valeurs homogénéisées des critères et les scores obtenus pour chaque solution

	a_1	a_2	C_μ
e	0.21	0.23	0.25
f	1	0.46	0.91
g	0.6	0.23	0.51
h	0	0.52	0.29

A partir des scores obtenus, la solution *f* est la meilleure par rapport aux autres solutions avec un score égal à 0.91.

5 SIMULATIONS ET RÉSULTATS

Dans cette section, nous présentons une série de tests expérimentaux pour évaluer notre approche pour l'ordonnancement multicritère. Ces tests ont été conduits sur un ensemble d'instances générées aléatoirement. Chaque instance est représentée par le nombre de travaux ainsi que le nombre d'étages $(n \times E)$:

- (10×5) et (10×10)
- (15×5) et (15×10)
- (25×5) et (25×10)

Le jeu de paramètres utilisés pour générer les instances sont comme suit :

- le nombre de machines par étage m_i est choisi de manière aléatoire entre $[1,5]$ et tel qu'au moins un étage soit composé de $m_i > 1$;
- les durées opératoires p_{ij} sont générées aléatoirement selon une loi de distribution

discrète uniforme comprise entre 3 et 25 ;

- la date d'échéance pour chaque travail est définie en utilisant l'équation :

$$d_j = (1+c) \times \sum_{i=1}^{E} p_{ij} \tag{5.19}$$

avec $c \in [0,1]$;

- le poids associé à tous les travaux en avance ω_j^E est égale à 0.5 ;
- le poids associé à tous les travaux en retard ω_j^T est égale à 1.

De même, pour utiliser notre algorithme de fourmis, des expérimentations préliminaires ont été conduites sur quelques instances pour trouver les meilleurs paramètres. Ces résultats sont présentés dans le Tableau 5.4.

Tableau 5.4 – Valeurs des paramètres utilisées pour l'algorithme de fourmis

Paramètres	**Plages des valeurs**	***Meilleure valeur***
$nbAnt$	−−	20
$t_{\max}$	−−	500
τ_0	−−	0.01
β	$0 \le \beta \le 10$	2
q_0	$0 \le q_0 \le 1$	0.8
ρ_ℓ	$0 \le \rho_\ell \le 1$	0.05
ρ_g	$0 \le \rho_g \le 1$	0.05

Pour chaque instance, 30 réplications sont réalisées. La matrice d'interaction est donnée par :

$$I = \begin{pmatrix} 0.6 & -0.3 \\ -0.3 & 0.4 \end{pmatrix}$$

Cette matrice d'interaction, est un exemple d'application pour l'intégrale de Choquet, qui privilégie le makespan au détriment de la somme pondérée associée à l'avance/retard.

Le Tableau 5.5 présente les résultats donnés par notre méthode multicritère. Rappelons que f_1 et f_2 représentent respectivement les valeurs des solutions pour les critères $C_{\max}$ et la somme pondérée des avances/retards. Les meilleurs scores C_μ et C_μ^{heur} des intégrales de

Choquet trouvées respectivement par notre méthode multicritère et par les heuristiques constructives EDD, ODD, DD/BJ, TPT&DD/BJ sont aussi exposés.

Tableau 5.5 – Résultats expérimentaux

n	E	$C_{\max}^{LB}$	f_1	f_2	C_μ	C_μ^{Heur}	CPU(s)
10	5	113	113	198.5	0.921	0.438	2.39
	5	171	173	315	0.870	0.608	2.33
	5	151	151	180	0.965	0.450	2.34
	5	151	156	308	0.735	0.719	2.42
	5	170	182	184.5	0.721	0.566	2.57
Moyenne		**160.75**	**165.5**	**246.9**	**0.823**	**0.588**	**2.42**
10	10	223	247	275	0.699	0.598	10.30
	10	242	263	417	0.754	0.598	9.98
	10	171	228	162	0.681	0.669	10.24
	10	250	250	347.5	0.970	0.765	10.31
	10	224	232	247.5	0.864	0.689	10.17
Moyenne		**221.75**	**243.3**	**293.5**	**0.817**	**0.680**	**10.18**
15	5	217	218	635	0.938	0.534	6.44
	5	236	240	691	0.851	0.652	6.44
	5	117	132	278	0.753	0.669	6.53
	5	252	257	938	0.894	0.668	6.43
	5	263	263	872	0.973	0.521	6.90
Moyenne		**217**	**223**	**694.8**	**0.868**	**0.628**	**6.58**
15	10	302	315	661	0.755	0.502	30.61
	10	284	296	790.5	0.755	0.690	31.18
	10	288	306	532.5	0.813	0.664	30.39
	10	330	346	962	0.677	0.564	29.67
	10	256	286	638	0.772	0.668	29.44
Moyenne		**289.5**	**308.5**	**730.8**	**0.754**	**0.647**	**30.17**
25	5	230	230	1592.5	0.873	0.820	22.75
	5	370	381	2693	0.687	0.553	23.01
	5	181	207	892.5	0.714	0.713	24.68
	5	408	408	3972	0.954	0.951	23.60
	5	225	231	1334	0.871	0.832	23.18
Moyenne		**296**	**306.8**	**2222.9**	**0.807**	**0.762**	**23.62**
	10	465	485	3126.5	0.579	0.428	122.13
	10	424	473	2796	0.666	0.625	123.36
	10	217	306	725	0.794	0.678	120.60
	10	446	457	3758	0.819	0.646	135.48
	10	421	443	3008.5	0.638	0.313	121.52
Moyenne		**377**	**419.8**	**2571.9**	**0.729**	**0.566**	**125.24**

Nous constatons que les solutions trouvées sont satisfaisantes pour chacun des critères considérés. Le calcul des scores est influencé par les valeurs de $f_i^{\min}$ et $f_i^{\max}$, d'où l'importance du choix judicieux de ces bornes pour l'obtention de bonnes solutions. Le fait d'avoir associé un algorithme de fourmis avec l'intégrale de Choquet nous a permis de

construire des solutions de bonnes qualités avec des temps de calcul raisonnables en comparaison avec les heuristiques de constructions. En effet, l'algorithme de fourmis fournit une multitude de solutions acceptables et l'intégrale de Choquet a pour rôle de mettre en évidence les différentes interactions entre les critères considérées contribuant ainsi à retrouver les meilleures solutions. Notons que les valeurs trouvées pour le makespan des instances testées sont proches des bornes inférieures de (Santos et al., 1995) (noté $C_{\max}^{LB}$ dans Tableau 5.5). Ceci prouve que notre méthode tient compte des choix du décideur (représentés par la matrice d'interaction). Si le score C_{μ} est proche de 1, nous avons une satisfaction de la stratégie du décideur.

