Approches de résolution de problèmes d'ordonnancement multi-agent à machines parallèles identiques

van Ut Tran

UNIVERSITÉ DE TOURS

École Doctorale Mathématiques, Informatique, Physique Théorique, Ingénierie des Systèmes (MIPTIS)

Laboratoire d'Informatique Fondamentale et Appliquée de Tours (LIFAT, EA 6300)
Equipe Recherche Opérationnelle, Ordonnancement et Transport (ROOT, ERL CNRS 7002)

THÈSE

présenté par **Van Ut TRAN**

soutenue le : 20 décembre 2018

pour obtenir le grade de : Docteur de l'Université de Tours

Discipline/ Spécialité : INFORMATIQUE

Approches de résolution de problèmes d'ordonnancement multi-agent à machines parallèles identiques

Thèse dirigée par :

Soukhal Ameur	Professeur des universités, Université de Tours

Rapporteurs :

Norre Sylvie	Professeur des universités, Université Clermont Auvergne, Clermont-Ferrand
Oulamara Ammar	Professeur des universités, Université de Lorraine, Nancy

Jury :

Billaut Jean-Charles	Professeur des universités, Université de Tours
Hao Jin-Kao	Professeur des universités, Université d'Angers
Kergosien Yannick	Maitre de Conférences, Université de Tours
Norre Sylvie	Professeur des universités, Université Clermont Auvergne, Clermont-Ferrand
Oulamara Ammar	Professeur des universités, Université de Lorraine, Nancy
Soukhal Ameur	Professeur des universités, Université de Tours

Remerciements

Je ne voudrais pas terminer ce travail sans remercier, ici, toutes celles et tous ceux, et ils sont nombreux, qui m'ont aidé à mener à terme cette thèse.

Je veux, tout d'abord et tout particulièrement, remercier et exprimer ma profonde gratitude à mon directeur de thèse le Professeur Ameur SOUKHAL pour toute son aide tout au long de thèse. Ses qualités scientifiques, mêlées à une gentillesse. Ses conseils avisés ont été très nombreux tout au long de ce travail, et m'ont permis de découvrir les fabuleux plaisirs de la recherche sous ses apparences les plus diverses. Je n'oublierai jamais son soutien et sa disponibilité dans les moments de doute. Je lui suis reconnaissant pour tous ces moments de partage qui ont agrémente mon parcours C'est également avec plaisir que je remercie mes rapporteurs pour le temps qu'ils ont accordé à la lecture de cette thèse et à lélaboration de leur rapport. Je remercie aussi les membre du Jury.

Je remercie tout particulièrement l'accueil du laboratoire d'Informatique, de lécole doctorale MIPTIS de l'université de Tours, ainsi que ses responsables qui m'ont permis de m'intégrer rapidement et de réaliser mes projets. Je n'oublie évidemment pas mes collègues du LIFAT avec lesquels j'ai partagé tous ces moments de doutes et de plaisirs. Tous ces instants autour les plates ou d'un café ont été autant de moments de d'entente indispensables pour une complète expression scientifique.

Je remercie le gouvernement du Vietnam de m'avoir accordé une bourse me permettant d'effectuer mes études de doctorat en France. Je remercie l'Université de Technologique de Can Tho pour m'avoir permis de venir en France pour faire mes études.

Je n'oublie pas, dans ces remerciements, l'association Touraine-Vietnam qui m'avez donné les cours de française, vos aides dans la vie de notre famille. Je remercie également l'association des étudiants Vietnamiens de Tours et Blois.

Je tiens à remercier tout particulièrement, et du fond du coeur, toute ma famille en pensant plus spécialement à mes parents, ma famille, notamment ma femme et ma fille pour tout leur soutien et leurs encouragements tout au long de ma thèse.

20 Décembre 2018, Tours, France
TRAN Van Ut

Résumé

Nous étudions des « problèmes d'ordonnancement multiagent non-disjoint ». Ces modèles considèrent différents agents associés à des sous-ensembles de travaux non nécessairement disjoints, chacun d'eux vise à minimiser un objectif qui ne dépend que de ses propres travaux. Deux types de critères sont considérés : minimisation du makespan et du nombre de travaux en retard. Nous cherchons donc les meilleurs compromis entre les critères des agents. Ces problèmes sont une classe particulière des problèmes d'ordonnancement « multi-agents » qui ont connu une grande expansion par leurs intérêts dans le domaine de l'ordonnancement et l'optimisation combinatoire. Dans nos travaux, nous nous sommes intéressés aux problèmes à machines parallèles identiques. Dans une première partie, nous étudions le cas d'une seule machine et celui où les travaux ont des durées identiques. Ainsi, des problèmes polynomiaux sont identifiés. Dans une seconde partie de nos travaux, nous abordons le problème à machines parallèles identiques avec deux agents. Notre étude porte sur l'énumération du front de Pareto par l'approche ε-contrainte en utilisant des modèles mathématiques à variables mixtes. Les résultats des expérimentations montrent les limites de ces méthodes. Pour résoudre des problèmes de grande taille, des heuristiques gloutonnes, hybrides, des métaheuristiques ou encore une matheuristique sont développées. Des expérimentations sont menées afin de montrer leurs performances par rapport aux méthodes exactes ou par rapport à la borne inférieure proposée.

Mots clés : Recherche Opérationnelle, Ordonnancement multi-agent, Machines parallèles, Fronts de Pareto, Programmation mathématique, Metaheuristiques, Matheuristique.

Abstract

We are studying "non-disjoint multi-agent scheduling problems'. These models consider different agents associated with non-disjoint subsets of jobs, each of them aims to minimize an objective function that depends only on its own jobs. Two types of criteria are considered: minimization of the makespan and minimization of the number of late jobs. We are therefore looking for the best compromise solution between the agents. These problems constitute particular class of "multi-agent" scheduling problems that have developed considerably due to their interests in scheduling and combinatorial optimization domains. In our work, we focused on problems with identical parallel machines. In a first part, we study the case of a single machine, then the case where the jobs are equal length. Thus, polynomial scheduling problems are identified. In a second part, we address the identical parallel machine scheduling problems with two agents. Our study focuses on the enumeration of the Pareto front by the ε-constraint approach using mathematical mixed integer programming. The computational results show the limitations of these methods. To solve large instances, greedy heuristics, hybrid heuristics, metaheuristic or even matheuristic are developed. The computational results show their performance compared to exact methods or to the proposed lower bound.

Keywords : Operational research, Multi-agent scheduling, Parallel machines, Pareto fronts, Mathematical programming, Metaheuristics, Matheuristic.

Contents

List of Tables

List of Figures

Chapter 1

Global introduction

Systems for production of goods and services are increasingly complex and do not facilitate the task of the decision-maker in charge of scheduling and managing production. Being competitive and staying competitive is a very important issue for companies. Moreover, to offer its products at a lower cost, in the right place and at the right time, the decision-maker is looking for the best organizational and production strategy. This begins with a detailed modeling of its production system, identifying: the tasks, the constraints related to their realization and those imposed by the system, and of course the objective function(s) to optimize. Then comes the resolution phase. The Operational Research tools provide solutions to the decision-maker regarding the scheduling jobs.

Scheduling theory has become an important field of combinatorial optimization, situated at the interface between applied mathematics, computer science and operational research. *"[...] scheduling is the process of organizing, choosing, and timing resource usage to carry out all the activities necessary to produce the desired outputs at the desired times while satisfying a large number of time and relationship constraints among the activities and the resources"* [Morton et Pentico, 1993].

Scheduling problems are often presented in a multi-criteria form. Indeed, the decision-makers generally seek to find the best compromise between several conflicting criteria. So we are talking about "multi-criteria scheduling problems". The classic definition of multi-criteria scheduling problems supposes that all jobs are concerned by all the criteria to be optimized. Several studies tackled this problem. More recently, few articles have appeared dealing with scheduling problems, considering that there may be jobs that are concerned by one criterion but not another. This corresponds to some practical cases.

- In the case of a supply chain, for example, the organization is based on the coordination of actors from the supplier to the customer. Synchronization and delivery problems require a perfectly ordered coordination, where each actor expects an irreproachable quality of service. Taking into account the particularities of customers (some are more demanding on deadlines, some on quantities, some wish to group their deliveries, others do not, etc.) requires the introduction of parameters and metrics dedicated to certain jobs and not to others. By nature, jobs compete for resources, jobs interfere with their achievement and their objectives are not necessarily the same.

- In communication networks, delays are not tolerated for audio or video packets. However, text packets may be delayed but data loss is less tolerated.

These situations generally lead to the definition of one criterion for some jobs and then another criterion for others jobs. This type of problem requires studies on multi-agent scheduling. The definition of multi-agent scheduling problems is as follows: given a set of jobs to be scheduled on a set of machines, sub-sets of jobs are defined such that each sub-set aims to minimize a criterion that depends only on the execution of its own jobs [Agnetis *et al.*, 2014]. In this case, each subset of jobs is associated with decision-maker or user. These subsets of jobs are called "agents".

Depending on the relationship between the subsets of jobs associated with agents, scenarios are defined as follows: *Competition* when agents only share common resources; *Interfering* when subsets of jobs are included in each other; A more general scenario is also defined, indicating that agents may have common jobs, it is referred to as *Non-disjoint*. The last scenario is the subject of our study. More specifically, we focus on two non-dissident agents, A and B. The two agents share the same identical parallel machines to schedule their jobs. The objective function of one agent is to minimize the makespan, while the objective function of the other agent is to minimize the total number of late jobs. The objective of our study is then to develop efficient methods to determine and compute the Pareto front. The ε-constraint approach is used.

The rest of the manuscript is structured as follows:

- Chapter 1 is dedicated to the presentation of the basic principles of multi-criteria optimization. Concepts such as dominance, resolution methods and multi-criteria resolution approaches are presented.

- In Chapter 2, after a short introduction to the mono-criterion scheduling problems and multi-criteria scheduling problems, we present the definitions and classification of multi-agent scheduling problems. The complexity relationships between the different scenarios are recalled. We then recall the main results of the literature on multi-agent scheduling problems. As we will see, there few results dealing with the non-disjoint scenario than the competition case.

- In Chapter 3, we study some particular cases and show that are polynomially solvable. We then present the optimal structure of a strict Pareto solution. To solve the problem at the optimum, we propose two mixed linear programming models. The tables of experimental results are presented at the end of this chapter.

- In chapter 4, we propose a lower bound for the minimization of the makespan. This lower bound then allows us to deduce the lower bound of the Pareto front. Some greedy heuristics and their hybridizations with exact methods (MILP or dynamic programming algorithm) are proposed. The tables of experimental results are also presented at the end of this chapter.

- Chapter 5 is dedicated to the presentation of iterative improvement methods: Tabu Search algorithm, NSGA-II and Matheuristics. Subsequently, experiments are carried out to show the performance of the proposed methods.

- We conclude this manuscript with Chapter 6, in which we give a summary of our work and our research perspectives.

Introduction globale

Les systèmes de production de biens et de services sont de plus en plus complexes et ne facilitent pas la tâche du décideur chargé d'ordonnancer et de gérer la production. Etre compétitif et rester compétitif est une question très importante pour les entreprises. De plus, pour offrir ses produits à moindre coût, au bon endroit et au bon moment, le décideur recherche la meilleure stratégie d'organisation et de production. Cela commence par une modélisation détaillée de son système de production, identifiant : les tâches, les contraintes liées à leur réalisation et celles imposées par le système, et bien sûr la ou les fonctions objectives à optimiser. Vient ensuite la phase de résolution. Les outils de recherche opérationnelle apportent des solutions au décideur en ce qui concerne l'ordonnancement des tâches.

La théorie de l'ordonnancement est devenue un domaine important de l'optimisation combinatoire, situé à l'interface entre les mathématiques appliquées, l'informatique et la recherche opérationnelle. *"[...] scheduling is the process of organizing, choosing, and timing resource usage to carry out all the activities necessary to produce the desired outputs at the desired times while satisfying a large number of time and relationship constraints among the activities and the resources"* [Morton et Pentico, 1993].

Les problèmes d'ordonnancement sont souvent présentés sous une forme multicritères. En effet, les décideurs cherchent généralement à trouver le meilleur compromis entre plusieurs critères conflictuels. Nous parlons donc de "problèmes d'ordonnancement multicritères". La définition classique des problèmes d'ordonnancement multicritères suppose que tous les travaux soient concernés par tous les critères à optimiser. Autrement dit, la qualité de la solution est mesurée par des critères appliqués sur tous les travaux sans distinction. C'est le cas classique largement étudié dans la littérature spécialisée. Cependant, peu d'articles sont parus sur les problèmes d'ordonnancement considérant qu'il peut y avoir des travaux qui sont concernés par un critère mais pas par un autre. Néanmoins, depuis une dizaine d'années, de plus en plus de chercheurs s'intéressent aux problèmes d'ordonnancement où les critères à optimiser sont définis sur des sous-ensembles de travaux. Cela correspond à certains cas pratiques comme suits.

- Dans le cas d'une chaîne logistique, par exemple, l'organisation repose sur la coordination des acteurs, du fournisseur au client. Les problèmes de synchronisation et de livraison nécessitent une coordination parfaitement ordonnée, où chaque acteur attend une qualité de service irréprochable. La prise en compte des particularités des clients (certains sont plus exigeants sur les délais, d'autres sur les quantités, certains souhaitent grouper leurs livraisons, d'autres non, etc.) nécessite l'introduction de paramètres et de métriques dédiés à certains travaux et non à d'autres. Par nature, les travaux sont en compétition pour l'utilisation des ressources, les travaux interfèrent pour leur réalisation et leurs objectifs ne sont pas nécessairement les mêmes.
- Dans les réseaux de communication, les retards ne sont pas tolérés pour les paquets audio ou vidéo. Cependant, les paquets de texte peuvent être retardés, mais la perte de données est moins bien tolérée.

These situations generally lead to the definition of one criterion for some job and then

another criterion for others jobs. This type of problem requires studies on multi-agent scheduling. The definition of multi-agent scheduling problems is as follows: given a set of jobs to be scheduled on a set of machines, sub-sets of jobs are defined such that each sub-set aims to minimize a criterion that depends only on the execution of its own jobs [Agnetis *et al.*, 2014]. decision maker is associated with each subset of jobs, these subsets are called "agents".

Ces situations conduisent généralement à la définition d'un critère pour certains travaux, puis d'un autre critère pour d'autres travaux. Ce type de problème nécessite des études sur l'ordonnacement multi-agents.

La définition des problèmes d'ordonnancement multi-agent est la suivante : étant donné un ensemble de travaux à planifier sur un ensemble de machines, des sous-ensembles de travaux sont définis de telle sorte que chaque sous-ensemble vise à minimiser un critère qui dépend uniquement de l'exécution de ses propres travaux [Agnetis *et al.*, 2014]. Dans ce cas, chaque sous-ensemble de travaux est associé à un décideur ou utilisateur. Ces sous-ensembles sont appelés "agents".

Depending on the relationship between the subsets of jobs associated with agents, scenarios are defined as follows: *Competition* when agents only share common resources; *Interfering* when subsets of jobs are included in each other; A more general scenario is also defined, indicating that agents may have common jobs, it is referred to as *Non-disjoint*. The last scenario is the subject of our study. More specifically, we focus on two non-dissident agents, A and B. The two agents share the same identical parallel machines to schedule their jobs. The objective function of one agent is to minimize the makespan, while the objective function of the other agent is to minimize the total number of late jobs. The objective of our study is then to develop efficient methods to determine and compute the Pareto front. The ε-constraint approach is used.

Selon la relation entre les sous-ensembles de travaux associés aux agents, des scénarios ont été définis comme suit : *Competition* lorsque les agents ne partagent que des ressources communes ; *Interfering* lorsque des sous-ensembles de travaux sont inclus les uns dans les autres ; Un scénario plus général est également défini, indiquant que les agents peuvent avoir des travaux communs, il est appelé *Non-disjoint*. Le dernier scénario fait l'objet de notre étude. Plus précisément, nous nous concentrons sur deux agents non disjoints, A et B. Les deux agents partagent les mêmes machines parallèles identiques pour ordonnancer leurs travaux. La fonction objective d'un agent est de réduire au minimum la durée du travail, tandis que la fonction objective de l'autre agent est de réduire au minimum le nombre total de travaux en retard. L'objectif de notre étude est ensuite de développer des méthodes efficaces pour déterminer et calculer le front de Pareto. L'approche ε-constraint est utilisée dans notre étude.

Le reste du manuscrit est structuré comme suit :

- Le chapitre 1 est consacré à la présentation des principes de base de l'optimisation multicritères. Des concepts tels que la dominance, les méthodes de résolution et les approches de résolution multicritères sont présentés.

- Dans le chapitre 2, après une brève introduction aux problèmes d'ordonnancement mono-critères et multicritères, nous présentons les définitions et la classification des

problèmes d'ordonnancement multi-agents. Les relations entre les différents scénarios sont rappelées nous permettant de déduire la complexité de certains problèmes. Nous rappelons ensuite les principaux résultats de la littérature sur les problèmes d'ordonnancement multi-agents. Comme nous le verrons, il y a peu de résultats concernant le scénario non disjoint, contrairement au scénario Compétition qui a connu ces dernières années de développement important.

- Au chapitre 3, nous étudions quelques cas particuliers et montrons qu'ils peuvent être résolus polynomialement. Nous présentons ensuite la structure optimale d'une solution de Pareto stricte. Pour résoudre le problème à l'optimum, nous proposons deux modèles de programmation linéaire en nombres mixtes. Les tableaux des résultats expérimentaux sont présentés à la fin de ce chapitre.

- Dans le chapitre 4, nous proposons une borne inférieure pour la minimisation du makespan. Cette borne inférieure nous permet alors de borner inférieurement le front de Pareto. Quelques heuristiques gloutonnes et leurs hybridations avec des méthodes exactes (MILP ou algorithmes de type programmation dynamique) sont proposées. Les tableaux des résultats expérimentaux sont également présentés à la fin de ce chapitre.

- Le chapitre 5 est consacré à la présentation des méthodes itératives d'amélioration : Algorithme de recherche Tabu, NSGA-II et Matheuristics. Par la suite, des expériences sont menées pour montrer la performance des méthodes proposées.

- Nous concluons ce manuscrit par le chapitre 6, dans lequel nous donnons un résumé de notre travail et de nos perspectives de recherche.

Chapter 2

Introduction to multi-criteria optimization

Résumé: Nous consacrons ce chapitre à la définition des notions théoriques qui seront utilisées dans ce manuscrit. Ce chapitre est une introduction au manuscrit.

Ce chapitre est organisé comme suit : dans la section 2.2 nous présentons la définition des problèmes d'optimisation multicritères ainsi que les concepts de dominance. La section 2.3 est consacrée aux différentes approches pour résoudre les problèmes d'optimisation multicritères. Dans la section 2.4, les différentes méthodes de résolution qui nous intéressent dans notre étude sont discutées. La section 2.5 est consacrée aux approches développées pour mesurer la qualité des solutions des problèmes multicritères. Enfin, le chapitre se termine par une conclusion présentée dans la section 2.6.

Abstract: We devote this chapter to the definition of the theoretical notions which will be used in this manuscript. This chapter is an introduction to the manuscript.

This chapter is organized as follows: in section 2.2 we present the definition of multi-criteria optimization problems as well as dominance concepts. Section 2.3 is devoted to the different approaches to solve multi-criteria optimisation problems. In section 2.4, the different resolution methods which interest us in this thesis study are discussed. Section 2.5 is devoted to measuring the quality of the solutions given for a multi-criteria problem. And finally the chapter concludes with a conclusion given in section 2.6.

2.1 Introduction

Scheduling problems are an important part of combinatorial optimization problems. These problems are encountered in any operating system when it comes to organize activities or tasks over time and determining their best allocation(s) to consumable or renewable resources [Carlier et Chrétienne, 1988]. The scheduling theory is a very broad field due to the diversity of systems and activities to be planned over time. However, it turns out that in practice the scheduling problems are complex and more difficult to solve. Thus, scheduling theory lies in the development of theoretical and algorithmic tools for modeling

and solving various problems encountered by practitioners. Dealing with industrial company for example, scheduling is one of important task for production management where the decision maker has to define the schedules for launching the activities while respecting the production constraints previously defined. Difficulties in job assigning and sequencing are also encountered in other areas such as communication or the exchange of all types of data via networks. We can refer to classic books presenting classical models, more recently studied models and also resolution algorithms. Blazewicz et al. [Blazewicz *et al.*, 2007], Brucker [Brucker, 2007] and Pinedo [Pinedo, 2008] are cited as reference books dedicated to single-criterion scheduling problems. These reference books tackle the classic cases of scheduling problems.

Dealing with multi-criteria scheduling problems, the quality of scheduling is depending on several objective functions. In this case, the whole set of jobs contributes in the same way to measuring the quality of the solution, and this is given by several criteria. Hence, a decision-maker applies a set of criteria on the totality of jobs that should be processed. For example, the maximum completion date of all jobs will be considered as one criterion and the maximum tardiness of all jobs as another criterion. Here too scheduling theory has received great interest. We can quote for example the works of T'kindt and Billaut [T'Kindt et Billaut, 2006], Jozefowska [Jozefowska, 2007] and [Rios-Mercado et Rios-Solis, 2012].

In our study, a set of objective functions is considered to determine best schedules where each criterion may relate to a subset of jobs. For example, some jobs should be scheduled as early as possible (minimize the maximum completion time), while respecting a limit for the greatest late of certain jobs. These problems arise in these terms for certain companies where daily production is evaluated by a criterion which covers some jobs, and where another subsets of jobs should respect constraints -sometimes imposed by customers or relating to delays already accumulated- according to other criteria. It is about a particular class of multi-criteria scheduling problems called "multi-agents scheduling problems". The fundamental idea of these models is the taking into account of several decision-makers (agents) who share the common resources to execute their respective jobs. As with multi-criteria problems, the objective in this case is to found a schedule, which gives better compromise solutions between the different decision-makers.