6 CONCLUSION

Dans ce chapitre nous avons étudié un problème d'ordonnancement de type flow-shop hybride minimisant le makespan et la pénalité d'avance et du retard. Afin de résoudre ce problème, nous avons proposé une méthode approchée hybride utilisant l'optimisation par colonies de fourmis et la logique floue. Des tests ont été menés sur des instances que nous avons générées aléatoirement pour valider notre méthode. Les temps d'exécution sont de l'ordre de quelques secondes. Les résultats obtenus prouvent l'efficacité de notre algorithme de fourmis et l'importance de la méthode d'agrégation utilisée. Cette dernière permet de mettre en évidence les interactions entre les critères.

CONCLUSION GÉNÉRALE

Le travail que nous venons de présenter nous a permis de tester un ensemble de méta-heuristiques à base d'optimisation par colonie de fourmis pour la résolution de problèmes d'ordonnancement de type flow-shop hybride, avec machines parallèles identiques sur chaque étage, dans le cas monocritère et multicritère. La recherche bibliographique réalisée a montré :

- l'importance de ce type d'atelier dans le monde industriel,
- une littérature abondante de méthodes exactes et approchées essentiellement pour les aux problèmes ayant pour critère d'optimisation le makespan,
- mais un faible recours aux méthodes de résolution à base d'algorithmes de fourmis pour ce type de problème en comparaison avec d'autres méthodes de résolution comme les algorithmes génétique, la recherche tabou ou le recuit simulé…

Notre contribution s'est focalisée sur l'étude de quatre modèles théoriques du problème d'ordonnancement de type flow-shop hybride en considérant la minimisation des critères suivants :

- le makespan $C_{\max}$,
- la somme pondérée de l'avance/retard de tous les travaux $\sum\left(\omega_j^E E_j + \omega_j^T T_j\right)$,
- la somme totale de l'avance/retard de tous les travaux $\sum\left(E_j + T_j\right)$,
- le makespan $C_{\max}$ et la somme pondérée de l'avance/retard de tous les travaux $\sum\left(\omega_j^E E_j + \omega_j^T T_j\right)$ simultanément.

D'un point de vue méthodologique, nous nous sommes intéressés aux approches à bases d'algorithmes de fourmis pour résoudre les problèmes considérés. Notre objectif étant d'explorer et de justifier l'utilisation de ce type d'approches pour ce problème, motivé par le fait que ces dernières ont permis de résoudre avec succès un certain nombre de problèmes

d'ordonnancement. De même, nous nous somme intéressés à l'étude d'un cas multicritère en intégrant un module d'aide à la décision par l'application de la logique floue.

Notre première contribution dans cette thèse est le développement de trois algorithmes à base de colonie de fourmis pour le $FHE,\left(Pm_i\right)_{i=1}^{E}\left|\right|C_{\max}$. Ces approches se distinguent les unes des autres par la conception et la manière d'aborder les décisions en rapport avec l'affectation et le séquencement. Nous avons pu tester différentes variétés de ces algorithmes en proposant différents modules pour la visibilité. D'après les résultats que nous avons obtenus sur des instances issues de la littérature, il en résulte que nos méthodes ont de bonnes performances, notamment *IOMACS-FSH* en les comparants avec ceux obtenues à partir d'autres approches telles que le B&B, le AIS, le GA et un algorithme de fourmis. Ceci prouve l'intérêt d'utiliser les méthodes d'optimisation à base de fourmis pour la résolution des problèmes flow-shop hybride.

Notre second apport a consisté en l'étude de problèmes d'ordonnancement juste à temps. Deux approches à base d'algorithmes de colonie de fourmis ont été respectivement proposées pour la résolution de $FHE,\left(Pm_i\right)_{i=1}^{E}\left|\right|\sum\left(\omega_j^E E_j+\omega_j^T T_j\right)$ et de $FHE,\left(Pm_i\right)_{i=1}^{E}\left|\right|\sum\left(E_j+T_j\right)$. La conception des approches proposées s'est fondée sur celles que nous avons développées précédemment pour le problème avec la minimisation du makespan, en considérant l'avance et le retard des travaux dans le procédé de construction des solutions. Une série de tests a été effectuée sur des instances que nous avons générées de manière aléatoire. Les résultats trouvés ont été comparés avec ceux obtenus en adaptant des heuristiques constructives issues de la littérature qui tiennent compte des dates d'échéance. Ces résultats ont montré que les approches à base de colonie de fourmis sont meilleures dans ce cas d'étude. Ainsi, une amélioration allant jusqu'à 27.31% a été obtenue pour le problème $FHE,\left(Pm_i\right)_{i=1}^{E}\left|\right|\sum\left(E_j+T_j\right)$. De même, nous avons constaté que l'utilisation de la règle TPT&DD/BJ comme heuristique de visibilité permet d'obtenir de meilleures solutions.

Notre troisième contribution se manifeste par l'étude d'un problème d'ordonnancement multicritère. Pour ce faire, nous avons, là encore, utilisé une approche d'optimisation par colonie de fourmis. Le choix d'une telle approche se justifie par l'avantage de conduire en fin de recherche à une population diversifiée de solutions. Dans le cadre de ce travail, l'approche, que nous avons considérée, transforme le problème multicritère en un problème monocritère par l'ajout d'un module d'aide à la décision. Ce module utilise une intégrale de Choquet

comme opérateur d'agrégation pour l'évaluation des solutions. Une telle méthode d'agrégation permet de prendre en compte les interactions entre les différents critères. Des tests ont été menés pour valider notre approche sur des instances générées aléatoirement. Les solutions obtenues ont démontré leur correspondance avec le choix souhaité du décideur.

Comme perspectives de recherche futures, plusieurs pistes nous semblent intéressantes à explorer. Une première ouverture, concernant le travail effectué au quatrième chapitre, est de développer d'autres méta-heuristiques ou des approches hybrides pour résoudre les problèmes traitant les critères d'ordonnancement juste à temps. Il serait aussi judicieux de tester, pour la résolution de ces problèmes, d'autres modèles de phéromones dans nos algorithmes de fourmis.

Nous nous sommes intéressés à des ordonnancements calés gauche pour résoudre les problèmes d'ordonnancement JAT considérés. Comme il s'agit de critères irréguliers, les solutions obtenues ne sont pas forcément les meilleures, car la somme des avances pourrait être importante. Pour améliorer les solutions (suite aux suggestions et remarques de Madame Portmann) :

- il est possible d'introduire des temps morts entre les opérations. Ainsi, il serait pertinent de proposer des heuristiques qui introduisent des temps morts sur le dernier étage ou qui travaillent sur les dates de début au lieu de travailler simplement sur les ordres. En outre, une autre voie pour remédier à ces ordonnancements (calés gauche) est l'utilisation de stratégies de recherche visant à résoudre les problèmes d'ordonnancement JAT connues sous le nom d'algorithmes de timing (Hendel & Sourd, 2007). Ils consistent à s'intéresser à la résolution du problème quand l'ordre d'exécution des opérations sur chaque machine est donné en introduisant des temps morts.
- une autre perspective pour cette partie serait de revoir la manière de générer les dates d'échéances des travaux. En effet, dans notre thèse, la génération de ces dates ne tient pas compte du nombre de machines en parallèle sur chaque étage. Par conséquent, les contraintes d'urgence des travaux peuvent changer en augmentant ou en diminuant le nombre de machines sur les étages sans que les dates d'échéance ne soient affectées.

Il serait judicieux aussi d'essayer de considérer le flow-shop hybride avec machines non identiques sur chaque étage.

Concernant le cinquième chapitre, une perspective pourrait concerner l'utilisation d'autres techniques de résolution pour les problèmes multicritères, notamment les méthodes dite Pareto.

BIBLIOGRAPHIE

Aghezzaf, E.H., Artiba, A., Moursli, O. & Tahon, C., 1995. Hybrid flowshop problems, a decomposition based heuristic approach. *In Proceedings of International Conference on Industrial Engineering and Production*, pp.43-56.

Alaya, I., Solnon, C. & Ghedira., K., 2007. Ant colony optimization for multi-objective optimization problems. *The 19th IEEE International Conference on Tools with Artificial Intelligence*, 1, pp.450-57.

Alaykýran, K., Engin, O. & Döyen, A., 2007. Using ant colony optimization to solve the hybrid flow shop sceduling problems. *International Journal of Advanced Manufacturing Technology*, 35(5-6), pp.541-50.

Baker, K.R., 1974. *Introduction to Sequencing and Scheduling*. New York: Wiley.

Baker, K.R. & Scudder, G.D., 1990. Sequencing with earliness and tardiness penalties: a review. *Operations Research*, 38, pp.22-36.

Baptiste, P., Flaminib, M. & Sourd, F., 2008. Lagrangian bounds for just-in-time job-shop scheduling. *Computers & Operations Research*, 35, pp.906-15.

Behnamian, J., Fatemi Ghomi, S.M.T. & Zandieh, M., 2010. Development of a hybrid metaheuristic to minimise earliness and tardiness in a hybrid flowshop with sequence-dependent setup times. *International Journal of Production Research*, 48(5), pp.1415-38.

Bellman, R., 1954. The theory of dynamic Programming. *Bulletin of the American Mathematical Society.* , 60(6), pp.503-15.

Ben Hmida, A., Huguet, M.-J., Lopez, P. & Haouari, M., 2007. Climbing depth-bounded discrepancy search for solving hybrid flow shop problems. *European Journal of Industrial Engineering*, 1(2), pp.223-43.

Bertel, S. & Billaut, J.-C., 2004. A genetic algorithm for the industrial multiprocessor flow shop scheduling problem with recirculation. *European Journal of Operational Research*, 159, pp.651-62.

Besbes, W., Loukil, T. & Teghem, J., 2006. Using genetic algorithm in the multiprocessor flow shop to minimize the makespan. *International Conference on Service Systems and Service Management*, 2, pp.1228-33.

Blazewicz, J., Ecker, K.H., Pesch, E. & Schmidt, G., 1996. *Scheduling Computer and Manufacturing Processes*. Berlin: Springer.

Blum, C., 2005. Beam-ACO-hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers & Operations Research*, 32, pp.1565-91.

Blum, C. & Sampels, M., 2002. Ant colony optimization for FOP shop scheduling: a case study on different pheromone representations. *In Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02), IEEE Computer Society Press*, 2, pp.1558-63.

Blum, C. & Sampels, M., 2004. An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms*, 3, pp.285-308.

Bouchon-Meunier, B., 1995. *La logique floue et ses applications*. Paris: Addison-Wesley.

Brah, S.A. & Hunsuker, J.L., 1991. Branch and bound algorithm for the flow shop with multiple processors. *European Journal of Operational Research*, 51, pp.88-99.

Brockmann, K. & Dangelmaier, W., 1998. A parallel branch and bound algorithm for makespan optimal sequencing in flow shops with parallel machines. *In Proceeding of the 2nd International Multiconference on Computational Engineering in Systems Applications (CESA)*, 3, pp.431-36.

Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P. & Schulenburg, S., 2003. Hyper-heuristics: an emerging direction in modern search technology. In F. Glover & G. Kochenberger, eds. *Handbook of Metaheuristics*. Kluwer. pp.457-74.

Campbell, H.G., Dudek, R.A. & Smith, M.L., 1970. A heuristic algorithm for the n job m machine sequencing problem. *Management Science*, 16(10), pp.630-37.