From multi-criteria combinatorial optimization domain, multi-agent scheduling offers exciting challenges for researchers by their theoretical and practical complexity. Indeed, the specificity of multi-agent scheduling problems is the existence of several objectives to be optimized at the same time, which recalls the definition of a multi-criterion optimization problem.

2.2 Definitions and dominance concept

Multi-criteria optimization (also known as multi-objective programming, vector optimization, multi-criteria optimization, multi-attribute optimization or Pareto optimization) is a branch of combinatorial optimization whose peculiarity is to simultaneously optimize several objectives of the same problem (against a single objective for classical combinatorial optimization).

Multi-objective problems have a growing interest in the industry where managers are forced to try to optimize conflicting objective functions. These problems are in general $\mathcal{NP}$-hard even if the mono-objective version is not. Hence, their resolution in reasonable times becomes necessary and requires a mastery of operational research tools. For example, let us consider the following polynomially solvable scheduling problem: single machine scheduling problem minimizing the total completion times. The problem instance is defined by a number of jobs that should be executed on a single machine minimizing the total completion times. However, if the set of jobs is split into two subset of jobs each one should be executed on the same machine to be executed in order to minimize the total completion times, the problem becomes $\mathcal{NP}$-hard [Baker et Smith, 2003].

Multi-criteria optimization consists in choosing, in the presence of multiple criteria, one or more alternatives among an infinite number of alternatives that generally vary in a continuous field. The field of multi-criteria optimization is undergoing a major evolution. This development has resulted in the development of a large number of methods. The multitude of multi-criteria optimization methods is perceived as an indisputable richness of this domain. Moreover, some justify it by the diversity of the problems as well as by the existence of different possible and legitimate approaches to the resolution of these problems.

The multi-objective optimization has been applied in many fields such as transportation, scheduling and planning, where optimal decisions should be taken in the presence of two or more trade-offs between conflicting objectives. Minimizing costs while maximizing quality of service for the production of good or service, or maximizing performance while minimizing energy consumption or greenhouse gas emissions are examples of a multi-objective optimization problems (at least two objectives). More general definition of Multi-criteria optimization problem can be introduced as following.

Definition 1 *Multi-criteria optimization problem:*

Given:

- *A vector function* $F(x) = (f^1(x), f^2(x), ..., f^K(x))$ *and*
- *A feasible solution space* Ω.

The Multi-criteria optimization problem consists to find a vector $x \in \Omega$ *that optimizes the vector function* $F(x)$*, where vector* $x = (x_1, x_2, ..., x_n)$.

x *is called the search point or solution candidate and* X *is the search space or decision space. Finally,* $f^1(x), f^2(x), ..., f^K(x)$ *denote the objective functions* $1, 2, ..., K$.

2.2.1 Dominance concept

Initially used in economic and social sciences, the concept of dominance was introduced by [Pareto, 1897] to express the fact that it is no longer possible to improve the situation of one individual without degrading that of at least one other. Nowadays, this concept is understood in other fields such as multicriteria optimization area. In fact, the concept of Pareto dominance is of extreme importance in multi-objective optimization, especially where some or all of the objectives and constraints are mutually conflicting. In such a

case, there is no single point that yields the "best" value for all objectives and constraints. Instead, the best solutions, often called a Pareto or non-dominated set, are a set of solutions such that selecting any one of them in place of another will always sacrifice quality for at least one objective or constraint, while improving at least one other. Formally, the description of such Pareto optimality for generic optimization problems can be formulated in the following [Santiago *et al.*, 2014].

Definition 2 *Pareto dominance:*

A vector x dominates y (denoted by $x \prec y$):

- *If $f^k(x) \leq f^k(y), \forall k = 1, ..., K$, and*
- *There is at least one i such that $f^i(x) < f^i(y)$.*

Definition 3 *Pareto optimal:*

A vector x^* is Pareto optimal if does not exists a vector $x \in \Omega$ such that $x \prec x^*$.

2.2.2 Pareto front structure and reference points

Definition 4 *Strict Pareto optimal solution*

A feasible solution x is a strict Pareto optimal solution or strictly nondominated solution with respect to the objective functions $f^1, f^2, \ldots, f^K$, if there is no feasible solution y such that $f^k(y) \leq f^k(x), \forall k, 1 \leq k \leq K$, with at least one strict inequality.

Definition 5 *Weak Pareto optimal solution:*

A feasible schedule x is a weak Pareto optimal solution or weakly nondominated solution with respect to the objective functions $f^1, f^2, \ldots, f^K$, if there is no feasible schedule y such that $f^k(y) < f^k(x), \forall k, 1 \leq k \leq K$.

The set of weak Pareto optimal points define the tradeoff curve. The position of strict and weak Pareto optimal solutions is illustrated in Fig. 2.1 for $K = 2$.

Definition 6 *Pareto optimal set:*

The Pareto optimal set for a multi-criteria optimization problem is defined as: $P^* = x^* \in \Omega$.

Definition 7 *Pareto front:*

Given a multi-criteria optimization problem and its Pareto optimal set, the Pareto front is defined as $PF^* = F(x)|x \in P^*$.

Example: Let consider two objective functions to be minimized. According to Figure 2.1, some Pareto solutions are represented, denoted by $P1, P2, P3, P4, P5, P6$, and $P7$

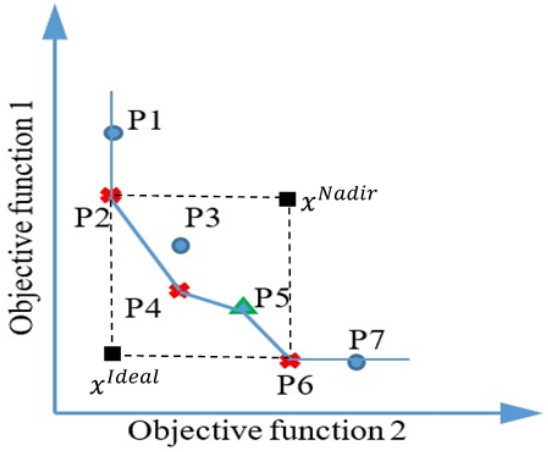

Figure 2.1: Pareto solutions

We can easily verify that:

- $P2$ dominates $P1$; $P4$ dominates $P3$; $P6$ dominates $P7$;
- $P2, P4, P5, P6$ are not dominated;
- $P2, P4, P5$ and $P6$ are strict Pareto optimal solution. Note that $P2, P4$ and $P6$ are also called *Supported Pareto optimal solutions* where $P5$ is called *Unsupported Pareto solution*;
- $P1, P3$ and $P7$ are weak Pareto optimal solution.

The Pareto front of a multi-objective optimization problem is bounded by a so-called *Nadir point* and *Ideal point*, if these are finite (see Figure 2.1).

- The Nadir point is defined as: $x^{Nadir} = max_T(x \in \Omega) f^k(x), \forall k = 1, \ldots, K$;
- The Ideal point is defined as: $x^{Ideal} = min_T(x \in \Omega) f^k(x), \forall k = 1, \ldots, K$.

Solving multi-objective optimization problem consists in finding Pareto optimal solutions. In case of finding one Pareto optimal solution, we usually interest only in the strict Pareto optimal. The whole set of Pareto optimal solutions can be obtained by finding strict Pareto optimal solutions one by one and iteratively. Finally, in counting the number of Pareto optimal solutions, the aim is to count the number of optimal solutions or to give an approximation of their number. These methods are described in details in [T'Kindt et Billaut, 2006].

2.3 Some multi-criteria approaches

At the phase of taking account of criteria, and following the information that sets out, the decision maker chooses a resolution approach for the multi-criteria problem and

thus defines his optimization problem. Taking account of the diversity of the methods of determining Pareto optima, the functions to optimize for the problem can take different forms. Each one translates a method of determining a Pareto optimum. The criteria do not change and they correspond to those defined during the phase of modeling of the problem [T'Kindt et Billaut, 2006].

2.3.1 Linear combination

Linear combination is an expression constructed from a set of terms by multiplying each term by a constant and adding the results (e.g. a linear combination of x and y would be any expression of the form $ax+by$, where a and b are constants). Based on this concept, we can apply linear combination of multi-criteria.

For example, dealing with k objective functions $f^1, \ldots, f^k$ to be minimized, linear combination approach consists in defining a linear combination of k objective functions: $a_1 f^1(x) + a_2 f^2(x), \ldots a_K f^K(x)$ with $x \in \Omega$ and $a_k, k = 1, \ldots K$ are constants and verify: $\sum_{k=1}^{K} a_k = 1$.

The solutions that can be obtained by this approach constitute a subset of the set of strict Pareto optimal solutions [Geoffrion, 1968]. In other words, it may be impossible to fix weights to the criteria so that all the Pareto optimal solutions are obtained. This is due to the shape of the tradeoff curve, which can be non-convex for problems and therefore, the non-supported Pareto optimal solutions cannot be returned by such a method.

2.3.2 $\varepsilon-$constraint approach

The idea here is to reduce the multi-criteria optimization problem to a mono-criteria optimization problem where only one objective function has to be optimized, denoted $\varepsilon(f^k/f^1, \ldots, f^{(k-1)}, f^{(k+1)}, \ldots, f^K)$. For example, $\varepsilon(f^1/f^2, f^3, ..., f^K)$ indicates that only the criterion f^1 is optimized, subject to all the other criteria being upper bounded by given values. This case is distinguished from the previous case because the function to be optimized is f^1 and this criterion is not subject to any bound constraint. More formally, dealing with minimizing problem, the problem consists of finding solution s^* such that:

$$\text{Min } f^1(s)$$

$$st. \begin{cases} f^2(s) \leq Q_2 \\ f^3(s) \leq Q_3 \\ ... \\ f^K(s) \leq Q_K \end{cases} \tag{2.1}$$

In general, this method leads to one weak Pareto solution. For finding a strict Pareto optimal solution, a symmetric problem has to be solved (see Algorithm 1).

Algorithm 1 Finding strict Pareto solution by $\varepsilon-$constraint approach: case of two objective functions

1: Let S be the set of feasible solutions
2: Find $s_1 \in S$ such that $x_1 = minf^1(s_1)$ and $f^2(s_1) \leq Q_2$, if there exists;
3: Find $s_2 \in S$ such that $x_2 = minf^2(s_2)$ and $f^1(s_2) \leq x_1$;
4: Return strict Pareto solution $s^* = s_2$ with $(x_1^*, x_2^*) = (f^1(s_2), f^2(s_2))$;

2.3.3 Lexicographic order

Denoted by $Lex(f^1, f^2, \ldots, f^K)$, the decision maker does not allow any trade-offs between the criteria when he uses this approach. The order in which the criteria are given is related to their importance, the most important being in first. The decision maker uses the lexicographical order and optimizes the criteria one after the other as follow: the first criterion is minimized. Then, we search for another solution that minimizes the second criterion under the constraint that this new solution is optimal for the first criterion, and so on.

2.3.4 Pareto set enumeration

Pareto set enumeration is denoted by $P(f^1, f^2, ..., f^K)$. In this case we are looking for the whole set of strict Pareto optimal solutions. To determine this set of strict Pareto optimal solutions, an $\varepsilon-$constraint approach can be used, for example, by modifying the vector $Q = (Q_1, \ldots, Q_K)$ iteratively. Notice that several algorithms have been introduced in the literature for improving the implementation of such procedure. Algorithm below illustrates this approach as introduced in [Agnetis *et al.*, 2014] in the case of K objectives functions. From Algorithm 2, the value δ_k denotes the predefined decrement of Q_k. LB_k and UB_k denote the bound of objective function f^k and the function $lex-min(f)$ returns the solution with minimum lexicographic value, i.e. with minimum f^1, then minimum f^2, ..., minimum f^k.

Algorithm 2 for the enumeration of Pareto optimal solution with $\varepsilon-$constraint approach [Agnetis *et al.*, 2014]

$R = \emptyset$;
for each $Q_2 = UB_2$ **down to** LB_2 **step** δ_2 **do**
  for each $Q_3 = UB_3$ **down to** LB_3 **step** δ_3 **do**
    .....
    for each $Q_K = UB_K$ **down to** LB_K **do**
      $x = lex - min\{f : f^k(x) \leq Q_k, 2 \leq k \leq K\}$
      if not exit $x' \in R$ such that $f(x') \leq f(x)$ **then**
        $R := R \bigcup \{x\}^*$
      end if
    end for
    .....
  end for
end for
Return R

2.3.5 Counting approach

Denoted by $\#(f^1, \ldots, f^K)$, this approach means that the objective is to count the number of strict Pareto optimal solutions.

2.4 Resolution methods

In this section, we will recall some of the resolution methods (exact and approximate) used to solve our scheduling problems studied in this thesis. It is therefore not a question of carrying out a complete and detailed state of the art, but simply of recalling these methods in their broad outlines.

2.4.1 General structure of multi-criteria optimization algorithms

As already mentioned, multi-criteria optimization belongs to a broad class of combinatorial optimization problems. Many resolution algorithms for multi-criteria optimization can be developed. In general, we divide two groups, that are exact algorithms and heuristics/metaheuristics methods. We can solve these problems by using exact algorithms which for sure always find optimal solutions, if there exists and can be calculated within reasonable time. This is the case of 'easy' problem, it means that a polynomial time optimization algorithm can be constructed to calculate strict Pareto solutions. Otherwise, some exact methods such as: Branch and Bound, Mathematical programming or Dynamic programming can be used to determine an optimal solution for small size instances.

Unfortunately, more multi-criteria problems are $\mathcal{NP}$-hard, and use exact methods is not effective. In this case, heuristics offer better trade-offs between the quality of the solution and the computation time. In fact, heuristic algorithm is a technique designed for solving a problem more quickly when exact methods are too slow, or for finding an approximate

solution when classic methods fail to find an exact solution. With the algorithm, we can find the feasible solution (some time, it is optimal solution) in short time. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut. Different approaches can be used to build heuristic: simple approach as dedicated heuristics based on priority rules for example, or complex approach such as metaheuristics, matheuristic, hybrid heuristic, etc.

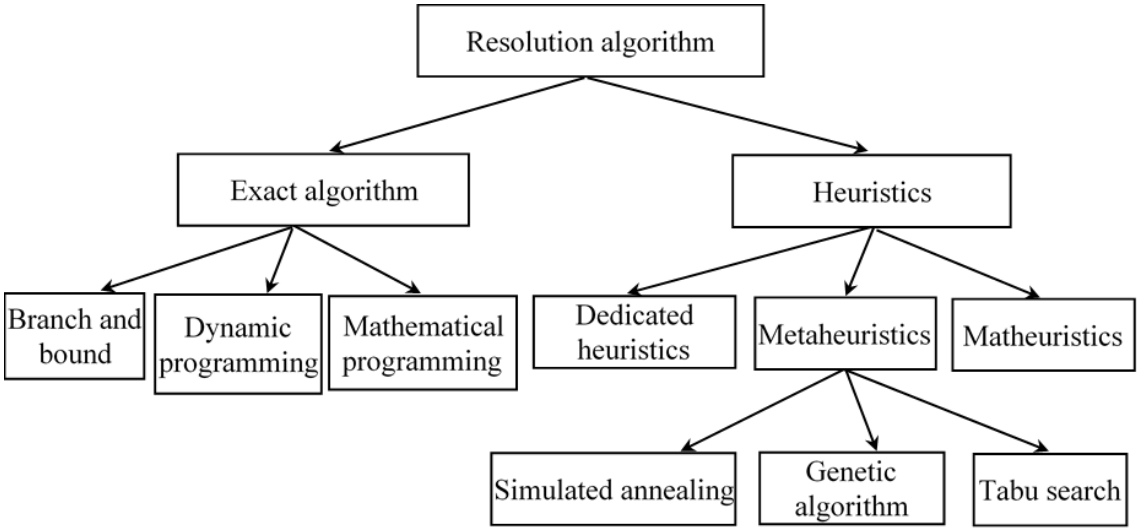

Figure 2.2: General structure of multi-criteria optimization algorithms

2.4.2 Exact algorithm

One of the exact techniques to obtain an optimal solution is the straightforward enumeration, where every possible solution is explored in order to find the optimal solution. However, due to the computational complexity, it is not practical even for problems of moderate sizes. We have therefore to search for more sophisticated resolution methods. For combinatorial optimization problems, it can be tempting to enumerate the whole set of potential solutions and to keep the best solution. For example, finding the best sequence of a scheduling problem with n jobs can be done by enumerating then $n!$ possible sequences. However, this method is not possible even we use supercomputers!

2.4.2.1 Mathematical Programming

Integer Linear Programming (ILP) or Mixed Integer Linear Programming (MILP) are two variants of mathematical programming that sometimes are used for solving combinatorial optimization problems. In both cases, first we formulate the problem to be solved in terms of variables, restrictions and functions, and next, using commercial or open source solvers to obtain solution(s). The resolution methods for solving ILP or MILP problems are generic methods using sophisticated variants of branch-and-bound algorithms.

The general formulation of mathematical programming is below:

$$\text{Minimize } f(x)$$
$$st. \begin{cases} g(x) \leq 0 \\ x \in S \subset R^n \end{cases} \tag{2.2}$$

x is a vector of R^n where $x = (x_1, x_2, ..., x_n)$. x_i is the decision variable. The function $f(x)$ is objective function to be minimized and $g(x)$ is the constrain of the problem.

According to decision variables and dealing with scheduling problems, in general there are three basic ways to formulate a scheduling problem using mathematical programming: positional variables, precedence variables, time-indexed variables.

Positional variables: Let j be the index of job and let l be the position of job where $j \in \{1, \ldots, n\}$ and $l \in \{1, \ldots, L\}$, n is the number of jobs to be scheduled and L is the total number of possible positions that can be assigned. Hence, we define binary variables $x_{j,k}$ equal to 1 if job j_j is in position l and 0 otherwise. For example, this type of variables can be used when the considered scheduling problem is equivalent to find the order of execution jobs on a single machine or non-permutation scheduling problems [Wagner, 1959].

Precedence variables: We define binary variables $x_{i,j}$ to define which job should executed before another one. We have: $x_{i,j}$ equal to 1 if job j_i precedes job j_j and 0 otherwise [Bowman et Edward, 1959].

Time-Indexed Variables: The classical time-indexed 0-1 linear programming formulations for scheduling problems involve binary variables indicating whether an activity starts precisely at or before a given time period. We define binary variables $x_{j,t}$ equal to 1 if job j is being performed at time t; 0 otherwise, where $t \in [0, \ldots, T]$ and T is the planning horizon. Some examples and different formultaions for time-indexed linear programming formulations can be found in [Bowman et Edward, 1959]. For example, we can define decision variables as follow: $x_{j,t}$ is equal to 1 if job j starts its processing exactly at time t; 0 otherwise.

2.4.2.2 Dynamic programming algorithm

Dynamic programming (also known as dynamic optimization) is a method for solving a complex problem by breaking it down into a collection of simpler sub-problems, solving each of those sub-problems just once, and storing their solutions. Dynamic Programming (DP) is an implicit enumeration method, based on the Bellman's principle of optimality [Bellman et Kalaba, 1957]. The main idea of the DP is that a problem -satisfying certain conditions- can be decomposed into sub problems of the same nature, and the optimal solution of the problem can be obtained from the optimal solutions of the sub-problems, by using recursive relations. Each step of the recursion is called a phase. The problem is in a given state at the beginning of the phase, and after taking some decisions, the problem enters into another state. A final state of the DP corresponds to an optimal solution. Depending on the number of states and phases, the running time of a DP algorithm can be polynomial, pseudo polynomial or exponential [Sadi *et al.*, 2014, Agnetis *et al.*, 2014, Soukhal, 2012].

A DP formulation is characterized by three types of expressions:

- some initial conditions
- recursive relation
- goal (i.e. an optimal value function).

2.4.3 Heuristic approaches

Heuristic methods are algorithms used to find solutions among all possible ones, but these algorithms do not guarantee that the best solution will be found. Therefore, they are considered as approximated algorithms. They usually find quickly and easily a solution closes to the optimal one. On some instances, these algorithms may return optimal solutions. They can be simple methods based on priority rules or complex methods based on Operational Research models or derived from graph theory such as min cost max flow problem, generalized assignment problem, knapsack problem, vehicles routing problems, etc.

2.4.4 Metaheuristic approaches

2.4.4.1 Local search algorithms

The search space associated with a combinatorial optimization problem is often non-enumerable in a reasonable amount of time. So we try to identify structure of solutions that can reduce the search space or indicate the best way to reach a best solution within a reasonable time. So, from one solution, we can find another, and so on. It is necessary to define a relation of neighborhood which is a function relating any solution of the search space to a neighborhood, that is to say a set of neighboring solutions. Local searches are therefore methods based on a neighborhood relation and on a procedure allowing this neighborhood to be explored. The local searches are differentiated by the procedure of neighborhood exploration. So the neighborhood function being able to be regarded as a parameter of this procedure.

Local descent method: The descent methods are classic and very fast. Algorithm 3 gives a global idea on the procedure exploiting the neighborhood [Papadimitriou et Harilaos, 1976] and [Papadimitriou et Steiglitz, 1982].

Algorithm 3 General Structure Local descent Methods

1: Let s be an initial solution from search space of solutions
2: Choosing a solution close to s' such that $f(s') \leq f(s)$
3: Changing s by s' and repeat (ii) until for any neighbor s' of s, $f(s) \leq f(s')$.
4: Return s which is a local optimum

There are several procedures for choosing a better neighbor, two of which are widely used.

- *Hill Climbing (HC)* consists of choosing the neighbor of the current solution with the best quality (exhaustive exploration of the neighborhood).

- *The First Improvement Hill Climbing (FIHC)* consists in choosing the first neighbor met that has a better quality (partial exploration of the neighborhood).