Carlier, J. & Chrétienne, P., 1988. *Problèmes d'ordonnancement : modélisation, complexité, algorithmes*. Paris: Masson.

Carlier, J. & Néron, E., 2000. An exact method for solving the multi-processor flow-shop. *RAIRO-RO*, 34(1), pp.1-25.

Cheng, T.C.E. & Gupta, M.C., 1989. Survey of scheduling research involving due date determination decisions. *European Journal of Operational Research*, 38, pp.156-66.

Chevalier, G., Barrier, J. & Richard, P., 1996. Production planning in the glass industry. *Revue VERRE*, 2(5), pp.27-29.

Chu, C. & Proth, J.-M., 1996. *L'ordonnancement et ses applications*. Paris: Masson.

Colorni, A., Dorigo, M. & Maniezzo, V., 1991a. Distributed optimisation by ant colonies. *Proceedings of ECAL91-First European Conference on Artificial Life*, pp.134-42.

Colorni, A., Dorigo, M. & Maniezzo, V., 1991b. *Positive feedback as a search strategy*. Technical Report 91-016. École polytechnique de Milan (Politecnico di Milano).

Colorni, A., Dorigo, M. & Maniezzo, V., 1992. An investigation of some properties of an ant algorithm. *Proceedings of the Parallel Problem Solving from Nature Conference (PPSN 92)*, pp.509-20.

Colorni, A., Dorigo, M. & Maniezzo, V., 1994. Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science (JORBEL)*, 34(1), pp.39-53.

Dannenbring, D.G., 1977. An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11), pp.1174-82.

Dauzère-Pérès, S. & Paulli, J., 1997. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70, pp.281-306.

Deal, D.E. & Hunsucker, J.L., 1991. The two-stage flowshop scheduling problem with m machines at each stage. *Journal of Information and Optimization Sciences*, 12(3), pp.407-17.

Den Besten, M., Stützle, T. & Dorigo, M., 2000. Ant colony Optimization for the total weighted tardiness problem. *In M. Schoenauer et al., editors, Proceedings of the Sixth International Conference on Parallel Problem solving from Nature(PPS VI), volume 1917 of lecture notes on computer Science*, pp.611-20.

Deneubourg, J.L., Pasteels, J.M. & Verhaeghe, J.C., 1983. Probabilistic behaviour in ants: a strategy of errors? *Journal of Theoretical Biology*, pp.259-71.

Dorigo, M. & Gambardella, L.M., 1997. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1), pp.53-66.

Dorigo, M., Maniezzo, V. & Colorni, A., 1996. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, 26(1), pp.1-13.

Dorigo, M. & Stützle, T., 2004. *Ant Colony optimization*. MIT Press.

Dubois, D. & Prade, H., 1995. La" logique floue" : un outil pour appréhender pratiquement la similarité, les préférences et l'incertitude dans les systèmes d'inférences. *In : Quaderni. Intelligence artificielle et entreprise : l'entreprise intelligente ?*, 25, pp.59-73.

Dubus, J.-P., Gonzales, C. & Perny, P., 2009. Choquet Optimization using GAI Networks for Multiagent/Multicriteria Decision-Making. *In Algorithmic Decision Theory, Lectures Notes in Artificial Intelligence*, pp.377-89.

Duvivier, D., 2000. *Etude de l'hybridation de méta-heuristiques, application au problème d'ordonnancement de type jobshop*. Thèse de Doctorat. Université du Littoral Côte d'Opale, Calais.

Eberhart, R.C. & Kennedy, J., 1995. A new optimizer using particle swarm theory. *In proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp.39-43.

Engin, O. & Döyen, A., 2004. A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Computer Systems*, 20(6), pp.1083-95.

Esquirol, P. & Lopez, P., 1999. *L'ordonnancement*. Paris: Economica.

Fakhrzad, M.B. & Heydari, M., 2008. Production scheduling to minimize the sum of the earliness and tardiness costs. *Journal of the Chinese Institute of Industrial Engineers*, 25(2), pp.105-15.

Feo, T.A. & Resende, M.G.C., 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6, pp.109-33.

Finke, D.A., Medeiros, D.J. & Traband, M.T., 2007. Multiple machine JIT scheduling: a tabu serach approach. *International Journal of Production Research*, 45(21), pp.4899-915.

Fortemps, P., Ost, C., Pirlot, M., Teghem, J. & Tuyttens, D., 1996. Using metaheuristics for solving a production scheduling problem in a chemical firm. *International Journal of Production Economics*, 46-47(1-3), pp.13-26.

Gagné, C., Price, W.L. & Gravel, M., 2002. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence dependent setup times. *Journal of the Operational Research Society*, 53(8), pp.895-906.

Gajpal, Y., Rajendran, C. & Ziegler, H., 2006. An ant colony algorithm for scheduling in flowshops with sequence dependent setup times of jobs. *International Journal of Advanced Manufacturing Technology*, 30(5-6), pp.416-24.

Garey, M. & Johnson, D., 1979. *Computers and Intractability : a guide to the theory of NP completeness*. New York: Freeman.

Ghedjati, F., 1994. *Résolution par des heuristiques dynamiques et des algorithmes génétiques du problème d'ordonnancement de type job-shop généralisé*. Thèse de Doctorat. Paris: Université de Paris 6.

Ghedjati, F. & Portmann, M.C., 2009. Dynamic heuristic for the generalized job-shop scheduling problem. *IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC'09)*, p.2636–2641.

Giard, V., 1988. *Gestion de la production*. Paris: Economica.

Glover, F., 1987. Tabu search methods in artificial intelligence and operations research. *ORSA, Artificial Intelligence Newletter*, 1(2).

GOThA, 1993. Les problèmes d'ordonnancement. *RAIRA-RO*, 27(1), pp.77-150.

Gourgand, M., Grangeon, N. & Norre., S., 2001. Modèle générique orienté objet du flow-fhop hybride hiérarchisé. *3e Conférence francophone de Modélisation et SIMulation « Conception, Analyse et Gestion des Systemes Industriels » (MOSIM'01).*

Gourgand, M., Grangeon, N. & Norre, S., 1999. Metaheuristics for the deterministic hybrid flow shop problem. *In Proceedings of International Conference on Industrial and Production management (IEPM'99)*, pp.136-45.

Grabisch, M., 1996. The application of fuzzy integrals in multicriteria decision-making. *European Journal of Operational Research*, 89, pp.445-56.

Grabisch, M., 1997. k-Ordered Discrete Fuzzy Measures and Their Representation. *Fuzzy sets and systems*, 92, pp.167-89.

Graham, R.L., Lawler, E.L. & Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Discrete Mathematics*, 5, pp.287-326.

Guinet, A., Solomon, M.M., Kedia, P.K. & Dussa, A., 1996. A computational study of heuristics for two-stage flexible flowshops. *International Journal of Production Research*, 34, pp.1399-415.

Gupta, .J.N.D., 1971. A functional heuristic algorithm for the flowshop scheduling problem. *Operational Research Quarterly*, 22(1), pp.39-47.

Gupta, J.N.D., 1988. Two-stage hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, 39, pp.359-64.

Gupta, J.N.D., Hariri, A.M.A. & Potts, C.N., 1997. Scheduling a two-stage hybrid flowshop with parallel machines at the first stage. *Mathematic of industrial Scheduling II (Annals of Operation Research)*, 96, pp.171-91.

Gupta, J.N.D., Krger, K., Lauff, V., Werner, F. & Sotskov, Y.N., 2002. Heuristics for hybrid flow shops with controllable processing times and assignable due dates. *Computers and Operations Research*, 29, pp.1417-39.

Gupta, J.N.D. & Tunc, E.A., 1991. Schedules for a two-stage hybrid flowshop with parallel machines at the second stage. pp.1489-502.

Haouari, M. & Hidri, L., 2008. On the hybrid flowshop scheduling problem. *International Journal of Production Economics*, 113(1), pp.495-497.

Haouari, M., Hidri, L. & Gharbi, A., 2006. Optimal scheduling of a two-stage hybrid flow shop. *Mathematical Methods of Operations Research*, 64(1), pp.107-24.

Haouari, M. & M'Hallah, R., 1997. Heuristic algorithms for the two-stage hybrid flowshop problem. *Operations Research Letters*, 21, pp.43-53.

Hendel, Y., 2005. *Contributions à l'ordonnancement juste-à-temps*. Thèse de Doctorat. Université Pierre et Marie Curie, Paris 6.

Hendel, Y. & Sourd, F., 2007. An improved earliness-tardiness timing algorithm. *Computers & Operations Research*, 34, pp.2931-38.

Hoogeveen, H., 2005. Multicriteria scheduling. *European Journal of Operational Research*, 167, pp.592-623.

Hoogeveen, J.A., Lenstra, J.K. & Veltman., B., 1996. Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. *European Journal of Operational Research*, 89(1), pp.172-75.

Huang, R.-H. & Yang, C.-L., 2008. Ant colony system for job shop scheduling with time windows. *International Journal of Advanced Manufacturing Technology*, 39, pp.151-57.

Janiak, .A., Kozan, E., Lichtenstein, M. & Oguz, C., 2007. Metaheuristic approaches to the hybrid flow shop scheduling problem with a cost-related criterion. *International Journal of Production Economics*, 105, pp.407-24.

Jin, Z.H., Ohno, K., Ito, T. & Elmaghraby, S.E., 2002. Scheduling hybrid flowshops in printed circuit board assembly lines. *Production and Operations Management*, 11(1), pp.216-30.

Jin, Z.H., Yang, Z. & Ito, T., 2006. Metaheuristic algorithms for multistage hybrid flowshop scheduling problem. *International Journal of Production Economics*, 100, pp.322-34.

Johnson, S.M., 1954. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1, pp.61-68.

Jungwattanakit, J., Reodecha, M., Chaovalitwongse, P. & Werner, F., 2009. A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *Computers & Operations Research*, 36(2), pp.358-78.

Khalouli, S., Ghedjati, F. & Hamzaoui, A., 2007. A meta-heuristic algorithm based on ant colony system for the multistage hybrid flow shop scheduling problem. *37th International Conference on Computers and Industrial Engineering (CIE'37)*, pp 2096-2106. Alexandria-Egypt, octobre 2007., pp.2096-106.

Khalouli, S., Ghedjati, F. & Hamzaoui, A., 2008a. Method based on ant colony system for solving the hybrid flow shop scheduling problem. *7th International Conference on Modelling, Optimization and SIMulation Systems (MOSIM'08)*, 2, pp.1407-16.

Khalouli, S., Ghedjati, F. & Hamzaoui, A., 2008b. Ant colony optimization for solving a bi-criteria hybrid flow shop problem. *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp.1440-45.

Khalouli, S., Ghedjati, F. & Hamzaoui, A., 2008c. Hybrid approach based on ant colony optimization and fuzzy logic to solve multi-criteria hybrid flow shop scheduling problem. *Proceedings of the 5th IEEE/ACM International Conference on Soft Computing as Transdisciplinary Science and Technology*, pp.44-50.

Khalouli, S., Ghedjati, F. & Hamzaoui, A., 2009. An integrated ant colony optimization algorithm for the hybrid flow shop scheduling problem. *International Conference on Computers & Industrial Engineering (CIE 2009)*, pp.554-59.

Khalouli, S., Ghedjati, F. & Hamzaoui, A., 2010a. Une méta-heuristique pour un ordonnancement juste à temps d'un atelier flow-shop hybride. *11ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2010)*, CD-ROM Proceedings.

Khalouli, S., Ghedjati, F. & Hamzaoui, A., 2010b. A meta-heuristic approach to solve a JIT scheduling problem in hybrid flow shop. *Engineering Applications of Artificial Intelligence*, 23(5), pp.765-71.

Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P., 1983. Optimisation by simulated annealing. *Science*, 220, pp.671-80.

Kis, T. & Pesch, E., 2005. A review of exact solution methods for the non-preemptive multiprocessor fowshop problem. *European Journal of Operational Research*, 164, pp.592-608.