In [Papadimitriou et Harilaos, 1976] and [Papadimitriou et Steiglitz, 1982], we can find more details.

Iterated Local Search (ILS) [Papadimitriou et Steiglitz, 1982]: descent methods are fast and simple to implement but do not generally lead to the best optima because they stop as soon as a local optimum is found. To avoid getting stuck on this local optimum, there are different strategies to continue. These strategies make it possible to continue the search after having found an optimum, it is then necessary to define a stopping criterion. The current stop criteria are the execution time, the number of iterations or the total number of evaluations.

A first strategy to overcome the quick stop of the search on a local optimum is to iterate the method of descent. Algorithm 4 gives principal steps which are carried out starting from the optimum found.

Algorithm 4 Exploiting neighborhood

1: Apply a perturbation on the current solution
2: Apply a method of descent on this solution
3: Choose via an acceptance criterion if the new optimum becomes the current solution and return to (1) until the stopping criterion is reached
4: Return last best obtained solution

The perturbation may consist of:

- restarting from a solution taken randomly in the search space
- choosing a solution in a neighborhood far from the optimum
- choosing a neighbor of the same quality as the optimum
- using the strategies to balance of "two-faced Janus" intensification and diversification to avoid local optimal

When we use iterative improvement method, sometimes we meet the local optimal. In this case, we use some escaping strategies. Let $N(S)$ be a set of neighboring solutions in the search space. Formally, $N(S)$ is a subset of the search space made of all solutions obtained by applying a single local transformation to S. Bellow we recall principal escaping strategies:

- Accept up-hill moves: i.e., the search moves toward a solution with a worse objective function value by climbing the hills and go downward in another direction; They accept a temporary deterioration of the solution which allows them to explore more thoroughly the solution space and thus to get a hopefully better solution (that sometimes will coincide with the global optimum). However, necessary to do some fine-tuning of its intrinsic parameters in order to adapt the technic to the problem.

- Change neighborhood structure during the search by different neighborhoods. This can be done by generating different search space topologies, which could simply be the set of feasible solutions to the problem.
- Formally, N(S) is a subset of the search space made of all solutions obtained by applying a single local transformation to S.
- Change the objective function so as to "fill-in" local minima. So, they can modify the search space with the aim of making more "desirable" not yet explored areas.
- *Intensification* and *Diversification* procedures. These two procedures are the driving forces of metaheuristic search. Intensification explores the accumulated search experience (e.g., by concentrating the search in small search space area). In the other hand, *Diversification* explores "in the large" of the search space. They are contrary and complementary: their dynamical balance determines the effectiveness of metaheuristics.

2.4.4.2 Simulated Annealing

Simulated Annealing (SA) as it is known today is motivated by an analogy to annealing in solids. The idea of SA comes from a paper [Metropolis, 1953].

In 1983, took the idea of the Metropolis algorithm and applied it to optimization problems, SA was developed in [Kirkpatrick *et al.*, 1983]. It approaches the global maximization problem. The idea is to use simulated annealing to search for feasible solutions and converge to an optimal solution. One can refer to [Chibante, 2010] for this interesting theory and its applications.

In general, SA algorithm requires definition of initial solution, initial temperature, perturbation mechanism, cooling schedule, stopping criteria and of course objective(s) function(s). Algorithm 5 describes the general structure of this resolution approach.

- *Initial solutions*: AS requires the use of an initial solution, which can be calculated by a simple and dedicated heuristic or randomly generated.
- *Initial temperature*: The control parameter T must be carefully defined since it controls the acceptance rule defined by $e^{-\Delta/T}$ must be large enough to enable the algorithm to move off a local minimum but small enough not to move off a global minimum.
- *Perturbation mechanism*: this is defined by the neighborhood generation procedure.
- *Cooling schedule*: The cooling schedule was implemented using a rule for temperature variation. For example: Update $T_{i+1} = aT_i, 0 < a < 1$ after each iteration.
- *Stopping criterion*: Several methods to control the stopping of the algorithm may be implemented. we can use one or more strategies among the following:
  - maximum number of iterations;
  - minimum temperature value;

- minimum value of objective function;
- minimum value of acceptance rate.
- time limited.

Algorithm 5 Simulated Annealing algorithm

```
1: i := 0;
2: Let S_0 be the initial solution and let T_0 := T be the initial temperature;
3: S := S_0; S* := S_0; best := f(S_0);
4: while Stopping criterion is not satisfied do
5:     S' := Some random neighboring solution of S;
6:     if f(S') < best then
7:         S* := S' ;
8:         best := f(S')
9:     else
10:        if random[0,1] < min{1, e^((f(S)-f(S'))/T)} then
11:            S := S' ;
12:        end if
13:    end if
14:    T_{i+1} := λT_i (λ < 1, quite frequently λ = 0,99) ;
15:    i := i + 1
16: end while
17: return best solution
```

2.4.4.3 Tabu search

Tabu Search (TS) has been initially proposed by Glover [Glover et Laguna, 1997]. TS is a metaheuristic local search algorithm that begins with an initial solution and successively moves to the best solution in the neighborhood of the current solution. The algorithm maintains a list of forbidden solutions, to prevent the algorithm from visiting solutions already examined.

For certain optimization problems, the Tabu method gave excellent results []; moreover, in its basic form, the method comprises less parameters of adjustment than simulated annealing, which makes it easier to use. However, the various additional mechanisms, like the intensification and diversification, bring a notable complexity.

Some mechanisms defining TS algorithm are the same as used in SA (initial solution, perturbation mechanism, objective(s) function(s), stopping criteria). In next, we only recall additional and specific concepts of TS algorithm which different from SA algorithm such as tabu list and its size [Glover, 1989, Glover, 1990].

- Tabu list: record the recent history of the search, which are stored in a short-term memory of the search. Usually only a fixed and fairly limited quantity of information is recorded. In any context, there are several possibilities regarding the specific information that is recorded. Here are some ways to proceed:

  - record complete solutions, but this can require a lot of storage space and makes it expensive to check whether a potential move is Tabu or not (it is therefore seldom used);
  - record the last few transformations performed on the current solution and prohibiting reverse transformations (most commonly used).
  - record based on key characteristics of the solutions themselves or of the moves.
  - record by multiple tabu lists can be used simultaneously and are sometimes advisable. For example, when different types of moves are used to generate the neighborhood, it might be a good idea to keep a separate Tabu list for each type. Standard Tabu lists are usually implemented as circular lists of fixed length.

- Tabu size: It has been shown that fixed-length Tabus cannot always prevent cycling, and some authors have proposed varying the Tabu list length during the search.

Algorithm 6 gives the general TS algorithm where the used notations are:

- S_0 is an initial solution, S^* is the best solution, S is the current solution,
- f is the value of solution S, f^* is the value of S^* (best solution value found),
- $N(S)$ is the whole neighborhood of $S, N'(S)$ is the neighborhood of S which is not tabu,
- LT is the tabu list.

Algorithm 6 Tabu search algorithm

1: $S := S_0$;
2: $S^* = S_0$,
3: $f^* = f(S_0)$, (f^* = best)
4: Tabu list $LT = \emptyset$
5: **while** Stopping criteria not satisfied **do**
6: Select $S = min[f(S')]$, where S is a solution belonging to $N'(S)$,
7: **if** $f(S) < f^*$ **then**
8: $f^* = f(S)$
9: $S^* = S$
10: **end if**
11: Record the current move in the Tabu list LT (delete oldest entry if necessary)
12: **end while**
13: **return** S^*

2.4.4.4 Genetic algorithm

Genetic algorithms (GA) have been originally proposed by Holland [Holland, 1992]. This is a general search technique where a population composed by individuals evolves following nature inspired mechanisms called "genetic operators". The population is composed by individuals that are evaluated by a fitness, which is often related to the objective

function. Starting from an initial population, new solutions are generated by selecting some "parents" randomly, but with a probability growing with fitness, and by applying genetic operators such as selection, crossover and mutation, which introduce random modifications. Some existing solutions are randomly selected for crossover, some solutions are selected for mutation, and a new population of the same size is obtained. The process is repeated until a given stopping criterion is reached, e.g. a time limit or when a suffciently satisfactory solution has been found. A typical genetic algorithm requires:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover and selection operators. The main steps to implement GA are:

- *Coding*: The choice of coding is the first step in the implementation of GA, it must take into account the specificities of the problem studied, in order to guarantee a better application of the methods. In fact, each individual is represented by a chromosome describes a gene sequence. Initially Holland [Holland, 1992] used a sequence of bits containing all the information necessary to describe any solution from search space. In general, this representation is known as indirect coding and requires decoding function to describe solution.
- *Initialization*: The population size depends on the nature of the problem. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be found from a heuristic solution are likely to be found. with the key idea: give preference to better individuals, allowing them to pass on their genes to the next generation.
- *Selection*: During each successive generation, a portion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions.
- *The fitness function* is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always adapting objective function to find the effective populations.
- *Genetic operators*: The next step is to generate a second generation population of solutions from those selected through a combination of genetic operators: crossover (also called recombination), and mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents".

- **Crossover** is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. It is analogous to reproduction and biological crossover, upon which genetic algorithms are based. Crossover is a process of taking more than one parent solution and producing a child solution from them. By recombining portions of good individuals, this process is likely to create even better individuals.

- **Mutation** is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. In mutation, the solution may change entirely from the previous solution. Its purpose is to maintain diversity within the population and inhibit premature convergence. Hence GA can come to a better solution by using mutation.

Algorithm 7 gives the main steps of GA algorithm.

Algorithm 7 General genetic algorithm

```
1: randomly initialize population
2: determine fitness of each individual in the current population
3: repeat
4:     select parents from population
5:     perform crossover on parents creating population
6:     perform mutation of population
7:     determine fitness of each individual in the current population
8: until best individual is good enough or stopping criteria satisfied
```

2.4.4.5 Non-dominated Sorting Genetic Algorithm

NSGA (Non-dominated Sorting Genetic Algorithm) is a popular non-domination based genetic algorithm for multi-criteria optimization [Srinivas et Deb, 1994].

The objective of the NSGA algorithm is to improve the adaptive fit of a population of candidate solutions to a Pareto front constrained by a set of objective functions. The algorithm uses an evolutionary process with surrogates for evolutionary operators including selection, genetic crossover, and genetic mutation. The population is sorted into a hierarchy of sub-populations based on the ordering of Pareto dominance. A similarity between members of each sub-group is evaluated on the Pareto front, and the resulting groups and similarity measures are used to promote a diverse front of non-dominated solutions.

NSGA is a very effective algorithm but has been generally criticized for its computational complexity, lack of elitism and for choosing certain parameter values. To overcome some of these criticisms, Deb et al. propose another version of NSGA called NSGA II.

NSGA II is a modified NSGA, which has a better sorting algorithm, incorporates elitism and no sharing parameter needs to be chosen a priori (Deb et al. 2002).

The population is initialized as usual. Once the population in initialized the population is sorted based on non-domination into each front. The first front being completely non-dominant set in the current population and the second front being dominated by the

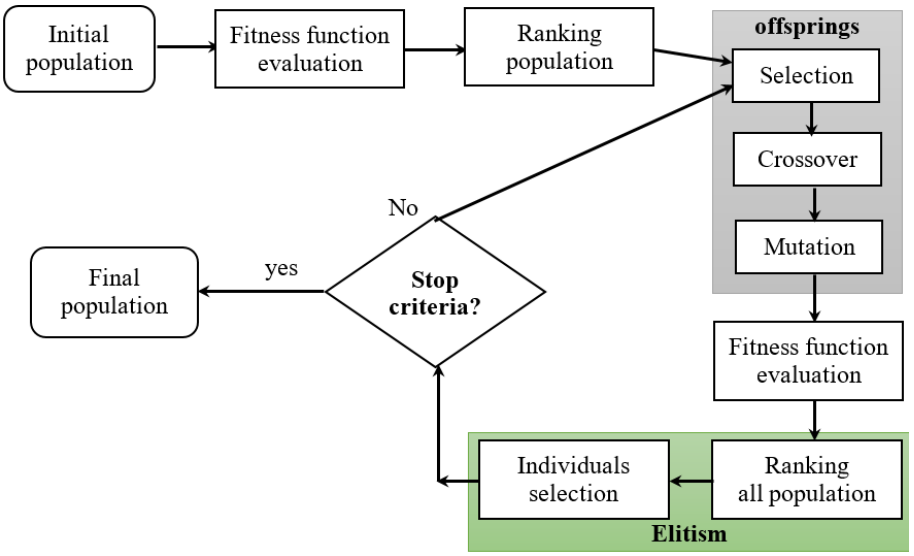

Figure 2.3: Diagram representing the NSGA-II

individuals in the first front only and the front goes so on. Each individual in each front is assigned rank (fitness) values or based on the front in which they belong to.

Individuals in the first front are given a fitness value of 1 and individuals in second are assigned fitness value as 2 and so on. In addition to fitness value, a new parameter called crowding distance is calculated for each individual. The crowding distance is a measure of how close an individual is to its neighbors. Large average crowding distance will result in better diversity in the population. Parents are selected from the population by using binary tournament selection based on the rank and crowding distance. An individual is selected in the rank is lesser than the other or if crowding distance is greater than the other [Beyer et Deb, 2001]. The selected population generates offsprings from crossover and mutation operators, which will be discussed in detail in a later section. The population with the current population and current offsprings is sorted again based on non-domination and only the best Pop_{size} individuals are selected, where Pop_{size} is the population size. The selection is based on rank and the on crowding distance on the last front.

The general model of the NSGA II is presented by Figure 2.3.

2.4.5 Matheuristic

Matheuristics are optimization algorithms made by the interoperation of (meta)heuristics and mathematical programming (MP) techniques. An essential feature is the exploitation in some part of the algorithms of features derived from the mathematical model of the problems of interest [Marco et Voss, 2010], [Talbi, 2013].

Since several years, a new type of approximated algorithms, including exact resolution inside heuristic approaches, has received a great interest in the literature, because of their very best performances on some difficult problems [Kergosien *et al.*, 2017]. These methods are also called "Hybridizing metaheuristics and mathematical programming" (MP) in

[Marco et Voss, 2010].

Exact MP algorithms are known to be time and/or memory consuming. We cannot be applied it to large instances of difficult optimization problems because their completion time is very long. However, we can their combination with metaheuristics may improve the effectiveness of heuristic search methods to getting better solutions.

Sometimes, this type of combination allows finding optimal solutions in a shorter time by using the design of more efficient exact methods.

As introduced in [Talbi, 2013], metaheuristics and exact algorithms are complementary optimization strategies in terms of the quality of solutions and the search time used to find the better solutions. In the last few years, solving exactly important optimization problems using integer programming techniques has improved dramatically. Moreover, the availability of efficient optimization software, frameworks for mathematical programming and high-level modeling languages will lead to more hybrid approaches effective combining metaheuristics and exact optimization algorithms more effective.

Depending on the implementation and the level of hybridization achieved through the interoperation of metaheuristics and exact methods, different matheuristics can be derived. Thus, following a classification of hybrid metaheuristics in terms of design problems introduced in [Talbi, 2013], we can propose a similar classification for matheuritic algorithms. As introduced by Talbi, at the first level, two types of hybridations can be defined: *low-level* and *high-level* hybridations. With low-level hybridation, only one metaheuristic is used and is combined with the exact method. In the case of high-level hybrid algorithm, different metaheuristics can be implemented and each is executed independently of the other. From this first level, a second level can be defined as follows: *Relay hybridization* and *Teamwork hybridization*. With Relay hybridization, a set of metaheuristics is applied on after another, each using the output of the previous as its input, when Teamwork hybridization represents cooperative optimization models, in which each metaheuristic evolve in parallel to carrie out a search in a (dedicated) solution space.

2.4.5.1 Low-Level Relay Hybrids (LRH)

This class of algorithms represents hybrid schemes in which a S-metaheuristic approach (single solution based metaheuristic) is embedded into an exact approach or an exact approach is embedded into an S-metaheuristic approach to improving the search strategy.

- **Embedding S-metaheuristic into exact algorithms**: Metaheuristic can involve in the design of some search components of an exact method of type branch and bound such as the node selection strategy, generation of an upper bound, cutting plane generation and pricing (see Figure 2.4).

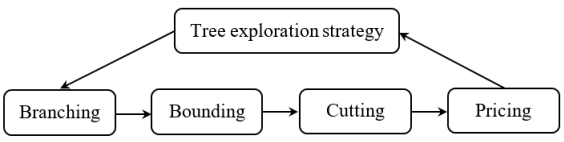

Figure 2.4: Branch and X algorithm

- **Embedding exact algorithms into S-metaheuristics**: We can use many combinations may be designed in which exact algorithms (Branch and bound, Dynamic programming, MILP) are embedded into search components of S-metaheuristics. It means tnat to find the best solution of the problem, we generate and search large neighborhoods. Partial exact algorithm is then applied on some of them for try to find an effective solution (see Figure 2.5).

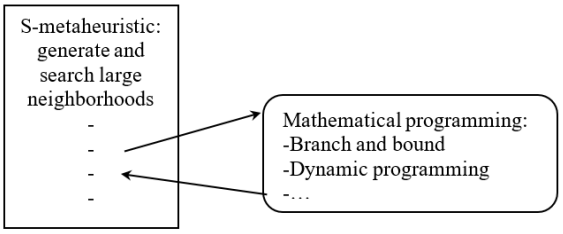

Figure 2.5: Combine S-metaheuristic and Mathematical programming

2.4.5.2 Low-Level Teamwork Hybrids (LTH)

The idea of this method is that the search component of a P-metaheuristic (Population based metaheuristics) is replaced by another optimization algorithm. With regard to the combination of P-metaheuristics and mathematical programming algorithms, two main hybrid approaches can be considered: hybrid exact search algorithms in which a P-metaheuristic is integrated into an exact algorithm, and heuristic search algorithms in which an exact algorithm is integrated into a P-metaheuristic (see Figure 2.6).

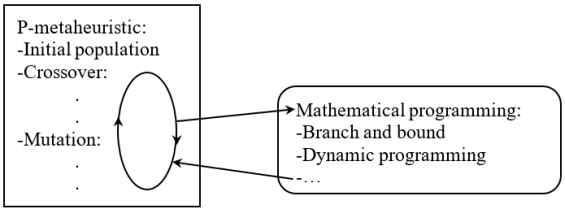

Figure 2.6: Mathematical programming is embedded into a P-metaheuristic

2.4.5.3 High-Level Relay Hybrids (HRH):

From this class, resolution methods are ordered and each one is used in sequence, defined as an autonomous algorithm algorithm. This can be considered as a pre-processing or a post- processing step (see Figure 2.7).

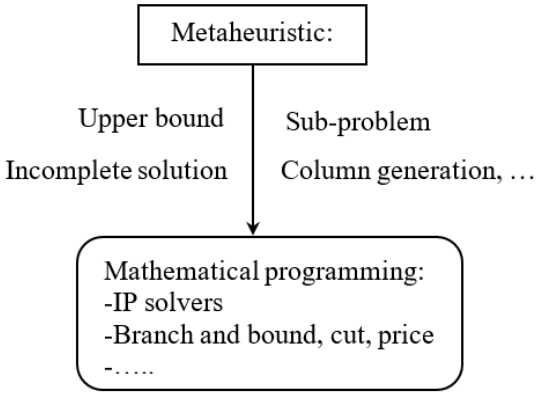

Figure 2.7: Sequence of combine metaheuristic and mathematical programming

2.4.5.4 High-Level Teamwork Hybrids (HTH)

Here the emphasis is on parallel cooperation. Since each method (metheuristic and PM) solves global or partial optimization problems, the basic idea is that these two methods are applied in parallel and that information exchanges take place regularly. For example, a possible implementation of such an algorithm is as follows: the input of the metaheuristic is an output of a sub-problem to be solved by mathematical programming, or a solution obtained by metaheuristic defines an upper bound for the problem to be solved by mathematical programming (see Figure 2.8).

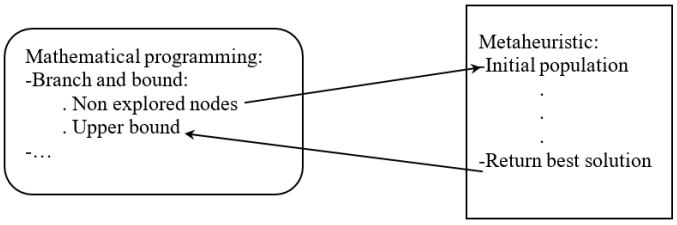

Figure 2.8: Parallel cooperative between mathematical programming and metaheuristic

It should be noted that the majority of the proposed hybridization approaches are also referred to as partial hybrids. Indeed, the research space is generally too large to be solved by an exact approach. Thus, the different algorithms implemented are seen as complementary and allow different problems to be solved. Sharing any information gathered during the research process thus improves the effectiveness and efficiency of the hybrid approach. Exchanged informations can be: solution, relaxed optimal solution and its dual, upper bound, lower bound, optimal solution of subproblem, partial solution, etc. (see Figure 2.9).

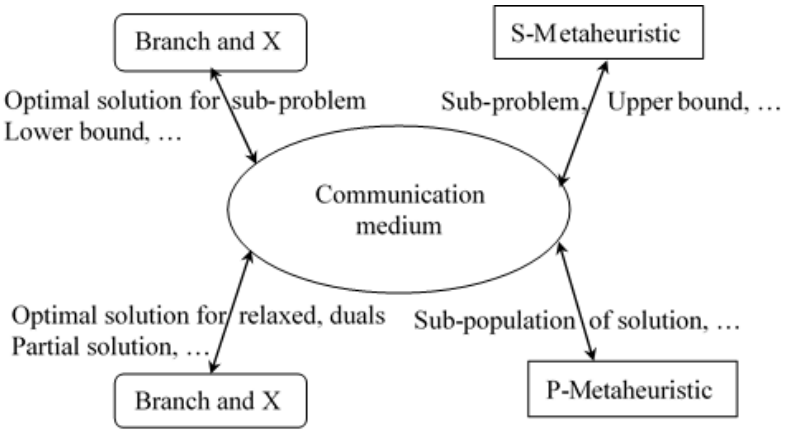

Figure 2.9: HTH cooperation between metaheuristic and mathematic programming

2.5 Performance analysis

The goal of multi-criteria optimization problem is to find a set of best compromise solutions for typically conflicting objectives. Due to the complex nature of most real-life problems, only an approximation to such an optimal set can be obtained within reasonable (computing) time. The performance of an approximation method is measured by the quality of Pareto's front determined by it. To compare such approximations, and thereby the performance of multi-criteria optimizer providing them, generally, a single metric is not enough to do this. Hence, several performance measures are proposed in the literature, some of which are designed to evaluate a distance between the approximate front and the exact front. Others are based solely on the approached front. In this thesis, we used four below measures to compare these results.

2.5.1 Hypervolume

The performance assessment of algorithms for multi-criteria optimization problem is far from being a trivial issue. Recent results indicate that unary performance measures, i.e performance measures which assign a single value to each optimal point set, are inherently limited in their inferential power. Despite these limitations, the hypervolume indicator (also known as Lebesgue measure or S metric) is still considered to possess some reasonable properties, having also been proposed as a guidance criterion for accepting solutions in multiobjective Evolutionary Algorithms, for example [Beume *et al.*, 2009]. Therefore, the computational time taken for computing the hypervolume indicator is a crucial factor for the performance of such algorithms.

Hypervolume (H) is hence a metric, which was introduced in [Zitzler et Thiele, 1998], and in a two-criteria case is defined as follow. For each solution x whose value is given by $(f^1(x), f^2(x))$, we calculates the area of the rectangle defined by the origin (0.0) and the point $(f^1(x), f^2(x))$. The sum of these surfaces gives us the totality of the covered surface. The difference of the hypervolume calculated for two Pareto fronts, is an indicator

of distance between two fronts (see Figure 2.10).

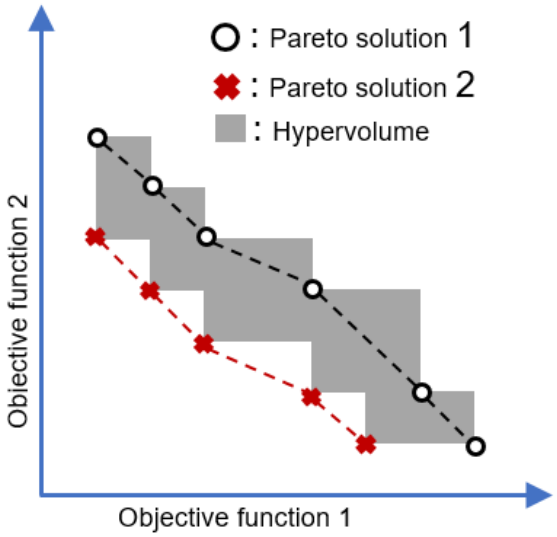

Figure 2.10: Hypervolume measures representations

Other variant of this measure are proposed in [Jarboui *et al.*, 2013]. The authors calculate the area dominated by a Pareto front where the origin is replaced by a reference point, which preferably corresponds to the ideal point (see Figure 2.1). These measures are obviously easily adaptable to problems of more than two dimensions.

2.5.2 Average distances

GD is a generational distance, and it is used to calculate the average of the minimum Euclidian distances between a front of exact Pareto and his approximate front [Cyzak et Jaszkiewicz, 1998]. Let $\mathcal{S}^*$ (rest. $\mathcal{S}$) be the exact (rest. approximate) Pareto front. The *GD* is given by the following formula:

$$GD = \frac{1}{|\mathcal{S}|} \sum_{S_1 \in \mathcal{S}} min_{S_2 \in \mathcal{S}^*} d_{s_1,s_2}$$

where d_{s_1,s_2} is the Euclidian distance between the element $S_1 \in \mathcal{P}$ and the element $S_2 \in \mathcal{P}^*$.

This distance is also used to measure the dispersion of points over the research domain. In fact, the approximate front must be a representative set of the of the exact front, so the solutions must be well distributed over the search space, not located and grouped in one place. The generational distance is applied to the approximate front. For each solution, the minimum distance to the others is calculated and the average of these distances gives then a an estimator of the dispersion. It is also possible to calculate the minimum distance between a point of approaches solution and an exact solution.

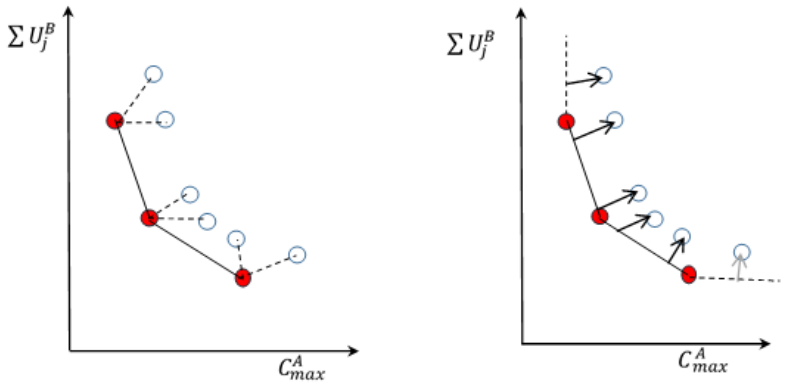

Figure 2.11: Minimum distance and Average distance

2.5.3 Number of non-dominated solutions

The two first metrics can give values equal to zero. In this situation, it does not necessarily mean that the obtained front is an exact front. Indeed, if only part of the effective region is reached, then the solutions overlap with the exact solutions, but some compromise solutions are not achieved. In this case, it is important to compare the size of the front obtained with the exact size of the front, in terms of number of points.

To do that, we apply two measures to evaluate a the number of points:

1. For each exact and approximate solution, we calculate the cardinalities of the generated Pareto fronts. We denote by $|\mathcal{S}^*|$ the cardinality of the exact Pareto front $\mathcal{S}^*$, and by $|\mathcal{S}|$ the cardinality of the near Pareto front $\mathcal{S}$.

2. We define a percentage of optimal solutions ($\%\mathcal{S}$) by using the cardinality. This metric calculates the number of exact solutions generated by each heuristic, it is given by $|\mathcal{S} \cap \mathcal{S}^*|/|\mathcal{S}^*|$.

2.6 Conclusion

This chapter is the introduction to the context of our research and gives basic concepts and definitions for studied multi-agent scheduling problems, which will be described in next chapter. In fact, the studied scheduling problems constitutes a part of the theory of multi-criteria optimization. Therefore, this chapter leads to define the necessary notions for multi-criteria analysis.

This chapter composed of four sections preceded by an introduction. In section 2.2, after recalling the general definition of a multi-criteria optimization problem, we focused on the presentation of the notion of non-dominance as well as the structure of all non-dominated solutions of a multi-criteria problem. This set is also referred to as the Pareto front. Multi-criteria approaches are presented in section 2.3, and details are given for approaches used in our study to solve scheduling problems. Besides, we present a briefly

overview of resolution methods (exact and approximates methods), some of them being used in the rest of this manuscript. The last part of this section is devoted to *matheuristic* methods. A discussion on the way and level of hybridization of such methods led us to propose a new classification, which takes up the same ideas proposed in the literature for hybrid metaheuristic approaches.

The performance of an approximation method is measured by the quality of Pareto's front determined by it. To compare such approximations, and thereby the performance developed algorithms, we recall some performance measures in section 2.5.

Next chapter allows us to get to the heart of the matter, that of studying "*multi-agent scheduling problems*".

Chapter 3

Scheduling theory and multi-agent scheduling problems

Résumé : Ce chapitre introduit un aperçu général des problémes d'ordonnancement, en particulier des problèmes d'ordonnancement multi-agents. Quelques définitions, exemples et notations utilisés tout au long de ce manuscrit sont ensuite introduits pour en faciliter la lecture de ce document. Dans un premier temps, la section 3.1 rappelle les concepts de base et les notations des problèmes d'ordonnancement, où la section 3.2 est consacrée aux problèmes classiques d'ordonnancement multicritères.

Pour situer nos travaux de recherche, une synthése des problèmes de planification multi-agents est présentée dans la section 3.3, en établissant des liens et soulignons des différences entre les scénarios étudiés dans la littérature. Notre travail se concentre sur une classe particulière de problèmes d'ordonnancement multi-objectifs, dite "*Problème d'ordonnancement multi-agent non-disjoint*". Nous exposons quelques applications réelles montrant ainsi nos motivations d'étudier cette classe de problèmes d'ordonnancement. Les principaux résultats traitants des problèmes proches de ceux qui font l'objet de notre étude sont ensuite présentés dans la section 3.4.

Abstract: This chapter is devoted to a general overview of scheduling problems, in particular to multi-agent scheduling problems. Some definitions, examples and notations used throughout this manuscript are then introduced for ease of reading. At first, Section 3.1 recalls basic concepts and notations of scheduling problems, where Section 3.2 is dedicated to the classical multi-criteria scheduling problems.

To situate our research works, a synthesis of multi-agent scheduling problems is presented in Section 3.3, drawing links and differences between the different studied scenarios in the literature. Our work focuses on a particular class of multi-objective scheduling problems, denoted "*Non-Disjoint multi-agent scheduling problem*". We hence give some real-applications showing our motivations to study this class of scheduling problems. Main results related to the studied problems are then presented in Section 3.4.

3.1 Classical scheduling problems: mono-criterion objective function

Scheduling problems are an important part of combinatorial optimization problems. These problems are encountered in any operating system when it comes to organizing activities or tasks over time and determining their best allocation(s) to consumable or renewable resources [Carlier et Chrétienne, 1988]. Hence, many definitions of a scheduling problem have been introduced in the literature. In [Morton et Pentico, 1993] authors have defined scheduling as follows: *"[...] scheduling is the process of organizing, choosing, and timing resource usage to carry out all the activities necessary to produce the desired outputs at the desired times while satisfying a large number of time and relationship constraints among the activities and the resources."* Another definitions have been introduced by researchers. For example, Du and Leung defined scheduling as follows: *"Scheduling is concerned with the allocation of scarce resources to activities with the objective of optimizing one or more performance measure."* [Du et Leung, 1990]; Where Pinedo's definition is: *"Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives"*. He also showed the importance of the sequencing and scheduling problems: *"scheduling [...], plays an important role in most manufacturing and production systems as well as in most information processing environments. It is also important in transportation and distribution settings and in other types of service industries"* [Pinedo, 2016].

Thus, the theory of scheduling is quite broad because of the diversity of systems and activities to be planned over time [Lopez et Roubellat, 2008], [Oulamara *et al.*, 2005, Oulamara *et al.*, 2009]. Reference can be made to classic books presenting classical models, more recently studied models and also resolution algorithms. As reference books dedicated to single criterion scheduling problems, we cite Blazewicz et al. [Blazewicz *et al.*, 2007], Brucker [Brucker, 2007] and Pinedo [Pinedo, 2008]. These reference books address the classic cases of scheduling problems.

Scheduling problems are very varied. They can be found in many areas as recalled by Sadi in [Sadi *et al.*, 2015]:

- Flexible Manufacturing Systems (FMS) [Lee et Kim, 1999]: these systems are designed to produce in small and medium quantities a wide variety of products included time preparations of machines. Their aim is to achieve not only high productivity, but also a high production flexibility allowing it to follow the variations of the demands.
- Computer system [Cordeiro *et al.*, 2011]: in such a system, scheduling refers to a component of the operating system that determines the order for running processus on processors.
- Real-time systems [Navet, 2006, Norre, 1993]: these systems are generally "embedded", that is, located within the very environment with which they must interact, like a calculator in vehicles or planes. In addition to the requirements of time performance, they are then subject to severe congestion constraints, of cost and sometimes energy.

- Communication networks [Peha, 1995]: achieving quality of service in such a system is dependent on the choice of configuration and implementation task scheduling strategies that allow remote entities of the same system to exchange information.

3.1.1 Concepts and scheduling notations

The great variety of scheduling problems we have seen from the definition motivates the introduction of a systematic notation that could serve as a basis for a classification scheme. Such a notation of problem types would greatly facilitate the presentation and discussion of scheduling problems. Hence, Graham et al. proposed three fields notation for scheduling problems $\alpha|\beta|\gamma$ [Graham *et al.*, 1979].

1. The first field α describes the processor environment. In general, $\alpha = \{\emptyset, P, Q, R, O, F, J\}$ characterizes the type of used processors (resources). Resources are "machine or human" means that can carry out all the jobs to be executed, and are available in limited quantities. They are classified according to their capabilities or operating modes. Two types of resources can be identified:

   - Non-dedicated machines: this means that a job can be executed on any machine, and it is defined by only one operation. Depending on the performance of the machines for job processing, we denote by: $\alpha = P$ identical parallel machines; $\alpha = Q$ uniform paralle machines; $\alpha = R$ unrelated parallel machines.
   - Dedicated machines: in this case, a job is defined by a set of operations in which each operation should be executed on a specific machine. According to the production routing, we denote by: $\alpha = F$ flow-shop system (all jobs have the same operating routing and the same execution order); $\alpha = J$ job-shop system (each job is defined by its operation routing); $\alpha = O$ open-shop (each job is defined by its operating routing but the execution order is not imposed).

   In general, a resource is referred to be "renewable" if, after performing a given job, it is available in the same quantity. It is consumable, if its use is limited to only a certain number of jobs: for example, raw materials, money, etc. We also distinguish disjunctive resources, which can only perform one job at a time, otherwise they are called cumulative. In the rest of this manuscript we have an environment with m identical parallel machines. We assume that these disjunctive resources are always available. Two different environments are considered in this study as follows.

   - Single machine. In such a model, the set of tasks to be performed is executed by a single machine [Karp, 1975]. We can find this kind of configuration in a production system including a bottleneck machine that disrupts the whole system of the process. Hence, in α-field we have: $\alpha = 1$.
   - Identical parallel machines. This model considers a set of identical parallel machines, the processing times of the jobs are independent of the machines. Each job must be performed on one or more machines. No overlapping is allowed, i.e., work cannot be performed on two machines at the same time. This is a type of workshop that is common in industry, it allows to multiply the speed

of execution of a given step in the manufacturing process. Hence, in α-field we have: $\alpha = P$.

2. The second field β describes jobs and resource characteristics, i.e. "constraints". The constraints encountered in scheduling problems are classified into the following classes:

   (a) Resource constraints: several types of constraints can be induced by the nature of the resources, for example: the limited capacity of a resource implies a constraint on the amount of jobs to be allocated to them; if the resource is cumulative, the constraint can limit the number of jobs to be performed simultaneously. A resource may also not be available in certain time windows (unavailability constraints), or not valid for performing given work (disability constraints).

   (b) Procedural constraints: they can be linked to production ranges, for example:
   - Precedence constraints (*prec*): tthey express the relationships between the tasks to be performed; a task cannot be performed before the end of its predecessors' execution.
   - Disjunctive constraints: two jobs are in disjunction if they cannot be executed simultaneously.
   - Preemption (*pmtn*): Preemption is allowed, if the execution of the work can be interrupted on a given date and then resumed and potentially on a different machine.
   - $p_j = p$ for cases where the job has the same operating time.

   (c) Time constraints: they represent restrictions on the values that can be take certain scheduling time variables: release date, date due or deadline.

3. The third field γ describes optimality criterion (performance measure). The definition of objectives to measure the quality of a scheduling is an important and even delicate fact. Objectives can be linked to production costs: minimization of production times, minimization of lates, maximization of production rates, etc. The classical objectives are those related to the completion dates of jobs. Because, these criteria are related to a better use of the production system, which inevitably minimizes production costs. Among these classical criteria we have: $\gamma = \{C_{max}, \sum C_j, \sum w_j C_j, L_{max}, \sum U_j, \sum w_j U_j, \sum T_j, \sum w_j T_j\}$

   - "Makespan", denoted by C_{max}, it defines the maximum completion time of jobs (last date of execution of the last job).
   - "Total completion time", represents the average waiting time for jobs to be achieved, it is denoted by $\sum C_j$ where C_j is the job completion time of job j.
   - "Delay", this measure is very important in practice, it allows to meet customers' requirements in terms of time. We can define function $L_j = C_j - d_j$ that is associated with the algebraic delay of job J_j, and function $T_j = maxC_j - d_j, 0$ that is associated with the absolute tardy of job J_j. The criteria L_{max} and T_{max} are defined for maximum lateness/tardiness, or $\sum T_j$ for the total tardiness. It

may happen that jobs do not have the same weight (degrees of importance of one job compared to another). In this case, the jobs are also identified by weights denoted w_j. The objective functions in this case can be: $\sum w_j T_j$ for the total weighted tardiness

- "Number of (weighted) tardy jobs" is denoted iby $\sum(w_j)U_j$, U_j, such that U_j is a binary function: $U_j = 1$ if job J_j is late and 0 otherwise.

3.1.2 Complexity of scheduling problems

The complexity of an algorithm lies in estimating its processing cost in time (time complexity) or in the required space memory (spatial complexity). Set apart for certain particular algorithms, as for example dynamic programming algorithms which usually take up a lot of memory space, spatial complexity has been less considered than time complexity. In both cases, it is possible to propose a theoretical complexity and a practical complexity. Theoretical complexity reflects an independent estimate on the machine which processes the algorithm. It is less accurate than the practical complexity which enables us to calculate the cost of the algorithm for a given computer. For the latter case, time complexity is obtained using an estimation of the calculation time for each instruction of the program. The advantage of theoretical complexity is that it provides an estimation independent of the calculation time for the machine [T'Kindt et Billaut, 2006]. According to Brucker, *complexity theory provides a mathematical framework in which computational problems are studied so that they can be classified as "easy" or "hard"* [Brucker, 2007].

This classification is useful to see if an efficient algorithm may exist, especially in terms of computation time, for solving a particular problem. A problem belongs to a class of complexity, which informs us of the complexity of the "best algorithm" which is able to solve it. Hence, if a given problem is shown to belong to the class of "easy" problems, it means that it exists a polynomial-time algorithm to solve it. Hence, this problem is in $\mathcal{P}$. Usually this is good news but unfortunately, this does not often happen for complex problems. Accordingly, if a problem belongs to the class of "hard" problems, it cannot be solved in polynomial time, i.e. the required CPU time to solve it becomes "exponential" [Brucker, 2007]. In this case, the problem is in $\mathcal{NP}$. We refer to [Garey *et al.*, 1976] and [Papadimitriou Christos H. , 1994] for the definition of complexity classes and more details on computational complexity. Also, More classification of scheduling problems can be found at *http://www.mathematik.uni-osnabrueck.de/research/OR/class*).

The classical criteria of scheduling problems presented previously are regular because they are non-decreasing functions of the completion times of jobs. Hence, in terms of complexity status, these criteria are linked by reduction relationships summarized in Figure 3.1.

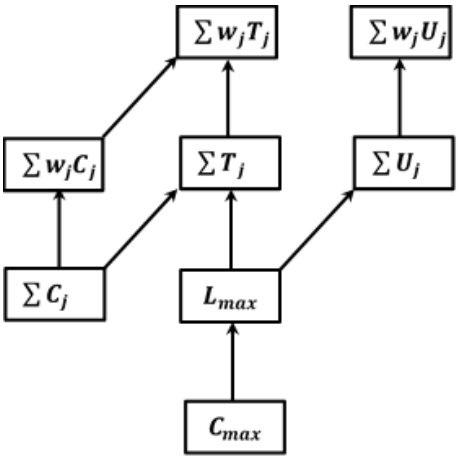

Figure 3.1: Criteria reduction relationships

3.1.3 Some classical scheduling algorithms

In this section, we will recall some algorithms developed to solve classical mono-criteria scheduling problems. These algorithms form a basis for our algorithmic developments to the problems studied in this manuscript.

3.1.3.1 Dynamic programming algorithm

Let's consider the scheduling problem: minimizing makespan on two identical parallel machines, denoted $P2||C_{max}$. This problem is shown to be $\mathcal{NP}$-hard in ordinary sens. In fact, an optimal solution can be computed in pseudo-polynomial time by applying the following dynamic programming algorithm proposed by [Huynh Tuong *et al.*, 2011]:

- We define P as sum of processing times $P = \sum p_j$, hence P is an upper bound of C_{max}
- Let $i = 1, 2$ be the number of machine; Let $t_1 = 0, 1, \ldots, P$ possible completion times of last job executed on machine M_1
- The recursion formula is then:

$$f(j, t_1) = min\Big(f(j-1, t_1 - p_j), f\big((j-1, t_1) + p_j)\big)\Big)$$

  where $f(j, t_1)$ corresponds to the completion time of the last job executed on machine M_2. The initial value is: $f(j, t_1) = \infty$ if $t_1 < 0$
- $C_{max} = min(max(t_i, f(j, t_1)))$

  The running time of this dynamic programming is then $O(nP)$.

Suppose that we have 5 jobs to be scheduled on two machines. The data are:

Job(j)	1	2	3	4	5
Processing time (p_j)	2	3	1	4	2

$P = \sum p_j = 12; t_1 = 0, 1, \ldots, 12;$

The next table gives the different values of $f(j, t_1)$

t_1	0	1	2	3	4	5	6	7	8	9	10	11	12
$f(1, t_1)$	2	2	0										
$f(2, t_1)$	5	5	3	2	2	0							
$f(3, t_1)$	6	5	4	3	2	1	0						
$f(4, t_1)$	10	9	8	7	6	5	4	3	2	1	0		
$f(5, t_1)$	12	11	10	9	8	7	6	5	4	3	2	1	0
$max(t_1, f(j, t_1))$	12	11	10	9	8	7	6	7	8	9	10	11	12

$C^*_{max} = min(max(t_1, f(j, t_1))) = min(12, 12, ..., 8, 6, 7, ..., 12) = 6$. Figure 3.2 illustrates an optimal schedule.