Laguna, M., 2002. Global optimization and meta-heuristics. *Encyclopedia of Life Support Systems*, Theme 6.5, Topic 2.

Larrañaga, P., Etxeberria, R., Lozano, J.A., Sierar, B., Inza, I. & Peña, J.M., 1999. A review of the cooperation between evolutionary computation and probabilistic graphical models. *Proccedings of the Second Symposium on Artificial Intelligence, CIMAF 99, Adaptive Systems*, pp.314-24.

Larrañaga, P. & Lozano, J.A., 2002. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. & Shomoys, D.B., 1993. Sequencing and scheduling: algorithms and complexity. *In S.C. Graves, A.H.G. Rinnooy Kan, and P. Zipkin, editors, Handbooks in Operations Research and Management Science: Logistics of Production and inventory*.

Lee, C.Y., Cheng, T.C.E. & Lin, B.M., 1993. Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science*, 39(5), pp.616-25.

Lee, C.Y. & Vairaktarakis, G.L., 1994. Minimizing makespan in hybrid flowshop. *Operation Research Letters*, 16(3), pp.149-58.

Letouzey, A., 2001. *Ordonnancement interactif basé sur des indicateurs : Applications à la gestion de commandes incertaines et à l'affectation des opérateurs*. Thèse de Doctorat. Institut National Polytechnique de Toulouse.

Li, S., 1997. A hybrid two stage flow-shop with part family, bath production, major and minor setups. *European Journal of Operational Research*, 102, pp.142-56.

Liao, C.-J. & Juan, H.-C., 2007. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. *Computers & Operations Research*, 34, pp.1899-909.

Linn, R. & Zhang, W., 1999. Hybrid flow shop scheduling: a survey. *Computers and Industrial Engineering*, 37(1-2), pp.57-61.

Lopez, P. & Roubellat, F., 2001. *Ordonnancement de la production*. Paris: Hermès science publications.

Lourenço, H., Martin, O. & Stützle, T., 2003. Iterated local search. In F. Glover & G. Kochenberger, eds. *Handbook of Metaheuristics, volume 57 of International Series in Operations Research & Management Science*. Kluwer Academic Publishers. pp.320-53.

MacCarthy, B.L. & Liu, J., 1993. Addressing the gap in scheduling research : a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, 31(1), pp.59-79.

Marichal, J.-L., 1998. *Agregation operators for multicreteria decision aid*. Thèse de Doctorat. Université de Liège.

Merkle, D. & Middendorf, M., 2005. On solving permutation scheduling problems with ant colony optimization. *International Journal of Systems Science*, 36, pp.255-66.

Metropolis, N., Rosenbluth, A., Rosenbluth, M. & Teller, A., 1953. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21, pp.1087-92.

M'Hallah, R., 2007. Minimizing total earliness and tardiness on a single machine using a hybrid heuristic. *Computers & Operations Research*, 34, pp.3126-42.

Mladenović, N. & Hansen, P., 1997. Variable neighbourhood decomposition search. *Computers & Operations Research*, 24, pp.1097-100.

Mouloua, Z., 2007. *Ordonnancements coopératifs pour les chaînes logistiques*. Thèse de doctorat. Institut National Polytechnique de Lorraine.

Moursli, O., 1999. *Scheduling the hybrid flowshop: branch and bound algorithms*. Thèse de Doctorat. Université Catholique de Louvain.

Narasimhan, S.L. & Mangiameli, P.M., 1987. A comparison of sequencing rules for two stage hybrid flow-shop. *Decision Sciences*, 18, pp.250-65.

Narasimhan, S.L. & Panwalkar, S.S., 1984. The flow shop with parallel machines: A tabu search approch. *International Journal of Production Research*, 22(4), pp.555-64.

Nawaz, M., Enscore, E.E. & Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *OMEGA*, 11, pp.91-95.

Negenman, E.G., 2001. Local search algorithms for the multiprocessor flow shop scheduling problem. *European Journal of Operational Research*, 128(1), pp.147-58.

Néron, E., 1999. *Du flow-shop hybrid au Problème cumulatif*. Thèse de Doctorat. Université de Technologie de Compiègne.

Néron, E., Baptiste, P. & Gupta, J.N.D., 2001. Solving an hybrid flow shop problem using energetic reasoning and global operations. *Omega*, 29, pp.501-11.

Nicholson, T.A.J., 1971. *Optimization in industry, Volume one, Optimization Techniques*. London: Longmann Press.

Nowicki, E. & Smutnicki, C., 1998. The flow shop with parallel machines: A tabu search approach. *European Journal of Operational Research*, 106(2-3), pp.226-53.

Ow, P.S. & Morton, T.E., 1989. The single machine early/tardy problem. *Management Science*, 35(2), pp.177-91.

Palmer, D.S., 1965. Sequencing jobs through a multi-stage process in the minimum total time - a quick method of obtaining a near optimum. *Operational Research Quarterly*, 16(1), pp.101-07.

Pankaj, C., Peeyush, M. & Devanath, T., 2009. Permutation flow shop scheduling with earliness and tardiness penalties. *International Journal of Production Research*, 47(20), pp.5591-610.

Parunak, H.V.D., 1985. Manufacturing experience with the contract net. *In Proceedings of the Fith Workshop on Distributed Artificial Intelligence*.

Paul, R.J., 1979. A production scheduming problem in the glass container industry. *Operation research*, 27(2), pp.290-302.

Pelikan, M., Goldberg, D.E. & Lobo, F., 2002. A Survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21, pp.5-20.

Pomerol, J.-C. & Barba-Romero, S., 2000. *Multicriterion Decision Making in Management : Principles and Practice*. Kluwer ed. Boston.

Portmann, M.C., 1988. Méthodes de décomposition spaciale et temporelle en ordonnancement de la production. *RAIRO-APII*, 22(5), pp.439-51.

Portmann, M.C., Vignier, A., Dardilhac, D. & Dezalay, D., 1998. Branch and bound crossed with GA to solve hybrid flowshops. *European Journal of Operational Research*, 107(2), pp.389-400.