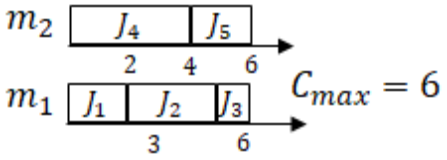

Figure 3.2: An optimal solution for $P2||C_{max}$

3.1.3.2 Mathematical programming

Let us consider the scheduling problem $P2||C_{max}$. MILP based assignent variables can be defined as follow [Garey *et al.*, 1976]:

Let x_{ij} be a binary variable that takes value 1 if job J_j is scheduled on machine m_i; 0 otherwise. Hence, we have:

Minimize C_{max}

$$s.t \begin{cases} \sum_{i=1}^{m} x_{ij} = 1 & \forall j, j = 1, \ldots, n \quad (1) \\ \sum_{j=1}^{n} p_j x_{ij} \leq C_{max} & \forall i, i = 1, \ldots, m \quad (2) \\ x_{ij} \in \{0, 1\} & \forall i, i = 1, \ldots, m, \forall j, j = 1, \ldots, n \quad (3) \end{cases}$$

Constraints (1) indicate that a job j is assigned to exactly one machine; Constraints (2) indicate that the C_{max} is greater than or equal to the completion time of all jobs on each machine (makespan of each machine).

3.1.3.3 List scheduling methods

Scheduling problems are in general $\mathcal{NP} - hard$. Hence, important and more efficient algorithms were developed and are based on "priority rules", called list scheduling methods. Note that heuristic based on priority rules is a $(2 - \frac{1}{m})$-approximation for scheduling problem $P||C_{max}$. In the following we recall well known of them.

1. ***Earliest Due Date (EDD)***: The earliest due date rule sorts jobs according to the non-decreasing due dates order. It means that the job with the earliest due date should be executed firstly. In general, this rule is more effective when it comes to optimization criteria related to minimizing delay costs.

   Example: Let's consider the classical scheduling problem minimizing the total number of tardy jobs on single machine, denoted $1|d_j|\sum U_j$. Suppose that we have to schedule 5 jobs defined as follow:

Job(j)	1	2	3	4	5
Processing time (p)	7	6	3	1	5
Due date (d)	8	7	5	4	6

   To solve this problem, we call Moor's algorithm [Moore, 1968] (see Algorithm 8).

Algorithm 8 Moore's algorithm

1: Compute the tardiness for each job in the EDD sequence. Set $N_T = 0$, and let k be the first position containing a tardy job. If no job is tardy go to step (4).
2: Find the job j with the largest processing time in positions 1 to k
3: Remove job j from the sequence, set $N_T = N_T + 1$, and repeat Step(1)
4: Place the removed N_T jobs in any order at the end of the sequence
5: return the obtained sequence (this sequence minimizes the number of tardy jobs)

   By applying Moore's algorithm on sorted jobs according to EDD order, we obtain an optimal solution with $\sum U_j = 3$ (see Figure 3.3).

Job(j)	4	3	5	2	1
Processing time (p)	1	3	5	6	7
Due date (d)	4	5	6	7	8
Tardy jobs (U)	0	0	1	1	1

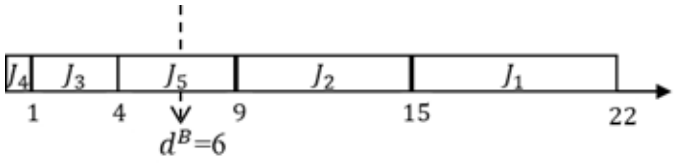

Figure 3.3: Optimal solution for $1|d_j|\sum U_j$

2. ***Shortest Processing Time (SPT)***: The shortest processing time rule sorts jobs according to the non-decreasing processing times order. Whenever a machine can process one job, the shortest job ready at the time will begin processing. For example, an algorithm based on this rule is optimal for finding the minimum total completion time where n jobs should be executed on a single machine without any additional constraints.

3. ***Longest Processing Time (LPT)***: The longest processing time rule sorts jobs according to the non-increasing processing times order. Whenever a machine can

process one job, the largest job ready at the time will begin processing. For example, to solve m identical parallel machines minimizing makespan, a heuristic based on this rule gives a solution with good gap from an optimal solution and it is a $\frac{4}{3}$-approximation heuristic.

4. ***First Available Machine (FAM)***: This rule consistes of assigning jobs to the first available machine. This rule is more efficient when dealing with regular criteria.

Example: The LPT&FAM-heuristic leads to a $(\frac{4}{3} - \frac{1}{3m})$-approximation for problem $P||C_{max}$, and the bound $(\frac{4}{3} - \frac{1}{3m})$ is tight (see [Alharkan, 1997]).

Recall the previous example with 5 jobs to be scheduled on two identical machines. By applying LPT&FAM-heuristic, we obtain an optimal schedule as presented in Figure 3.2:

- Step1: Jobs are sorted as follow: J_4, J_2, J_1, J_5, J_3
- Step2: Jobs are assigned to machines as follow:

Machine	M_1		M_2		
Job(j)	4	5	2	1	6
Processing time (p_j)	4	3	2	2	1
Completion time (C_j)	4	6	3	5	6

Hence, we have an optimal solution with $C_{max} = max_{(j=1,...,5)}C_j = 6$.

However, by considering a second example we can show that LPT&FAM-heuristic cannot each time find an optimal solution.

Job(j)	1	2	3	4	5
Processing time (p_j)	3	3	2	2	2

Applying LPT&FAM-heuristic we have:

Machine	M_1			M_2	
Job(j)	1	3	5	2	4
Processing time (p_j)	3	2	2	3	2
Completion time (C_j)	3	5	7	3	5

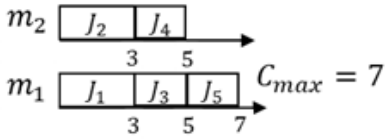

Figure 3.4: LPT&FAM-heuristic minimzing makespan

We obtain $C_{max} = max_{(j=1,...,5)}C_j = 7$. This is not optimal solution. In fact, an optimal schedule is given by schedule jobs J_1 and J_2 on machine M_1 and the remaining

jobs are scheduled on the second machine. In this case, the optimal value of makespan is $C^*_{max} = 6$.

Practical situations in scheduling rarely correspond to the optimization of a single objective. They are generally of a multi-objective nature inherent in the organization's various internal performance measures and customer requirements. Hence, let us introduce multi-criteria scheduling problems and in particular, the main of our study which is dealing with multi-agent scheduling problems.

3.2 Multi-criteria scheduling problems

When we look at the problems of multi-criteria scheduling problems where several objective functions mesure the quality of scheduling, here too the theory of scheduling has received great interest [T'Kindt et Billaut, 2006], [Rios-Mercado et Rios-Solis, 2012] and [Jozefowska, 2007]. Multi-criteria scheduling problems consider several objective functions to be optimized, each job to be scheduled is subject to them. More formally, the f^k functions to be minimized depend on the vector C of the completion times of jobs, $C = (C_1, \ldots, C_n)$. Under these conditions, the definition can be given by:

$$Minimizing F(C) = (f^1(C), \ldots, f^k(C), \ldots, f^K(C))$$
$$\text{s.t. } S \in \mathcal{S} = \{\text{Any sequence verifying } g(C) = b\}$$

Multi-criteria scheduling problems require a different analysis than mono-criterion problems and the methods applied to obtain Pareto solutions must be adapted to their specificities. In fact, in multicriteria scheduling area, if we find good solutions with respect to one objective may be bad with respect to other objectives. Therefore, we must find solutions of good compromise of the objectives. Smith [Smith, 1956] is the first publication considering this type of problem where the author studied a single machine scheduling problems minimizing two criteria: total completion times as well as the maximum lateness. Several states of the art are proposed later in the literature and the authors propose different classifications. We can mention the work of Dileepan and Sen [Dileepan et Sen, 1988] which deals with problems of two-criteria scheduling on a single machine. In [Nagar *et al.*, 1995], the authors propose a state of the art on multi-criteria scheduling problems. In this work, the results dealing with the case of single or parallel machines for the minimization of two or more regular criteria are presented. In [Hoogeveen, 1996] the author was interested in multi-criteria scheduling problems on a single machine and with several objective functions such as min-max. When it comes to the bi-criteria problem $1||P(f_{max}, g_{max})$, the enumeration of the Pareto front is shown polynomial and can be computed in $O(n^4)$ running time. Hoogeveen's algorithm is based on Lawler's procedure [Lawler, 1973]. This result is extended to the case of three criteria $1||P(f_{max}, g_{max}, h_{max})$. in this last case, the proposed algorithm is of complexity $O(n^8)$. The author also shows that when the number of criteria is not fixed, the problem is $\mathcal{NP}$-hard in the strong sense. In [Hoogeveen, 2005] the author presents the most important results on the complexity of multi- criteria scheduling problems, more precisely, on the criteria minimizing the tardiness and earliness of jobs. A fairly complete state of the art is proposed in [T'Kindt et Billaut, 2006] where we also find an extension of the scheduling notation proposed by Graham et al. to the multi-criteria

case.

3.2.1 Multi-criteria scheduling problem notations

According to the three-field notation presented in [T'Kindt et Billaut, 2006] which is an extension of the one proposed by Graham et al. for single-criterion problems, we have :

- $\alpha|\beta|f^1,\ldots,f^K$: in the case where several objective functions are to be minimized and K is the number of functions.
- $\alpha|\beta|\varepsilon(f^1,\ldots,f^K)$: when the ε-constraint approach is considered; we try to minimize criterion f^1, subject to a maximum limit on each of the others objective functions must be respected. We then associate a vector $Q=(Q_2,\ldots,Q_K)$, such that each element Q_k is the upper bound of the criterion $f^k, k=2,\ldots,K$.
- $\alpha|\beta|F_l(f^1,\ldots,f^K)$: expresses the linear combination of all criteria.
- $\alpha|\beta|P(f^1,\ldots,f^K)$: this notation corresponds to the total enumeration of the Pareto front problem.
- $\alpha|\beta|\#(f^1,\ldots,f^K)$: expresses the study of the number of Pareto solutions of the problem $\alpha|\beta|f^1,\ldots,f^K$.

3.2.2 Example

Let us consider the bi-criteria scheduling problem $P_2|d_j=d|\sum U_j, C_{max}$ with common due date. This scheduling problem is $\mathcal{NP}$-hard. Suppose that we have to schedule 5 jobs on the 2 identical parallel machines where the data are:

Job(j)	1	2	3	4	5
Processing time (p_j)	1	2	3	4	5
Due date (d_j)	6	6	6	6	6

We know that LPT&FAM-heuristic is efficient for solving scheduling problem $P_2||C_{max}$. Hence, to solve this scheduling problem we can propose the following heuristic:

Step1: Apply LPT&FAM-heuristic to obtain solution σ

Step2: From σ build new solution where jobs assigned to machine M_1 (resp. M_2) are scheduled according to SPT rule.

According to Step1, we obtain $C_{max}=8$ and $\sum U_j=3$:

Machine	M_1			M_2	
Job(j)	5	2	1	4	3
Completion time (C_j)	5	7	8	4	7
Tardy job (U_j)	0	1	1	0	1

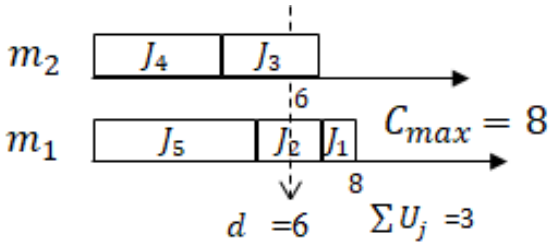

Figure 3.5: Minimizing both makespan and total tardy jobs on two-identical parallel machines

Finally, according to Step2, we build another solution. On both machines M_1 and M_2, jobs are sequenced with respect to SPT order. We obtain solution presented in Figure 3.6 with $C_{max} = 8$ and $\sum U_j = 2$, which is the first optimal Pareto solution

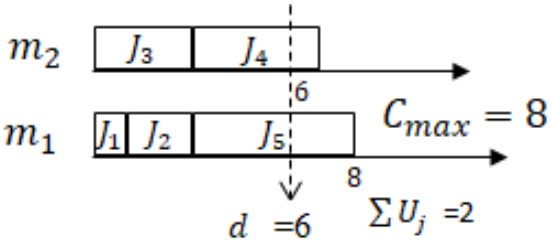

Figure 3.6: Makespan and total tardy jobs minimization: $C_{max} = 8$ and $\sum U_j = 2$

Now, if we start with Step2, we obtain a better solution for $\sum U_j = 1$ but the value of the makespan is worsening $C_{max} = 9$ which is the second optimal Pareto solution (see Figure **??**).

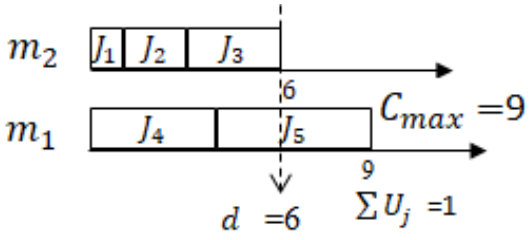

Figure 3.7: Makespan and total tardy jobs minimization: $C_{max} = 9$ and $\sum U_j = 1$

3.3 Multi-agent scheduling problems

Traditional scheduling models consider that all jobs are equivalent where the quality of overall scheduling is assessed according to a measure defined by a single objective function, applied to all jobs with almost no distinction [Baptiste et Brucker, 2004]. The

distinction between jobs is made by generally associating a weighting factor to each job [Dauzère-Pérès et Sevaux, 2003, Dauzère-Pérès et Pavageau, 2003]. This weight makes it possible to give more or less importance to one job than another. This distinction may not be sufficient. However, in this case, the same measure is applied to all work to quantify the quality of a schedule. Let us take for example, the case of the minimization of the total weighted completion times of jobs $\sum_{j\in\mathcal{N}} w_j C_j$ or the minimization of the weighted number of late jobs $\sum_{j\in\mathcal{N}} w_j U_j$ [Dauzère-Pérès et Sevaux, 2003]. In these cases, the weight or importance of the job is represented by the value w_j and the same measure applies to all jobs.

However, in a real-world context, these scheduling models are not always relevant. In some concrete situations, it may be necessary to examine several aspects of scheduling such as average completion times, with another measure relating to the respect of due dates. Give a good compromise between the criteria, it is the challenge of any scheduler. These are the multi-criteria scheduling problems [T'Kindt et Billaut, 2006, Jozefowska, 2007]. Again, in these studies, each performance measure is applied to all jobs, which can be perfectly justified.

In some practical cases, application of the same measure for all jobs may not be relevant, even in the case of classical multi-criteria scheduling. Indeed, it is possible to envisage a workshop where the job has the following particularity: jobs of a first subset may have due dates with a possible delay (to be reduced to a minimum), those of a second subset may have deadlines (to be respected obligatorily) and other jobs constituting a third subset do not have due dates but the objective is to minimize the work in progress. For the first subset, the decision is to reduce tardiness as much as possible; for the second subset, it cannot tolerate any delay; for the last type of jobs, we wants to reduce the total completion times. The scheduling of each subset is evaluated according to different objectives, but the jobs are all in competition for the use of resources (machines). This is a multi-criteria scheduling problem where a new type of compromise must be achieved. These scheduling problems are referred to in the literature as "multi-agent scheduling" [Agnetis *et al.*, 2007, Ng *et al.*, 2006a], [Kovalyov *et al.*, 2012] and [Kovalyov *et al.*, 2015] or "scheduling with competing agents" [Agnetis *et al.*, 2004, Agnetis *et al.*, 2009b] or "interfering job scheduling problems" [Hoogeveen, 2005]. Hence, in a multi-agent scheduling problem, there are several agents, each one is assigned a subset of jobs. Each agent has its own performance measure that depends solely on the scheduling of his jobs. It is therefore a local objective. However, agents should share the same resources to complete their respective jobs. We are therefore looking for a good compromise solution. These problems are close to the combinatorial optimization area and cooperative game theory [Agnetis *et al.*, 2000, Agnetis *et al.*, 2009a].

3.3.1 Definitions and notations

A scheduling problem involving several actors, where each has its own decision-making autonomy, in charge of executing its subset of jobs on the same resources (the jobs are competing for the use of the same machines), can be assimilated to a multi-agent scheduling problem, where a new type of compromise must be achieved.

We define the term "agent" as an entity associated with a subset of jobs. This entity

may be associated with another decision maker who intervenes in the choice of the final solution. Each agent aims to minimize a criterion of his own because it depends only on his own jobs. These agents compete since they share the same resources.

We note by $\mathcal{J}$ the set of n jobs to be scheduled on m parallel machines ($m \geq 1$). Without loss of generality, index j is used to refer to jobs, i for machines, k for agent, and we note J_j^k the j^{th} job of agent k and M_i the i^{th} machine. The following data and characteristics are associated with each job J_j:

- n: number of jobs to be scheduled;
- n_k: number of jobs of agent k;
- p_j^k: processing time of J_j^k on machine M_i (machines are identical);
- C_j^k: completion time of J_j^k, and we note $C_j^k(\sigma)$ the completion time of J_j^k according to sequence σ;
- d_j^k: due date of J_j^k; it is a date on which the job should be finished, otherwise a penalty cost is applied;
- $\tilde{d}_j^k$: deadline of J_j^k, i.e. no tolerance for delays;
- w_j^k: weight of J_j^k.
- C_{max}^k: makespan of agent k
- U_j^k: if $C_j^k < d_j^k$ then job J_j^k is late and $U_j^k = 1$; $U_j^k = 0$ otherwise.

3.3.1.1 Competing scenario

This is scheduling problem when the agents are independent, i.e. they have no common job. It means that all jobs of the same subset $\mathcal{J}^k$ belong exclusively to the sole agent k ($\mathcal{J}^A \cap \mathcal{J}^B = \emptyset$). In this case, we are talking about a COMPETITION scheduling problem. This is a class of problems introduced by Agnetis et al [Agnetis *et al.*, 2000, Agnetis *et al.*, 2004]. In this case the agents' notation is symmetrical, i.e. the problems with objective functions f^A and g^B are the same as the problems with objective functions g^A and f^B. According to the three fields notations of scheduling problems, authors propose to introduce term $''CO''$ in β-field: $\alpha|CO, \beta|f^A, g^B$ (see Figure 3.8).

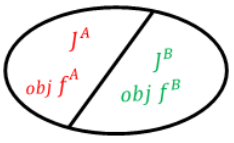

Figure 3.8: Competing scenario

3.3.1.2 Interfering scenario

In this case the subsets of the jobs are numbered as follows: $\mathcal{J} = \mathcal{J}_1 \supseteq \mathcal{J}_2 \supseteq \ldots \supseteq \mathcal{J}_K$. Note that agent 1 is a global agent (in charge of the whole jobs), i.e. $\mathcal{J} = \mathcal{J}_1$. According to the three fields notations of scheduling problems, authors propose to introduce term $''IN''$ in β-field: $\alpha|IN, \beta|f^A, g^B$. This class of problems is asymmetric. For example, problems $\alpha|IN, \beta|f^A, g^B$ and $\alpha|IN, \beta|g^A, f^B$ may have a different complexity (see Figure 3.9).

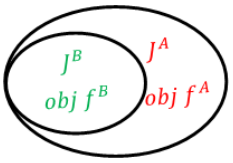

Figure 3.9: Interfering scenario

3.3.1.3 Non-disjoint scenario

This is the most general case, in which the two subsets of work can share work, i.e. $\exists k$ and $k' : \mathcal{J}^k \cap \mathcal{J}^{k'} \neq \emptyset$. According to the three fields notations of scheduling problems, authors propose to introduce term $''ND''$ in β-field: $\alpha|ND, \beta|f^A, g^B$ (see Figure 3.10).

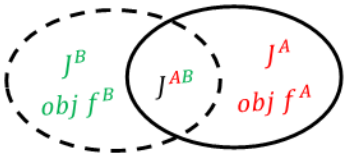

Figure 3.10: Non-disjont scenario

3.3.2 Examples

By the following example, we analyze the three scenarios $\beta_x \in \{CO, IN, ND\}$ and starting by considering classical bi-criteria scheduling problem. The approach considered in this example is the ε-constraint approach. Let be 6 jobs to be scheduled without preemption on a single machine, where the objective is to minimize the two criteria: $\sum C_j^A$ and $C_{\max}^B$.

The problem data are:

j	1	2	3	4	5	6
p_j	1	2	3	4	5	6

Classical bi-criteria problem (BI)
The problem is denoted as $1|BI, C_{\max}^B \leq Q_B| \sum C_j^A$. The problem is therefore to determine a sequence minimizing the total completion times ($\sum C_j^A$) of agent A by respecting the constraint $C_{\max}^B \leq Q_B$ (the completion time of the last job is less than or equal to Q_B).

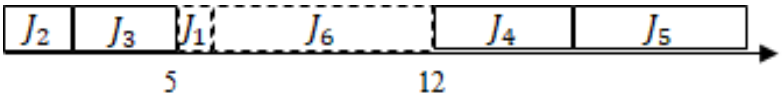

Figure 3.11: Strict Pareto solution: $(\sum C_j^A = 44, C_{\max}^B = 12)$

We notice that there is only one solution that optimizes the $\sum C_j$ criterion since the makespan is a constant given by the sum of the processing times. It is therefore the sequence $\sigma = (1, 2, 3, 3, 4, 5, 6)$ where the jobs are scheduled in SPT order, with $\sum C_j^A = 56$ and $C_{\max}^B = 21$.

Competition problem (CO)
Here, it is a scheduling problem with two competing agents where the two subsets of the jobs are independent, i.e. $\mathcal{J}_A \cap \mathcal{J}_B = \emptyset$. The problem $1|CO, C_{\max}^B \leq Q_B| \sum C_j^A$ consists in looking for a solution that minimizes $\sum C_j^A$ of agent A's jobs while respecting the upper bound Q_B imposed on the makespan of agent B's jobs. In this case, the value of each agent's objective function is based solely on the completion times of each respective agent's jobs.

With the same data of previous example, we assume that agent A should perform the following 4 jobs: $(J_1^A = J_2; J_2^A = J_3; J_3^A = J_4; J_4^A = J_5)$. The jobs of agent B are $(J_1^B = J_1; J_2^B = J_6)$.

As it is a question of respecting the upper bound on agent B's makepan, their completion times are given by the last scheduled job. In this case, it is trivial to consider a single job J_0^B of agent B to be scheduled with processing time $p_0^B = p_1^B + p_6^B = 7$, since the value of agent A's objective $(\sum C_j^A)$ is independent of the completion times of agent B's jobs, i. e. agent B's jobs are to be performed contiguously.

By listing all solutions, we have only five optimal strict Pareto. Any other solution is therefore dominated. We have:

1. $(J_2^A, J_3^A, J_4^A, J_5^A, J_0^B)$: criteria values are $(\sum C_j^A, C_{\max}^B) = (30, 21)$
2. $(J_2^A, J_3^A, J_4^A, J_0^B, J_5^A)$: criteria values are $(\sum C_j^A, C_{\max}^B) = (37, 16)$
3. $(J_2^A, J_3^A, J_0^B, J_4^A, J_5^A)$: criteria values are $(\sum C_j^A, C_{\max}^B) = (44, 12)$
4. $(J_2^A, J_0^B, J_3^A, J_4^A, J_5^A)$: criteria values are $(\sum C_j^A, C_{\max}^B) = (51, 9)$
5. $(J_0^B, J_2^A, J_3^A, J_4^A, J_5^A)$: criteria values are $(\sum C_j^A, C_{\max}^B) = (58, 7)$

Thus, for example, for $Q_A = 46$, the only strict Pareto solution corresponds to the sequence $(J_2^A, J_3^A, J_0^B, J_4^A, J_5^A)$ for a value of $C_{\max}^B = 12$ with $\sum C_j^A = 44$ (see Figure 3.11).

Interfering problem (IN)
The problem denoted $1|IN, C_{\max}^B \leq Q_B| \sum C_j^A$ corresponds to a multi-agent scheduling problem with two agents: agent A aims to minimize the total completion times of all jobs, however, agent B seeks to minimize the completion time of his jobs.

With the same data of previous example, we assume that the jobs of agent B are $(J_1^B = J_1; J_2^B = J_6)$, as in the case of CO.

The first observation we can make in this case is that the execution of agent B's jobs in a contiguous manner is not dominant. Thus, the number of combinations increases compared to the previous case, i.e. we can have more strict Pareto solutions in this case than in the CO case. Thus, for a value $Q_2 = 12$ for example, the optimal sequence is $(J_1, J_2, J_3, J_6, J_4, J_5)$ where the values of criteria are: $\sum C_j^1 = 59$ and $C_{\max}^2 = 12$

Non-disjoint problem (ND)
The problem denoted $1|ND, C_{\max}^B \leq Q_B| \sum C_j^A$ corresponds to a multi-agent scheduling problem with two agents: agent A aims to minimize the total completion times of all jobs, however, agent B seeks to minimize the completion time of his jobs. In this case, the value of each agent's objective function is based on the completion times of each respective agent's jobs including the common job.

With the same data of previous example, we assume that agent A should perform 3 jobs belonging to him: $(J_1^A = J_2; J_2^A = J_3; J_3^A = J_4)$ and the fourth job $J_1^{(A,B)} = J_5$ that shares it with agent B. Hence, the jobs of agent B are $(J_1^B = J_1; J_2^B = J_6)$ and the third job $J_1^{(A,B)}$.

As case CO, it is trivial to consider a single job J_0^B of agent B to be scheduled with processing time $p_0^B = p_1^B + p_6^B = 7$. However, as there is a sharing job, its completion time contributes to both criteria. Dealing with criterion of agent A this job should be executed earliest, i.e. before job J_0^B and therefore agent's A jobs (including $J_1^{(A,B)}$) are scheduled in SPT order.

By listing all solutions, we have only four optimal strict Pareto. Any other solution is therefore dominated. We have:

1. $(J_2^A, J_3^A, J_4^A, J_1^{(A,B)}, J_0^B)$: criteria values are $(\sum C_j^A, C_{\max}^B) = (30, 21)$
2. $(J_2^A, J_3^A, J_1^{(A,B)}, J_0^B, J_4^A)$: criteria values are $(\sum C_j^A, C_{\max}^B) = (38, 17)$
3. $(J_2^A, J_1^{(A,B)}, J_0^B, J_3^A, J_4^A,)$: criteria values are $(\sum C_j^A, C_{\max}^B) = (47, 14)$
4. $(J_1^{(A,B)}, J_0^B, J_2^A, J_3^A, J_4^A)$: criteria values are $(\sum C_j^A, C_{\max}^B) = (57, 12)$

Thus, for example, for $Q_A = 50$, the only strict Pareto solution corresponds to the sequence $(J_2^A, J_1^{(A,B)}, J_0^B, J_3^A, J_4^A,)$ for a value of $C_{\max}^B = 14$ with $\sum C_j^A = 47$

3.3.3 Complexity study

Let Π_1 and Π_2 be two problems, we note $\Pi_1 \propto \Pi_2$ if Π_1 is at least as difficult as Π_2. The following proposal was made [Agnetis *et al.*, 2014].

Proposition 1 *The following proposals are true:*

1. $\alpha|\beta|f^k \propto \alpha|\beta|f^1, \ldots, f^k, \ldots, f^K$ *(Mono-criterion scheduling problem* $\propto$ *Multi-criteria scheduling problem)*
2. $\alpha|\beta|f^k \propto \alpha|CO, \beta|f^1, \ldots, f^k, \ldots, f^K$ *(Mono-criterion scheduling problem* $\propto$ *Computing scheduling problem)*
3. $\alpha|\beta|f^k \propto \alpha|IN, \beta|f^1, \ldots, f^k, \ldots, f^K$ *(Mono-criterion scheduling problem* $\propto$ *Interfering scheduling problem)*
4. $\alpha|CO, \beta|f^1, \ldots, f^K \propto \alpha|ND, \beta|f^1, \ldots, f^K$ *(Competing scheduling problem* $\propto$ *Non-disjont scheduling problem)*
5. $\alpha|IN, \beta|f^1, \ldots, f^K \propto \alpha|ND, \beta|f^1, \ldots, f^K$ *(Interfering scheduling problem* $\propto$ *Non-disjont scheduling problem)*

According to Proposition 1, we deduce that multi-criteria scheduling problems are a special case of the interfering scheduling problem. Indeed, according to the definition of interfering scenario the subset of jobs are all included or equal to the total set of jobs $\mathcal{J}$. On the other hand, obviously the non-disjoint scenario is the general case, and hence any scheduling problem is reduced to this general case (see Figure 3.12).

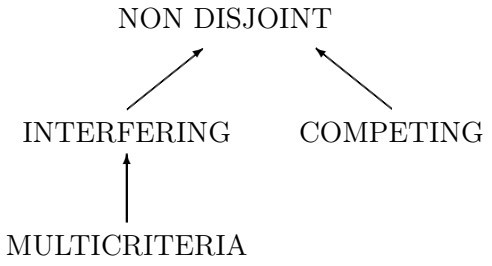

Figure 3.12: Reduction graph between scenarios

We know that if mono-criterion scheduling problem $\alpha|\beta|f$ is $\mathcal{NP}$-hard then all multi-criteria scheduling problems including at least objective function f are $\mathcal{NP}$-hard. Figure 3.12 shows that: if the multi-criteria scheduling problem is $\mathcal{NP}$-hard, then both interfering and non-disjoint multi-agent scheduling with the same objective functions are also $\mathcal{NP}$-hard. If an algorithm polynomial can solve non-disjoint jobs, then this algorithm can be applied to both multi-criteria and interfering jobs problems, and also to the competing case.

Let's now consider linear combination of criteria approach. Determine an optimal solution for non-disjoint scheduling problem we have to minimize: $F = \sum_{k=1}^{K} \lambda_k f^k$.

Proposition 2 *If objective functions* f^k *are the same and are of type* $min - sum$*, i.e.* $f^k = \sum_{J_j \in \mathcal{J}^k} f(C_j)$*, then non-disjoint scheduling problem* $\alpha|ND, \beta|F_l(f^1, \ldots, f^K)$ *and mono-criterion scheduling problem* $\alpha|\beta|\sum_{J_j \in \mathcal{J}} w_j f_j$ *are equivalent.*

Proof: Suppose that the objective functions f^k all the same and of type $min - sum$. Hence, for each job J_j we can define weight w_j as follow: $w_j = \sum_{k=1}^{K} \lambda_k x_{jk}$, where

$x_{jk} = 1$ if $J_j \in \mathcal{J}^k$; 0 otherwise. In this case, the non-disjoint scheduling problem $\alpha|ND, \beta|F_l(f^1, \ldots, f^K)$ is equivalent to $\alpha|\beta|\sum_{J_j \in \mathcal{J}} w_j f_j$ (the mono-criterion scheduling problem).

For example, let's consider the multi-agent scheduling problem $1|ND|F_l(\sum w_j^1 C_j^1, \ldots, \sum w_j^K C_j^K)$. According to Proposition 2, this problem is equivalent to $1||\sum w_j C_j$ where an optimal solution can be obtained in $O(nlogn)$ running time, i.e. jobs are scheduled according to $WSPT$ rule. However, scheduling problem $1|ND|\sum w_j^1 C_j^1, \ldots, \sum w_j^K C_j^K$ admits an exponential number of strict Pareto solutions. In fact, in competing case Hoogeveen show that the scheduling problem $1|CO|\sum w_j^1 C_j^1, \ldots, \sum w_j^K C_j^K$ accepts an exponential optimal solutions, even if $K = 2$ [Hoogeveen, 1992].

3.3.4 Studied problems and motivations

3.3.4.1 Studied problem

In this thesis, the studied problem denoted $Pm|ND, d^A|C^B_{\max}, \sum U_j^A$ corresponds to a multi-agent scheduling problem with two agents: agent A aims to minimize the number of late jobs, however, agent B seeks to minimize the completion time of his jobs. In this case, the value of each agent's objective function is based on the completion times of each respective agent's jobs including the common jobs. Agent A's jobs should be achieved before a common due date d^A. However, agents should share the same identical parallel machines to complete their respective jobs. We are therefore looking for a good compromise solution. To determine a Pareto optimal solution or the optimal Pareto front, ε-approach, linear combination of criteria and enumerating Pareto front are considered. in this study the symmetric case is also studied, i.e. ε-constraint problems $Pm|ND, d_j^A, C^B_{\max} \leq Q_B|\sum U_j^A$ and $Pm|ND, d_j^B, \sum U_j^B \leq Q_B|C^A_{\max}$.

3.3.4.2 Complexity of studied problem

In general, if mono-criterion scheduling problem is $\mathcal{NP}$-hard then the corresponding multi-agent scheduling problem on term of the objective function is also $\mathcal{NP}$-hard [Agnetis *et al.*, 2014]. Hence, the studied scheduling problem $Pm|d^B|\sum U_j^B, C^A_{max}$ is also $\mathcal{NP}$-hard. Note that in the case of single machine scheduling, Competing ε-constraint problem $1|CO, d_j^A, C^B_{\max} \leq Q_B|\sum U_j^A$ is polynomial and can solved in $O(n_A logn_A + n_B + logn_B)$ time by suitable generalization of Moor's algorithm [Agnetis *et al.*, 2014]. This result is always valid for any regular $min - max$ type criterion of agent B, i.e. $1|CO, d_j^A, f^B_{max} \leq Q_B|\sum U_j^A$ is polynomial. Unfortunately, this result cannot be applied to the other scenarios IN and ND. Thus, the complexity of the ε-constraint problems $1|IN, d_j^A, C^B_{\max} \leq Q_B|\sum U_j^A$ and $1|ND, d_j^A, C^B_{\max} \leq Q_B|\sum U_j^A$ is still open. In our study, we focus on the common due date, denotes d^k, $k \in A, B$.

To compute the Pareto set, obviously agent A holds $\sum U_j^A$. So, there are $O(n_A)$ Pareto optimal solutions. Hence, in the competing scenario $1|CO, d^A|P(\sum U_j^A, C^B_{\max})$ can be solved in polynomial time.

Example

Let's consider the two-machine multi-agent scheduling problem $P2|ND, d^B = 6, \sum U_j^B \leq 0|C_{\max}^A$ where $\varepsilon-$constraint approach is used to determine an optimal schedule. Note that agent B wants to execute all his tasks in time (no tardy jobs for agent B). The set jobs data problem are:

Job(j)	1	2	3	4	5
Processing time (p)	1	2	3	4	5
Agent	A,B	B	A	A	B

The common due date is: $d^B = 6$. If we reuse the optimal solution for agent A, we obtain $C_{max}^A = 4$. But for agent B we have: $\sum U_j^B = 1$, see Figure 3.13.(a) (that is not satisfy the request).

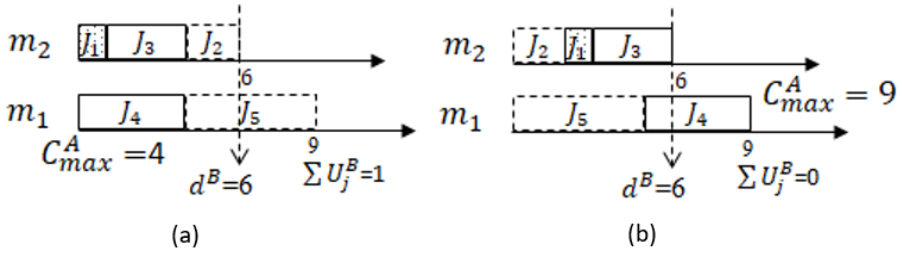

Figure 3.13: Minimizing makespan and number of tardy jobs on two machines

Let's now consider a second solution as presented in Figure 3.13.(b). Then we have $C_{max}^A = max_{(j=1,3,4)} C_j^A = 9$ and $\sum U_j^B = 0$ (that is satisfy the request).

3.3.4.3 Motivations

The problem addressed in this thesis may occur in a computer system, such as networks or IT project management.

Network systems

Scheduling with competing agents can occur in networks, more precisely when switching packets in an asynchronous ATM time transfer mode or in a computing grid (see Figure 3.14.

For ATM networks, known as service integration networks, they are networks that carry various types of traffic such as voice, video, image by transfer, and various types of computer data. In such systems, the information carried by the network is first divided into small packets. These packets are put in a "buffer" tempo zone waiting for transmission and a scheduling algorithm determines the order of their transmission. It is relevant to classify these types of traffic according to whether they have constraints on their completion time (e. g. deadline), or whether they should be processed as soon as possible, i. e. without constraints on the completion time. Therefore, the diversity of traffic implies various objective functions. For example, for most types of computer data, performance is generally measured by considering the average queue flow, which is equivalent to minimizing the total weighted completion times $\sum_j w_j C_j$. However, packets corresponding to

voice and video, if they remain in a queue for a long time and do not reach their destination in time for playback, will be lost. The appropriate objective for these packages is surely to minimize the average number of weighted late jobs, which we note $\sum_j w_j U_j$ [Peha et Tobagi, 1990]. Therefore, some treatments can be mutualized and then these tasks form common processes. This is typically the case of threads execution, which they share information about the state of the process, memory zones, and other resources. This is a non-disjoint multi-agent scheduling problem.

Figure 3.14: No-disjoint multi-agent scheduling problem vs Networks

IT project management

A well-known French digital services company (ESN), an expert in the field of new technologies and IT, supports its client companies in the implementation of IT projects (see Figure 3.15). These projects can compete for the use of renewable resources shared over time (people with diverse and varied skills) [Dhib *et al.*, 2011]. Each project manager is responsible for the execution of his project and must therefore negotiate the use of resources with the other project managers. In this case, all the activities of an agent are defined by the tasks of his project. All projects are disjointed. However, some of the activities carried out may benefit other projects (web development, database, administration, test platform, etc.). Each project manager therefore tries to build his team by defining a share allocated to each one, i.e. to define the participation rate per week of a person in the project in question. A person can be involved in various projects. The objective of each project manager is to complete the project as quickly as possible, while maximizing the satisfaction of the people assigned to the tasks.

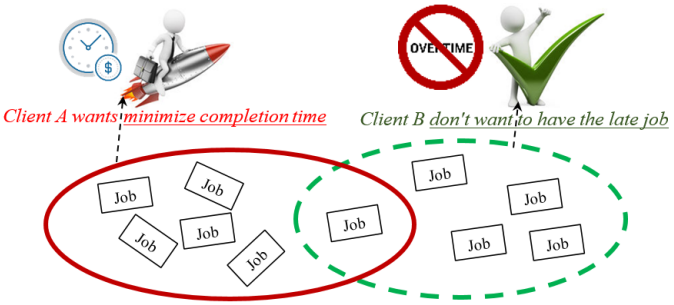

Figure 3.15: No-disjoint multi-agent scheduling problem vs IT projects

3.4 Main results related to the studied problems

The scheduling problem on a single machine with competing agents "CO" has been widely studied in the literature [Baker et Smith, 2003, Agnetis *et al.*, 2004, Yuan *et al.*, 2005, Ng *et al.*, 2006a, Cheng *et al.*, 2008] unlike the Interferering case "IN" which is particularly studied in [Huynh Tuong et Soukhal, 2009] and [Huynh Tuong *et al.*, 2012]. In this section, it is not a question of establishing a complete state of the art but of highlighting the most relevant results in terms of links with our approach and objectives, namely the exact resolution of problems with the ε constraint approaches and the linear combination of criteria. The complexity of enumerating the Pareto front is deduced, where possible. In [Agnetis *et al.*, 2014], a large complete state of the art is established where general table summarizing the results elaborated on the complexity of multi-agent scheduling problems. This book is hence dedicated to the multi-agent scheduling problems. The authors give an introduction to multi-agent scheduling, introducing general definitions and notation, several resolutions approach for multicriteria problems and different scenario when considering several agents. A lot of the algorithms have been presented dealing with different criteria, in particular regular criteria.

In 2004 Agnetis et al. [Agnetis *et al.*, 2004] were the first, to our knowledge, to introduce definitions and concepts of multi-agent scheduling problems. The authors are interested in the COMPETITION case when the ε-constraint approach is considered for regular criteria. New complexity results and dynamic programming algorithms have been developed. Agnetis et al. are also interested in listing all non-dominated solutions (Pareto front) [Agnetis *et al.*, 2007].

Note that in 2003, Baker and Smith [Baker et Smith, 2003] studied a scheduling problem on a single machine in charge of performing jobs for two or three clients. Unlike the classic case where a single measure is applied to all client work, the authors consider an aggregation function on all client criteria. This is therefore the case of COMPETITION with two and three agents where the criteria to be minimized are: the makepan C_{max}, the maximum delay L_{max} and the total weighted execution time of the work $\sum w_j C_j$. Baker and Smith show that the combination of these criteria makes the problem, in some cases, $\mathcal{NP}$-hard. Additional results to the work of Baker and Smith were provided by Yuan et al. [Yuan *et al.*, 2005]. Other works were introduced much earlier as presented in section 3.3.4.3 [Peha, 1995].

The COMPETITION problem with K agents was addressed in [Ng *et al.*, 2006a], where the objective of each subset is to minimize the weighted total number of late jobs. The *goal programming* approach is considered. In the case where K is not fixed, the problem is shown $\mathcal{NP}$-hard in the strong sense. However, it can be resolved in pseudo-polynomial time when the number of agents (K) is fixed and a fully polynomial approximation scheme (FPTAS) is proposed. In the case where the weights of the work are unitary, the problem is shown polynomial. In [Ng *et al.*, 2006b], authors show that problems with two agents for minimizing the total weighted completion times and minimizing the maximum lateness are NP-difficult in the strong sense, and propose a pseudo-polynomial algorithm for the problem with unit weights.

In 2008, Cheng et al. also looked at the COMPETITION case with K agents, where

each agent's objective is of the type *min-max* [Cheng *et al.*, 2008]. The goal programming approach is considered. The authors show that the feasibility problem can be solved in polynomial time, even in the presence of precedence constraints. Some polynomial cases are identified.

Agnetis et al. [Agnetis *et al.*, 2009a] propose a branch& bound method to solve three scheduling problems with two competing agents on a single machine. Lagrangian relaxation is used to obtain a lower bound. The criteria chosen lead to an effective resolution in polynomial time of the Lagrange dual. The problems considered minimize the total completion times of the first agent, where upper bounds on the makespan, maximum lateness or total weighted completion times of the second agent should be respected.

In [Lee *et al.*, 2009], authors treat problems with K agents. In their model, each agent minimizes the weighted total completion times of its own jobs. An approximation algorithm is given. In [Leung *et al.*, 2010], authors propose a complexity analysis for two agents competing scheduling problem with single machine. The agents' criteria are min-max, total weighted completion times, total tardiness and total number of tardy jobs. Some jobs have specific execution constraints: Agent A's jobs is carried out without pre-emption and are not available at time zero, where Agent B's jobs are all available at time zero and they can pre-empted. The authors show the $\mathcal{NP}$-hardness of their studied problems by a reduction from the PARTITION problem.