Potts, C.N., Shmoys, D.B. & Williamson, D.P., 1991. Permutation vs. non-permutation flow shop schedules. *Operations Research Letters*, 10, pp.281-84.

Rajendran, C. & Alicke, K., 2007. Dispatching in flowshops with bottleneck machines. *Computers & Industrial Engineering*, 52, pp.89-106.

Riane, F., Artiba, A. & Elmaghraby, S.E., 1998. A hybrid three stage flow-shop problem: efficient heuristics to minimize makespan. *European Journal of Operational Research*, 109, pp.321-29.

Richard, P., Barrier, J. & Proust, C., 1998. Economical aspects of short-term planning in the bottle-glass industry. *Revue VERRE*, 4(4), pp.7-14.

Rinnooy Kan, A.H.G., 1976. *Machine Scheduling Problems: Classification, Complexity, and Computations*. Netherlands: The Hague.

Rios-Solis, Y.A., 2007. *Ordonnancement avance-retard sur machines parallèles*. Thèse de Doctorat. Université Pierre et Marie Curie, Paris 6.

Rossi, A. & Boschi, E., 2009. A hybrid heuristic to solve the parallel machines job-shop scheduling problem. *Advances in Engineering Software*, 40, pp.118-27.

Rossi, A. & Dini, G., 2007. Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robotics and Computer-Integrated Manufacturing*, 23, pp.503-16.

Roy, B., 1985. *Méthodologie multicritère d'aide à la décision*. Paris.

Roy, B. & Sussman, B., 1964. *Les problèmes d'ordonnancement avec contraintes disjonctives*. Notes D.S. n° 9 bis, SEMA. Paris, France.

Ruiz, R. & Allahverdi, A., 2009. Minimizing the bicriteria of makespan and maximum tardiness with an upper bound on maximum tardiness. *Computers & Operations Research,* 36(4), pp.1268-83.

Ruiz, R. & Maroto, C., 2006. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research,* 169(3), pp.781-800.

Ruiz, R. & Vázquez-Rodríguez, J.A., 2010. The hybrid flow shop scheduling problem. *European Journal of Operational Research,* 205(1), pp.1-18.

Saad, I., Hammadi, S., Borne, P. & Benrejeb, M., 2008. Choquet integral for criteria aggregation in the flexible job-shop scheduling problems. *Mathematics and Computers in Simulation,* 76, pp.447-62.

Sankar, S.S., Ponnambalam, S.G., Rathinavel, V. & Visveshvaren, M.S., 2005. Scheduling in parallel machine shop: an ant colony optimization approach. *Industrial Conference on Industrial Technology, ICIT*, pp.276-80.

Santos, D.L., Hunsucker, J.L. & Deal, D.E., 1995. Global lower bounds for flow shops with multiple processors. *European Journal of Operational Research,* 80, pp.112-20.

Santos, D.L., Hunsucker, J.L. & Deal, D.E., 1996. An evaluation of sequencing heuristics in flow shops with multiple processors. *Computers and Industrial Engineering,* 30, pp.681-91.

Schoenauer, M. & Michalewicz, Z., 1997. Evolutionary computation. *Control and Cybernetics*, 26(3), pp.307-38.

Sherali, H.D., Sarin, S.C. & Kodialam., M.S., 1990. Models and algorithm for a two stage production process. *Production Planning and Control,* 1(1), pp.27-39.

Shyu, S.J., Lin, B.M.T. & Yin, P.Y., 2004. Application of ant colony optimization for no-wait flowshop scheduling problem to minimize the total completion time. *Computers & Industrial Engineering*, 47(2-3), pp.181-93.

Soewandi, H. & Elmaghraby, S.E., 2001. Sequencing three-stage hybrid flowshop with identical machines to minimize makespan. *IIE Transactions*, 33, pp.985-93.

Sourd, F. & Kedad-Sidhoum, S., 2003. The one machine scheduling with earliness and tardiness penalties. *Journal of Scheduling,* 6, pp.533-49.

Srinivasa Raghavan, N.R. & Venkataramana, M., 2009. Parallel processor scheduling for minimizing total weighted tardiness using ant colony optimization. *International Journal of Advanced Manufacturing Technology*, 41, pp.986-96.

Sriskandarajah, C. & Sethi, S., 1989. Scheduling algorithms for flexible flowshops: worst and average performance. *European Journal of Operational Research,* 43, pp.143-60.

Stützle, T., 1998. An ant approach to the flow shop problem. *In Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing*, pp.1560-64.

Stützle, T. & Hoos, H., 1997. Improvement in the ant system: introducing min-max ant system. *In proceedings of the international conference on Artificial Neuronal Networks and Genetic algorithms*, pp.266-74.

Sugeno, M., 1977. Fuzzy measures and fuzzy integrals: a survey. *In M.M Gupta, G.N Saridis, and B.R Gains, editors, Fuzzy automata and decision processes*, pp.89-102.

T'kindt, V. & Billaut, J.-C., 2002. *Multicriteria Scheduling : Theory, Models and Algorithms*. Berlin: Srpinger.

Talbi, E., 1999. *Métaheuristiques pour l'optimisation combinatoire multiobjectif*. Tutorial, Journées Evolutionnaires Trimestrielle.

Talbi, E.G., 2002. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8, pp.541-64.

Tay, J.C. & Ho, N.B., 2008. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3), pp.453-73.

T'kindt, V., Monmarche, N., Tercinet, F. & Laugt, D., 2002. An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. *European Journal of Operational Research*, 142(2), pp.250-57.

Valente, J.M.S. & Alves, R.A.F.S., 2007. Heuristics for the early/tardy scheduling problem with release dates. *International Journal of Production Economics*, 106, pp.261-74.

Van Der Zwaan, S. & Marques, C., 1999. Ant colony optimisation for job shop scheduling. *Proceedings of the 3rd Workshop on Genetic Algorithms and Artifical Life*.

Vázquez-Rodríguez, J.A. & Salhi, A., 2007. A Robust meta-hyper-heuristic approach to hybrid flow-shop scheduling. *Studies in Computational Intelligence (SCI)*, pp.125-42.