In [Wu *et al.*, 2013] examine the problem with two disjoint agents each aiming to minimize the total weighted completion times of his jobs with release dates. The authors show that the problem studied is $\mathcal{NP}$-hard in the strong sense. Several dominance conditions are proposed and implemented in a branch&bound procedure. Four genetic algorithms and ant colonies optimization heuristic are proposed. Their experiments show that the exact procedure solves instances of 16 jobs, and that the use of the results of the approached methods considerably improves its performance in terms of computation time and the number of nodes explored. [Lee *et al.*, 2013] consider two competing disjointed agents on a single machine. One of the agents aims to minimize the linear combination of the total completion times and the maximum lateness, while the second agent does not admits any delay for his jobs. A procedure by separation and evaluation resolving instances of 24 jobs has been developed. In [Ketan et Balasubramanian, 2014], the authors study two $\mathcal{NP}$-hard problems. The considered criteria of the first problem are the minimization of the total completion times as well as the minimization of the number of late jobs. An effective heuristic combining SPT and EDD rules is developed. The second problem, agent A are looking for solution that minimizes the weighted total completion times and the second agent wants to minimize the maximum lateness. Another efficient heuristic both based on WSPT and EDD rules is developed.

Three disjoint agents are considered in [Lee et Wang, 2014]. The first wants to minimize the total weighted completion times while respecting two constraints: the second agent's makepan does not exceed a given value and the third agent must ensure a maintenance task between two very specific dates. For this particular problem, the authors develop an exact method based on branch&bound procedure. The lower bound used is based on the results of Posner [Posner, 1985]. The main idea of the proposed resolution method is to solve the problem with pre-emption when the maintenance task is scheduled as late as possible.

Concerning interfering scenario, i.e. results related to multi-agent problems with a global objective function can be found in [Huynh Tuong *et al.*, 2011]. In fact, the authors introduce new complexity results of several single-machine problems in which the agents compete to perform their objectives, knowing that they all have an impact on the global objective function. The authors consider both linear combination and $\varepsilon-$ constraint approaches. Dynamic programming and specific polynomial algorithms are also proposed. Interfering two-agent parallel machines problems are also tackled in [Sadi *et al.*, 2014]. Pseudo-polynomial dynamic programming algorithms are derived for various problems with the different combination of the objective functions.

We will not recall here the results of the literature dealing with the classical multi-criteria scheduling problems. T'Kindt and Billaut's book [T'Kindt et Billaut, 2006] presents a detailed study of the main results.

Very few results of the literature on the COMPETITION case are dedicated to the case of m parallel machines. Remember that the case $1|CO|\sum C_j^A, \sum C_j^B$ is $\mathcal{NP}$-hard. The COMPETITION problems $Pm|CO|f^A, f^B$ are therefore $\mathcal{NP}$-hard, whatever the classical regular objective function considered.

Peha [Peha, 1995] considers the case of scheduling jobs in networks on identical processors, $P|r_j, p_j = 1, d_j|lex(\sum w_j U_j^A, \sum w_j C_j^B)$. The authors propose pseudo-polynomial algorithms according to the due dates. However, these due dates in a network context are generally limited by the number of jobs.

Balasubramanian et al. [Balasubramanian *et al.*, 2009a] are interested in COMPETITION scheduling on identical parallel machines in the presence of two agents, noted $Pm|CO|\varepsilon\big(\sum C_j^A/C_{max}^B\big)$. The problem is $\mathcal{NP}$-hard in the ordinary sense. The authors develop efficient heuristics to enumerate the Pareto front. As it is the C_{max}^B criterion, on each machine the jobs of agent B are grouped into a block, thus separating agent A's jobs into two blocks. The jobs of agent A (or agent B) is scheduled according to the SPT (or LPT) order. An evolutionary algorithm has been developed. The authors also propose a MIP to generate all the strictly non-dominated solutions in an iterative way. In their study, the *varepsilon*-constraint approach is considered. Similarly, an optimal solution for the $Pm||\varepsilon(\sum w_j C_j^A)/C_{max}^B)$ can be obtained in pseudo-polynomial running time where a dynamic program is proposed to calculate a Pareto solution.

In [Elvikis *et al.*, 2011], the authors consider the problem $Qm|CO, p_j = p|P\big(f_{max}^A, f_{max}^B\big)$. For the enumeration of all solutions of the strict Pareto, the authors propose a polynomial algorithm in $O(n_A^2 + n_B^2 + n_B^2 + n_A n_B log(n_B))$ (n_A and n_B correspond to the number of jobs of each agent). The authors also study the classic cases $f_{max}^k \in \{L_{max}^k, C_{max}^k\}$.

In the context of scheduling problems in grids, in [Cordeiro *et al.*, 2011], the authors are interested in the case of K multi-agent scheduling problems where the objective is the distribution of loads under an overall objective function that is the makepan, it is about interfering scenario. The authors propose a 2-approximation algorithm to calculate collaborative solutions.

Dealing with multi-agent scheduling problems, few polynomial cases have been identified. The problem $P2|CO, pmtn|\varepsilon(\sum C_j^A/f_{\max}^B)$ is shown polynomial. However, the case of 3 machines remains open [Wan *et al.*, 2010]. In [Sadi *et al.*, 2014] authors show that when preemption is considered and objective function of each agent is the same and is

of type min-max, the parallel machines interfering multi-agent scheduling problems are polynomial. In [Sadi et Soukhal, 2017], authors give complexity analyses for multi-agent scheduling problems with a global agent and equal length jobs. The authors had the new results in scheduling problem where disjoint agents are competing to schedule their jobs on the same identical parallel machines and aim at minimizing their own objective functions. A global objective function on the set of jobs has to be minimized. All the jobs have equal length requirements. Their new complexity results and polynomial time algorithms are developed.

For non-disjoint multi-agent scheduling problems and based on the reduction graph presented in section 3.3.3, we can deduce the following complexity results presented in Table 3.1 and Table 3.2 (see [Agnetis *et al.*, 2014]).

Problem	Complexity
$1\|ND\|\alpha C^B_{max} + \beta C^A_{max}$	$O(n)$
$1\|ND\|\alpha f^B_{max} + \beta f^A_{max}$	$O(n^4)$
$1\|ND\|\alpha \sum C^B_j + \beta C^A_{max}$	$O(n_B \log n_B)$
$1\|ND\|\alpha \sum C^B_j + \beta f^A_{max}$	$O(n^4)$
$1\|ND\|\alpha \sum w^B_j C^B_j + \beta C^A_{max}$	$O(n \log n)$
$1\|ND\|\alpha \sum w^B_j C^B_j + \beta L^A_{max}$	sNPH
$1\|ND\|\alpha \sum C^B_j + \beta \sum C^A_j$	$O(n \log n)$
$1\|ND\|\alpha \sum w^B_j C^B_j + \beta \sum w^A_j C^A_j$	$O(n \log n)$
$1\|ND\|\alpha \sum C^B_j + \beta \sum U^A_j$	bNPH
$1\|ND\|\alpha \sum w^B_j C^B_j + \beta \sum U^A_j$	sNPH
$1\|ND, d^B_j = d^A_j\|\alpha \sum U^B_j + \beta \sum U^A_j$	$O(n^3)$
$1\|ND\|\alpha \sum U^B_j + \beta \sum U^A_j$	bNPH

sNPH: strongly NP-Hard. bNPH: binary NP-Hard.

Table 3.1: Complexity results of non-disjoint two-agent scheduling problems: Linear combination approach

Problem	Complexity
$1\|ND, f^2_{max} \leq Q_2, ..., f^K_{max} \leq Q_K\|f^1_{max}$	$O(n^2)$
$1\|ND, f^2_{max} \leq Q_2, ..., f^K_{max} \leq Q_K\|\sum C^1_j$	$O(n \log n)$
$1\|ND, C^2_j \leq Q_2, ..., C^K_j \leq Q_K\|\sum C^1_j$	bNPH, $O(n^{2K-1}\overline{Q}^{K-1})$
$1\|ND, d^1_j = d^2_j = ... = d^K_j, \sum U^2_j \leq Q_2, ..., \sum U^K_j \leq Q_K\|\sum U^1_j$	$O(n^{k+1})$
$1\|ND, d^1_j = d^2_j = ... = d^K_j, \sum w^2_j U^2_j \leq Q_2, ..., \sum w^K_j U^K_j \leq Q_K\|\sum w^1_j U^1_j$	bNPH, $O(nW_1 Q_2, Q_2, ..., Q_K)$
$1\|ND\|\alpha_k C^k_{max}$	$O(n2^K)$
$1\|ND\|\sum_{k=1}^{K-1} \alpha_k (\sum w^k_j C^k_j) + \alpha_K C^K_{max}$	$O(n \log n)$
$1\|ND\|\sum_{k=1}^{K} \alpha_k (\sum w^k_j C^k_j)$	$O(n \log n)$
$1\|ND\|\sum_{k=1}^{K-1} \alpha_k (\sum w^k_j C^k_j) + \alpha_K \sum U^K_j$	$sNPH$
$1\|ND\|\sum_{k=1}^{K-1} \alpha_k (\sum U^k_j) + \alpha_K \sum w^K_j C^K_j$	$sNPH$
$1\|ND, d^k_j = d_j\|\sum_{k=1}^{K-1} \alpha_k \sum U^k_j$	$O(n^{K+1})$

Table 3.2: Complexity results of non-disjoint K-agent scheduling problems
Note: K fixed. We let U denote $\overline{Q} = max_{2 \leq k \leq K}\{Q_k\}$

3.5 Conclusion

This chapter is devoted to a general overview of scheduling problems, in particular to multi-agent scheduling problems. At first, The notion of multi-agent scheduling is defined to remove any ambiguity with multi-agent systems. Some definitions, examples and notations used throughout this manuscript are then introduced for ease of reading. To situate our research work in relation to the specialized literature, a synthesis of multi-agent scheduling problems is presented, drawing links and differences between the different scenarios identified. Our work focuses on a particular class of multi-objective scheduling problems "Non-disjoint scenario". Based on the reduction graph presented in section 3.3.3, some complexity results are deduced. We then introduce the studied scheduling problem and our motivations based on real-application, denoted $Pm|ND, d^B|\sum U_j^B, C_{max}^A$. As there are few results, at the and of this chapter, we present some related works. The next chapters are dedicated to the developed resolution methods (exact and heuristics).

Chapter 4

Exact methods and solvable cases

Résumé: Dans ce chapitre, nous analysons les propriétés de la solution optimale de Pareto des problèmes d'ordonnancement multi-agents étudiés. Plus précisément, nous nous intéressons à la classe de scénario non disjoint, noté ND. Une définition détaillée de cette classe de problèmes est rapplée dans la section 3.3. Nous rappelons que chaque agent est associé à son ensemble de travaux, mais certains travaux sont communs aux deux agents. Chaque agent minimise une fonction objective qui dépend de ses propres tâches. Cette classe de problèmes n'a pas connu le même développement théorique que les autres classes de problèmes d'ordonnancement multi-agents. Notre objectif dans ce chapitre est de proposer d'abord, une étude sur la structure de la solution optimale de Pareto. Ensuite, nous analysons quelques cas particuliers pour identifier ceux qui peuvent être résolus en temps polynomial. Ainsi, le cas d'une seule machine est étudié où l'on montre que les problèmes sont polynomiaux. Lorsque les travaux sont de même temps de traitement (durées opératoires identiques), des algorithmes polynomiaux sont proposés pour les machines paralléles. A la fin de ce chapitre, nous proposons deux MILPs pour calculer le front de Pareto exact. Des résultats expérimentaux montrant la performance de ces deux MILPs sont présentés et discutés.

Abstract: In this chapter we analyse the properties of Pareto optimal solution of studied multi-agent scheduling problems. More precisely, we are interested in the class of non-disjoint scenario, this class is denoted ND. A detailed definition of this class of problems is given in section 3.3. We recall that each agent is associated with his set of jobs, but some jobs are common. Each agent minimizes an objective function that depends on its own jobs. This problem class has not undergone the same theoretical development as the other multi-agent scheduling problem classes. Our objective in this chapter is to propose a study of the structure of Pareto optimal solution. Then, we analyze some particular cases to identify those that can be resolved in polynomial time. Thus, the case of a single machine is studied where we show that the problems are polynomial. When the jobs are equal length (identical processing times), polynomial algorithms are proposed for parallel machines. At the end of this chapter, we propose two MILPs to compute the exact Pareto front. Experimental results showing the performance of these two MILPs are conducted and presented.

4.1 Studied problems

In this chapter, the addressed scheduling problems are the following:

- Single-machine case
  1. $1|ND, d^B, \sum U_j^B \leq Q_B|C_{max}^A$
  2. $1|ND, d^A, C_{max}^B \leq Q_B|\sum U_j^A$
  3. $1|ND, d^B|P(C_{max}^A, \sum U_j^B)$
- m-paralle machines with equal length jobs
  1. $Pm|ND, d^B, p_j = p, \sum U_j^B \leq Q_B|C_{max}^A$
  2. $Pm|ND, d^A, p_j = p, C_{max}^B \leq Q_B|\sum U_j^A$
  3. $Pm|ND, d^B, p_j = p|P(C_{max}^A, \sum U_j^B)$
- m-paralle machines, general case
  1. $Pm|ND, d^B, \sum U_j^B \leq Q_B|C_{max}^A$
  2. $Pm|ND, d^A, C_{max}^B \leq Q_B|\sum U_j^A$
  3. $Pm|ND, d^A, |P(\sum U_j^A, C_{max}^B)$

In this chapter, the exact methods are applied to the problems resulting from the use of ε-constraint approach $Pm|ND|\varepsilon(f^A/f^B)$ with Q_B the maximum bound on the agent B's criterion. Recall that jobs of agent A (respectively B) are in $\mathcal{J}^A$ (respectively $\mathcal{J}^B$). We denote by $\mathcal{J}^{A,B}$ the set of common jobs, if necessary.

4.2 Preliminary results

Dealing with non-disjoint scheduling problem and two agents, in the following we propose a study on the structure of non-dominated solutions. This study provides subsequently some reflections for the design of our resolution methods and also justifies the choice of coding for proposed metaheuristics and matheuristics presented in the next chapters.

Let be the problem $Pm|ND, d_j^A = d^A|\varepsilon(\sum U_j^A/C_{max}^B)$ where agent A aims to minimize his number of tardy jobs, under the constraint that agent B's makespan does not exceed Q_B. It should be noted that this problem is equivalent to the mono-criterion scheduling problem minimizing the number of tardy jobs: $Pm|d_j, \tilde{d_j}|\sum U_j$ with:

- $\forall J_j \in \mathcal{J}^A \setminus \mathcal{J}^{A,B}$: $d_j = d^A$ and $\tilde{d_j} = UB$
- $\forall J_j \in \mathcal{J}^{A,B}$: $d_j = min(d^A, Q_B)$ and $\tilde{d_j} = Q_B$
- $\forall J_j \in \mathcal{J}^B \setminus \mathcal{J}^{A,B}$: $d_j = \tilde{d_j} = Q_B$.

where UB is an upper bound on the makespan of all jobs

Schedule σ is called Pareto optimal solution if there does not exist another solution that dominates it. On the basis of the studied problem properties, we want to determine the overall structure of the Pareto solutions. Some of these properties are a generalization of classical single machine scheduling problems. According to Q_B and d^A, jobs of agent A are submitted to at most two common due dates. Hence, it is easy to see that they have to be sequenced on each machine according to their shortest processing time order, thus to minimize the number of tardy jobs of agent A. It can also be shown that on a given machine, the tardy jobs of agent A that belong to $\mathcal{J}^A \setminus \mathcal{J}^{A,B}$, have to be scheduled last, otherwise the makespan of the agent B can be increased. So, we can write the following propositions.

According to values of d^A and Q_B we have the two following propositions.

Proposition 3 ***Case*** $d^A \leq Q_B$
If the scheduling problem $Pm|ND, d_j^A = d^A, C_{max}^B \leq Q_B|\sum U_j^A$ *accepts a solution, it is then possible to build an optimal sequence such that on each machine we have three blocks of jobs, may happen certain blocks are empty (see Figure 4.1):*

1. *Early jobs from* $\mathcal{J}^A$ *are scheduled first according to SPT rule;*
2. *The rest jobs of* $\mathcal{J}^B$ *(some of them may be are common jobs, so they are tardy jobs) form the second block and are performed contiguously closely to* Q_B*;*
3. *Tardy jobs of agent A (i.e. the remaining jobs from* $\mathcal{J}^A \setminus \mathcal{J}^{A,B}$*) are scheduled in any order at the end of the sequence and hence form the last block of jobs.*

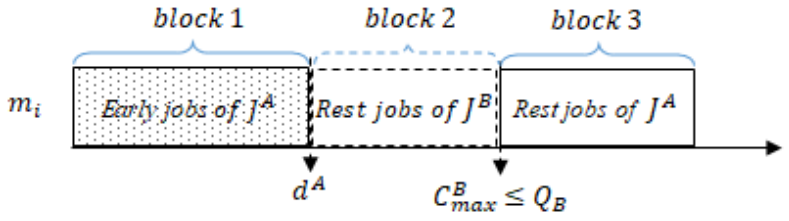

Figure 4.1: Structure of an optimal solution

Proof:

- Proof of 3.1: let's consider $J_{j_1}^A$ and $J_{j_2}^A$ two early jobs assigned to the same machine such that $p_{j_1} > p_{j_2}$. Let σ be a sequence given by the concatenation of sub-sequences π_l: $\sigma = \pi_1//J_{j_1}^A//\pi_2//J_{j_2}^A//\pi_3$). According to σ we have: $C_{j_1}(\sigma) < C_{j_2}(\sigma) \leq d^A$. The new sequence σ' is constructed by switching between the two jobs. In this case we have: $C_{j_2}(\sigma') < C_{j_1}(\sigma) \leq d^A$ and $C_{j_1}(\sigma') = C_{j_2}(\sigma)$. So, the new sequence σ' is feasible, and we have $\sum U_j^A(\sigma) = \sum U_j^A(\sigma')$. Note that $C_{max}^B(\sigma')$ may be improved as $C_{\pi_2}(\sigma') < C_{\pi_2}(\sigma)$ Hence, sequence σ' cannot be dominated by sequence σ (see Figure 4.2).

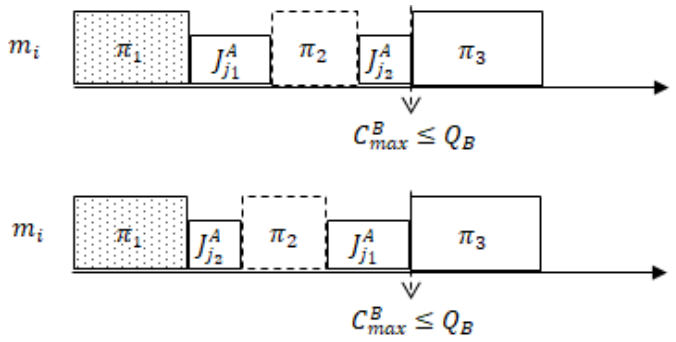

Figure 4.2: Proof of Proposition 3.1

- Proof of 3.2: Let's consider J_1^B and J_2^B two jobs of agent B assigned to the same machine such that there exist some jobs only of agent A scheduled between these two jobs. Let σ be a feasible schedule defined by the concatenation of sub-sequences π_l: $\sigma = \pi_1//J^B_{j_1}//\pi_2//J^A_j//J^B_{j_2}//\pi_3$.

  Then, we have: $C_{j_1}(\sigma) < C_j(\sigma) < C_{j_2}(\sigma) \leq Q_B$.

  If J_j^A is early then we build new sequence σ' by switching between $J^B_{j_1}$ and J_j^A. Note that in this case, $J^B_{j_1}$ is in $\mathcal{J}^B \setminus \mathcal{J}^{A,B}$; Else it is early job. Then we have: $\sigma' = \pi_1//J^A_j//\pi_2//J^B_{j_1}//J^B_{j_2}//\pi_3$. So, we have $C^A_{j_1}(\sigma') < C^A_j(\sigma)$ and then $\sum U^A_j(\sigma) = \sum U^A_j(\sigma')$ with $C^B_{max}(\sigma) = C^B_{max}(\sigma')$. If J^A_j is late job, then it is removed to be scheduled at the end of σ. This is done without deterioration of objectif value of agent A. We have: $C^B_{max}(\sigma') \leq C^B_{max}(\sigma)$. Hence, sequence σ' cannot be dominated by sequence σ (see Figure 4.3).

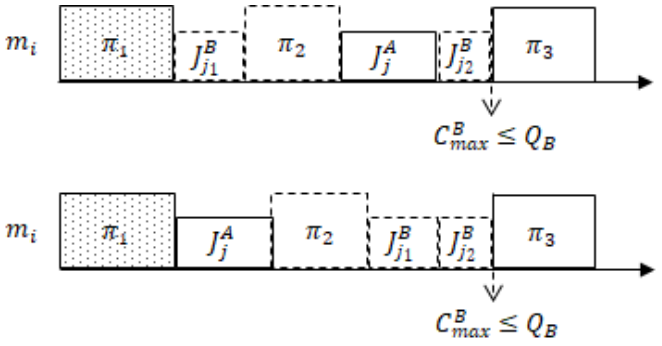

Figure 4.3: Proof of Proposition 3.3

- Proof of 3.4: trivial case.

Proposition 4 ***Case*** $d^A > Q_B$

If the scheduling problem $Pm|ND, d^A_j = d^A, C^B_{max} \leq Q_B|\sum U^A_j$ *accepts a solution, it is then possible to build an optimal sequence such that on each machine we have three blocks of jobs, may happen certain blocs are empty (see Figure 4.4):*

1. *Jobs from $\mathcal{J}^B$ are performed contiguously and define the first block of jobs;*
2. *The rest early jobs from $\mathcal{J}^A$ are scheduled in the second block according to SPT rule;*
3. *The rest jobs from $\mathcal{J}^A$ (tardy jobs) are performed contiguously and form the thrird block.*

Proof: the proof is given by pairwise interchange argument based on the same way to proof Proposition 3.