Ventura, J.A. & Kim, D., 2003. Parallel machine scheduling with earliness–tardiness penalties and additional resource constraints. *Computers & Operations Research*, 30(13), pp.1945-58.

Vignier, A., 1997. *Contribution à la résolution des problèmes d'ordonnancement de type monogamme, multimachines ("flow-shop hybride")*. Thèse de Doctorat. Université de Tour.

Vignier, A., Billaut, J.C. & Proust, C., 1999. Les Problèmes d'ordonnancement de type flow-shop hybride: état de l'art. *RAIRO-RO*, 33(2), pp.117-83.

Vignier, A., Billaut, J.-C., Proust, C. & Tkindt, V., 1996. Resolution of some 2-stage hybrid flowshop scheduling problems. *IEEE International Conference on Systems, Man, and Cybernetics*, 4, pp.2934-41.

Vignier, A., Commandeur, C. & Proust, P., 1997. New lower bound for the hybrid flowshop scheduling problem. *In Proceedings of IEEE sixth International Conference on Emerging Technologies and Factory Automation (ETFA 97)*, pp.446-51.

Voudouris, C. & Tsang, E., 1995. *Guided local search*. Technical Report CSM-247. Colchester, UK: University of Essex.

Widmer, M., 1991. *Modèles mathématiques pour une gestion efficace des ateliers flexibles*. Presses Polytechniques et Universitaires Romandes.

Wu, Z. & Weng, M.X., 2005. Multiagent scheduling method with earliness and tardiness objectives in flexible job shops. *IEEE Transactions on Systems, Man and Cybernetics-Part B*, 35, pp.293-301.

Yagmahan, B. & Yenisey, M.M., 2008. Ant colony optimization for multi-objective flow shop scheduling problem. *Computers and Industrial Engeneering*, 54, pp.411-20.

Yang, Y., Kreipl, S. & Pinedo, M., 2000. Heuristics for minimizing total weighted tardiness in flexible flow shops. *Journal of Scheduling*, 3(2), pp.89-108.

Ying, K.-C. & Liao, C.-J., 2003. An ant colony system approach for scheduling problems. *Production Planning and Control*, 14(1), pp.68-75.

Ying, K.-C. & Liao, C.-J., 2004. An ant colony system for permutation flowshop sequencing. *Computers & Operations Research*, 31(5), pp.791-801.

Ying, K.-C. & Lin, S.-W., 2006. Multiprocessor task scheduling in multistage hybrid flow-shops: an ant colony system approach. *International Journal of Production Research*, 44(16), pp.3161-77.

Ying, K.-C. & Lin, S.-W., 2007. Multi-heuristic desirability ant colony system heuristic for non permutation flowshops scheduling problems. *International Journal of Advanced Manufacturing Technology*, 33(7-8), pp.793-802.

Zadeh, L.A., 1965. Fuzzy sets. *Information and Control*, 8, pp.338-53.

Zhang, J., Hu, X., Tan, X., Zhong, J.H. & Huang, Q., 2006. Implementation of an ant colony optimization technique for job shop scheduling problem. *Transactions of the Institute of Measurement and Control*, 28, pp.93-108.

Zitzler, E. & Thiele, L., 1999. Multiobjective evolutionary algorithms: a comparative case study and strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4), pp.257-71.

Zlochin, M., Birattari, M., Meuleau, N. & Dorigo, M., 2004. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131, pp.373-95.

Méta-heuristiques à base de modèles : applications à l'ordonnancement d'atelier flow-shop hybride monocritère et multicritère

Résumé

Nous proposons dans cette thèse l'étude de problèmes d'ordonnancement monocritère et multicritère sur un atelier de production flow-shop hybride. Nous nous intéressons à la résolution approchée de problèmes qui diffèrent par leur fonction objectif que l'on cherche à minimiser : la date d'achèvement du travail le plus tardif (makespan), la somme pondérée des pénalités avance/retard et la somme totale des pénalités avance/retard. Notre objectif est d'explorer et de justifier l'utilisation d'algorithmes d'optimisation à base de colonie de fourmis pour la résolution de ces problèmes.

Le problème multicritère, que nous abordons, considère le makespan et la somme pondérée des pénalités avance/retard comme critères d'optimisation. Une approche hybride basée sur un algorithme de colonie de fourmis et la logique floue est développée pour résoudre ce problème. Cette approche permet de générer une multitude de solutions et emploie un module d'aide à la décision et d'évaluation pour sélectionner une solution parmi les solutions possibles en utilisant comme operateur d'agrégation une intégrale de Choquet.

Des expérimentations ont été effectuées sur des instances issues de la littérature ou générées aléatoirement, pour chacune des méthodes présentées. Les résultats obtenus sont prometteurs et l'intérêt de chaque approche est discuté.

Mots-Clés : ordonnancement, flow-shop hybride, optimisation par colonie de fourmis, production juste-à-temps, multicritère, logique floue, intégrale de Choquet.

Model-based Meta-heuristics: applications for mono-criterion and multi-criteria hybrid flow-shop scheduling problems

Abstract

In this thesis, we propose the study of mono-criterion and multi-criteria scheduling problems on a hybrid flow-shop factory. We focused on the approximate resolution of the problems that differ by their objective function that we seek to minimize: the makespan, the weighted sum of earliness/tardiness penalties and the total sum of earliness/tardiness penalties. Our objective is to explore and justify the using of algorithms based on ant colony optimization for the considered problems.

The multi-criteria problem that we address, considers the makespan and the weighted sum of earliness/tardiness penalties as optimization criteria. A hybrid approach based on an ant colony algorithm and fuzzy logic is developed in order to solve this problem. So, this approach enable the generation of multiple solutions and use a module for decision support and evaluation to select a solution among the possible solutions, by using a Choquet integral as aggregation operator.

Experimentations were performed on several benchmarks taken from the literature or randomly generated for all proposed methods. The obtained results are promising and their quality is discusses.

Key words: scheduling, hybrid flow-shop, ant colony optimization, just-in-time production, multicriteria, fuzzy logic, Choquet integral.