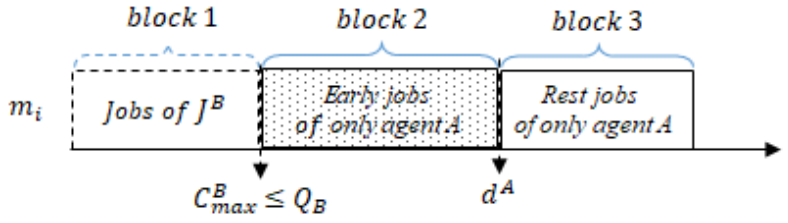

Figure 4.4: Proof of Proposition 4

Proposition 5 *Let σ a Pareto optimal solution of scheduling problem $Pm|ND, d_j^B = d^B, \sum U_j^B \leq Q_B|C^A_{max}$. It is then possible to build an optimal sequence such that on each machine we have three blocks of jobs, may happen certain blocs are empty (see Figure 4.5):*

1. *Early jobs from $\mathcal{J}^B$ are scheduled first according to SPT rule, This schedule is satisfy $\sum U_j^B \leq Q_B$ condition;*
2. *The rest jobs from $\mathcal{J}^A$ are scheduled in the second block to have C^A_{max};*
3. *The tardy jobs from $\mathcal{J}^B$ are performed contiguously in the three block of jobs.*

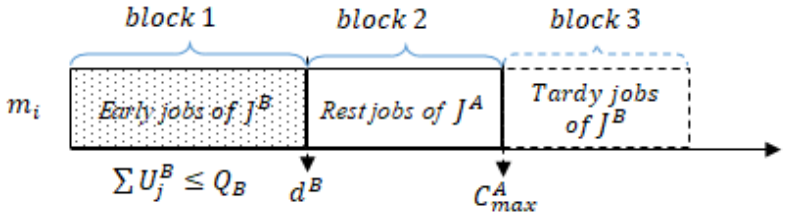

Figure 4.5: Solution optimal of one machine

Proof: Dealing with ε-constraint problem $Pm|ND, d_j^B = d^B, \sum U_j^B \leq Q_B|C^A_{max}$, to prove this proposition, it is sufficient to consider $C^A_{max}(\sigma) = y$ $(y \in [0, UB])$ as the upper bound of makespan of agent A. And hence we can follow the same steps that allowed us to show previous propositions.

4.3 Single machine multi-agent scheduling problem

Let's consider the case of single machine with only two agents A and B. In this section, the ε-constraint approach is considered. Computing the Pareto front is also discussed in this section.

4.3.1 Problem $1|ND, d_j^A = d^A, C_{max}^B \leq Q_B| \sum U_j^A$

Let's consider ε-constraint problem $1|ND, d^A, C_{max}^B \leq Q_B| \sum U_j^A$. Agent A aims to minimize the number of tardy jobs, under the constraint that the makespan of agent B does not exceed Q_B. To determine a strict Pareto optimal solution where $d^A \leq Q_B$, we propose Algorithm 9 . In the case of $d^A > Q_B$ we propose Algorithm 10.

Algorithm 9 Problem $1|ND, d^A, C_{max}^B \leq Q_B| \sum U_j^A$ with $d^A \leq Q_B$

re-index jobs of $\mathcal{J}^A$ according to SPT order, in case of equality, jobs from $\mathcal{J}^{A,B}$ have priority
$P^B = \sum_{j \in \mathcal{J}^B} p_j$
$P_0^B = \sum_{j \in \mathcal{J}^B \setminus \mathcal{J}^{A,B}} p_j$
$t = Q_B - P^B$
if $t < 0$ **then**
    STOP, no feasible solution
else
    Let S be a maximum subset of smallest jobs from $\mathcal{J}^A \setminus \mathcal{J}^{A,B}$ such that $\sum_{j \in S} p_j \leq t$
    Schedule jobs of $(S \cup \mathcal{J}^{A,B})$ according to SPT order in the time intervalle $[0, Q_B - P_0^B]$
    Remove all tardy jobs belonging subset S to the end of schedule
    Schedule remaining jobs of agent B and deduce C_{max}^B
    Schedule remaining jobs of agent A at the end of schedule
    Let L be the set of jobs scheduled before Q_B
    if $\sum_{j \in L} p_j \leq d^A$ **then**
        Jobs of $\mathcal{J}^B$ from L are scheduled first (to improve C_{max}^B)
    return solution

Proposition 6 *An optimal solution for problem* $1|ND, d^A, C_{max}^B \leq Q_B| \sum U_j^A$*, if there exists, can be computed in* $O(n_A log n_A + n)$ *running time by applying Algorithm 9 or Algorithm 10.*

Proof: Algorithm 9 is based on the Proposition 3 where Algorithm 10 is based on the Proposition 4.

- **Case** $d^A \leq Q_B$:

  We can show that the classical scheduling problem $1|d^A| \sum U_j$ with a common due date is polynomial and that optimal scheduling is therefore achieved by scheduling

jobs according to SPT rule. The first step of Algorithm 9 is that the jobs of agent A are reindexed according to the SPT rule. Without loss of generality, suppose that $Q_B \geq P^B$, it means that there is a solution where $C^B_{max} \leq Q_B$. According to step (8), we determine a set S that contains a maximum number of jobs $\mathcal{J}^A \setminus \mathcal{J}^{A,B}$, i.e. smallest jobs of only agent A, which can be scheduled in the time interval $[0, Q_B]$ without damaging makepan of agent B. To minimize the number of tardy jobs, jobs should be scheduled according to SPT order. This is done by Step (9). As $d^A \leq Q_B$, maybe some jobs of S are late. Therefore, to respect agent B's makespan, these jobs are removed to be scheduled later. Thus the structure of the solution obtained by Algorithm 9 is as described in Proposition 3 (see Figure 4.6).

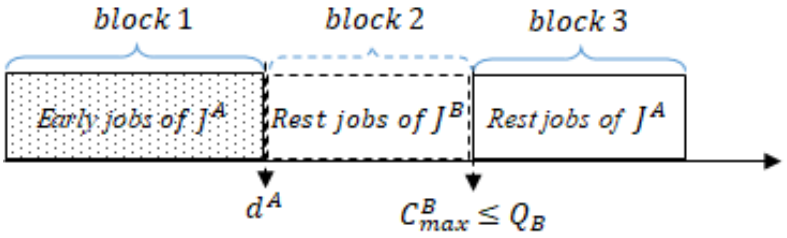

Figure 4.6: Miniminzing $\sum U^A_j$ with $d^A \leq Q_B$

Algorithm 10 Problem $1|ND, d^A, C^B_{max} \leq Q_B| \sum U^A_j$ with $d^A > Q_B$

1: re-index jobs of $\mathcal{J}^A \setminus \mathcal{J}^{A,B}$ according to SPT order
2: $P^B = \sum_{j \in \mathcal{J}^B} p_j$
3: **if** $Q_B < P^B$ **then**
4: STOP, no feasible solution
5: **else**
6: Schedule jobs of $\mathcal{J}^B$, deduce C^B_{max}
7: Let S be a maximum subset of smallest jobs from $\mathcal{J}^A \setminus \mathcal{J}^{A,B}$ such that $\sum_{j \in S} p_j \leq d^A - P^B$
8: Schedule jobs of S (S is a set of early jobs) to form the second block of jobs.
9: Schedule the rest of jobs of $\mathcal{J}^A \setminus \mathcal{J}^{A,B}$ and deduce $\sum U^A_j$
10: return solution

- **Case** $d^A > Q_B$: this case is trivial. In fact, there is only two blocs of jobs. The jobs of agent B are scheduled (maybe any order) followed by the jobs of agent A, which they are scheduling according to the SPT rule (see Figure 4.7).

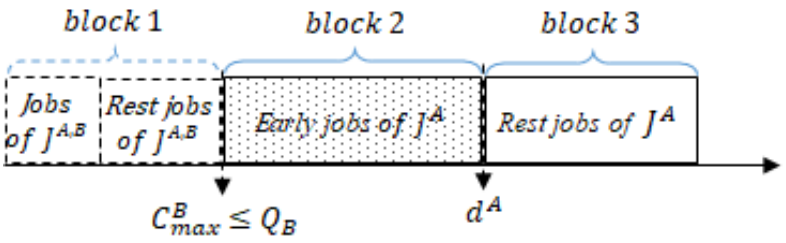

Figure 4.7: Algorithm 10: case $d^A > Q_B$

4.3.2 Problem $1|ND, d^B, \sum U_j^B \leq Q_B|C^A_{max}$

Here we address the ε-constraint problem with two agents, where agent A objective function is minimizing makespan (C_{max}) function while keeping the number of late jobs for agent B less or equal to Q_B. This problem is denoted by $1|ND, d^B|\varepsilon(C^A_{max}/\sum U_j^B)$.

Let UB be the upper bound given by $\sum_{j\in\mathcal{J}^A\cup\mathcal{J}^B} p_j$.

Let us consider the decision version of the scheduling problem denoted by:

$$1|ND, d^B, \sum U_j^B \leq Q_B, C^A_{max} \leq y|-$$

where y is a positive value in the interval $[0, UB]$. Hence an optimal solution can be obtained by applying Algorithm 9 (respectively Algorithme 10) if $y \leq d^B$ (respectively $y > d^B$) as follow: by dichotomic procedure, choose $y \in [0, UB]$; if there is a feasible (respectively no feasible) solution then the value of y is decreased (respectively increased).

Proposition 7 *An optimal solution for ε-contraint problem $1|ND, d^B|\varepsilon(C^A_{max}/\sum U_j^B)$ if there exists, can be obtained in $O(n_B log n_B + nlog(UB))$ running time.*

4.3.3 Problem $1|ND, d_j^B = d^B|P(C^A_{max}, \sum U_j^B)$

In this section we study the computing Pareto front problem. As the number of late jobs is limited by the number of jobs of agent B, we can generate the Pareto font by varying U_j^B from 0 to n_B. For each value of U_j^B, we hence solve problem $1|ND, d^B|\varepsilon(C^A_{max}/\sum U_j^B)$. Thus, we have the following proposal.

Proposition 8 *To compute Pareto front, an optimal solution for $1|ND, d^B|P(C^A_{max}, \sum U_j^B)$ if there exists, can be obtained in $O(n_B nlog(UB))$ running time.*

Proof: In fact, for each value of the number of tardy jobs of agent B in $[0, n_B]$, we slove problem $1|ND, d^B, \sum U_j^B \leq Q_B|C^A_{max}$ (see proposition 7). Hence the needed time to compute the Pareto front is $O(n_B^2 log(n_B) + n_B nlog(UB))$.

4.4 Equal length jobs

In this section, we study the non-disjoint scheduling of independent jobs where two agents compete to perform their jobs on common identical parallel machines. The jobs

have equal processing requirements and each job should scheduled before its due date d_j. With combinations of the objective functions C_{max} and $\sum U_j$, polynomial algorithms are derived to find an optimal solution that minimizes one objective function, subject to the constraint that the objective function of the second agent does not exceed a given threshold.

4.4.1 $Pm|ND, d_j^A, p_j = p|\varepsilon(\sum U_j^A / C_{max}^B)$

In this section, the agent A aims at minimizing the total number of late jobs while the makespan C_{max} of the second agent does not exceed a given threshold. The studied problem is denoted by $Pm|ND, d_j^A, p_j = p|\varepsilon(\sum U_j^A / C_{max}^B)$. Thus, every feasible solution has to verify $C_j^B \leq Q_B$, $\forall J_j \in \mathcal{J}^B$. We can now define the deadlines $\tilde{d}_j = Q_B$, $\forall J_j \in \mathcal{J}^B$. This problem is then equivalent to the mono-criterion scheduling problem minimizing the number of late jobs, when each job is fitted with due date and deadline: $Pm|d_j, \tilde{d}_j, p_j = p|\sum U_j$. Since the jobs in $\mathcal{J}^A \backslash \mathcal{J}^{A,B}$ have no deadline, we suppose that their deadline are limitless.

Let us define the cost matrix $\Delta_{(n \times n)}$, where the lines are associated with jobs, columns with positions on the machines and $\delta_{j,r}$ is the assigned cost of job J_j at position r ($\delta_{j,r}$=1 if job is late; 0 otherwise). Each position is held by one and only one job. We set $\lceil \frac{n}{m} \rceil$ positions for the $n \ mod(m)$ first machines and $\lfloor \frac{n}{m} \rfloor$ for the remaining machines. We therefore have $r = 1, \ldots, n$ positions, numbered in lexicographical order according to the machine numbers. Each position r is associated with one completion time C_r which can be computed in constant time. For position r on machine M_i, let r' be the total number of positions of the $i-1$ first machines. We have: $C_r = r \times p$ if $m = 1$; otherwise $C_r = (r - r') \times p$.

$$\delta_{j,r} = \begin{cases} 0 & \text{if } C_r \leq d_j \\ 1 & \text{if } d_j < C_r \leq \tilde{d}_j \\ \infty & \text{otherwise} \end{cases}$$

We introduce a binary variable $x_{j,r}$ that takes the value 1 if J_j is assigned to the position r, and 0 otherwise. The main problem is modeled by the following mathematical integer program.

$$(LP1) \quad \min \sum_{j=1}^{n} \sum_{r=1}^{n} \delta_{j,r} x_{j,r}$$

$$\sum_{r=1}^{n} x_{j,r} = 1 \qquad j = 1, \ldots, n \tag{4.1}$$

$$\sum_{j=1}^{n} x_{j,r} = 1 \qquad r = 1, \ldots, n \tag{4.2}$$

$$x_{j,r} \in \{0, 1\} \quad \forall j, r \tag{4.3}$$

Constraints (4.1) assign each job to only one position. Constraints (4.2) ensure that each position is occupied by a single job.

The constraint matrix of $LP1$ corresponds to the incidence matrix of bipartite graphe $G = (X_1 \cup X_2, E, \Delta)$. Where nodes in X_1 are associated with the jobs and those in X_2 are associated with the positions. There is arc $(j, r) \in E$ if $\delta_{j,r} \neq \infty$. Hence, the scheduling problem is equivalent to the minimum cost maximum matching problem, that can be solved in $O(n^{5/2})$ time [Hopcroft et Karp, 1973]. Consequently, we have the following theorem.

Proposition 9 *An optimal solution for $Pm|ND, d_j^A, p_j = p|\varepsilon(\sum U_j^A / C_{max}^B)$ if there exists, can be obtained in $O(n^{5/2})$ running time.*

4.4.2 $Pm|ND, d_j^B, p_j = p|\varepsilon(C_{max}^A / \sum U_j^B)$

Here we address the ε-constraint problem with two agents, where agent A objective function is minimizing makespan (C_{max}) function while keeping the number of late jobs for agent B less or equal to Q_B. This problem is denoted by $Pm|ND, d_j^B, p_j = p|\varepsilon(C_{max}^A / \sum U_j^B)$. See Figure 4.8.

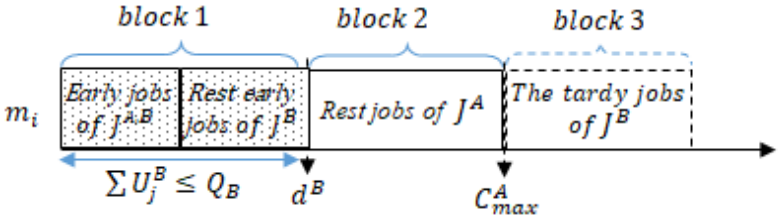

Figure 4.8: Structure of an optimal schedule of $Pm|ND, d_j, p_j = p|\varepsilon(C_{max}^A / \sum U_j^B)$

Let us consider the decision version of the problem denoted by:

$$Pm|ND, d_j, p_j = p, \sum U_j^B \leq Q_B, C_{max}^A \leq y|-$$

where y is a positive value in the interval $[\lceil \frac{n}{m} \rceil, \lceil \frac{n}{m} \rceil p]$ ($UB = \lceil \frac{n}{m} \rceil p$).

As introduced in section 4.4.1, for each position r, we can compute completion time C_r in constant time. The cost matrix Δ is as follow:

$$\delta_{j,r} = \begin{cases} \infty & \text{if } C_r > y \text{ and } j \in \mathcal{J}^A \\ 0 & \text{if } C_r > UB \text{ and } j \in \mathcal{J}^B \setminus \mathcal{J}^{A,B} \\ \begin{cases} 1 & \text{if } d_j > C_r \\ 0 & \text{if } d_j \leq C_r \end{cases} & \textit{otherwise} \end{cases}$$

Considering the following linear constraints $LP2$:

$(LP2)$

$$\sum_{j=1}^{n}\sum_{r=1}^{n}\delta_{j,r}x_{j,r} \leq Q_B \tag{4.4}$$

$$\sum_{j=1}^{n}x_{j,r} = 1 \qquad r=1,\ldots,n \tag{4.5}$$

$$\sum_{r=1}^{n}x_{j,r} = 1 \qquad j=1,\ldots,n \tag{4.6}$$

$$x_{j,r}\in\{0,1\} \quad \forall j,r, \tag{4.7}$$

Constraints (4.4) ensure that the number of late jobs is less or equal to Q_B. Constraints (4.5) assign one and only one job at each position. Constraints (4.6) assign one position to each job. The integrity constraints are given by inequalities (4.7).

In order to obtain a feasible solution, we can minimize $\sum \delta_{j,r}x_{j,r}$ subject to constraints (4.5), (4.6) and (4.7). If the best returned solution is less or equal to y, then this solution is feasible, otherwise there is no a feasible solution for the chosen y.

The constraint matrix of $LP2$ defined by (4.5) and (4.6) corresponds to an incidence matrix of bipartite graphe $G=(X_1\cup X_2, E, \Delta)$ as introduced in section 4.4.1. Hence an optimal solution can be obtained by applying a Hopcroft's procedure [Hopcroft et Karp, 1973] as follow: by dichotomic procedure, choose $y\in[0,UB]$; if there is a feasible (respect. no feasible) solution then the value of y is decreased (respect. increased).

Proposition 10 *An optimal solution for $Pm|ND, d_j, p_j = p|\varepsilon(C^A_{max}/\sum U^B_j)$ if there exists, can be obtained in $O(n^{5/2}\log(UB))$ running time.*

4.4.3 Problem $Pm|ND, d^B_j, p_j = p|P(C^A_{max}, \sum U^B_j)$

In this section we study the computing Pareto front problem. As the number of late jobs is limited by the number of jobs of agent B, we can generate the Pareto font by varying U^B_j from 0 to n_B. For each value of U^B_j, we hence solve problem $Pm|ND, d_j, p_j = p|\varepsilon(C^A_{max}/\sum U^B_j)$. Thus, we have the following proposal.

Proposition 11 *To compute Pareto front, an optimal solution for $Pm|ND, d^B_j, p_j = p|P(C^A_{max}, \sum U^B_j)$ if there exists, can be obtained in $O(n_B n^{5/2}\log(UB))$ running time.*

4.5 Mixed Integer Linear Programming

Mixed Integer Linear Programming (MILP) is a variant of mathematical programming that is used for solving scheduling problems. There are different manners of formulating a scheduling problem using mathematical programming. In this thesis, two type formulations

are considered: model based on Assignment-Precedence Variables and model based on Time-Indexed Variables.

In literature, [Balasubramanian *et al.*, 2009b] study the competing scheduling problems on parallel machines. Each agent has its own criterion to be minimized (C_{max} and $\sum C_j$). The authors propose a time-indexed integer programming formulation, heuristic and an efficient genetic algorithm.

On the basis of the studied problem properties, we have deduced the overall structure of an optimal solution giving a good compromise for agents. Note that the two following mathematical formulations led us to obtain the jobs assignment to machines with an optimal makespan for the first agent respecting all constraints, in particular the ε-constraint of the second agent. The total description of jobs sequence on each machine is obtained easily following proposals outlined above.

4.5.1 Assignment-based formulation

Dealing with ε-contraint problem $P_m|d^B, \sum U_j^B \leq Q|C_{max}^A$, we define the following decision variables:

- $x_{i,j}$ is a binary variable that takes the value 1 if job J_j is scheduled on machine M_i; 0 otherwise.

- $y_{j,k}$ is a binary variable that is equal to 1 if job J_j is executed before job J_k on the same machine; 0 otherwise.

- U_j^B is a binary variable that is equal to 1 if job J_j is scheduled after its due date d^B; 0 otherwise.

- We also need introduce continuous variables: C_j is the completion time of job J_j and C_{max}^A is the makespan of agent A (his objective function).

Let HV be a positive high value. $\mathcal{J}$ is the set of all jobs, $\mathcal{J}^B$ is the set of jobs belong to agent B. Hence, the mathematical formulation based on assignment-precedence denoted $MILP1$ is as follow: Considering the following linear constraints $LP2$:

$$
\begin{aligned}
(MILP1)\quad & Minimize\ \ C^A_{max} \\
s.t.\quad & \sum_{i=1}^{m} x_{i,j} && = 1 && \forall J_j \in \mathcal{J} && (4.8)\\
& C_j - \sum_{k=1}^{n} y_{j,k} \times p_k && \geq p_j && \forall J_j \in \mathcal{J} && (4.9)\\
& x_{i,j} + x_{i,k} - y_{j,k} - y_{k,j} && \leq 1 && \forall J_j, J_k \in \mathcal{J}; i = 1, ..., m && (4.10)\\
& x_{i,j} + y_{j,k} - x_{i,k} && \leq 1 && \forall J_j \in \mathcal{J}; i = 1, ..., m && (4.11)\\
& y_{j,k} + y_{k,l} - y_{j,l} && \leq 1 && \forall J_j, J_k, J_l \in \mathcal{J} && (4.12)\\
& C_j - d^B - U^B_j HV && \leq 0 && \forall J_j \in \mathcal{J}^B && (4.13)\\
& \sum U^B_j && \leq Q_B && \forall J_j \in \mathcal{J}^B && (4.14)\\
& C_j && \leq C^A_{max} && \forall J_j \in \mathcal{J}^A && (4.15)\\
& x_{i,j}, y_{j,k}, U^B_j \in \{0,1\} && && \forall J_j, J_k \in \mathcal{J}^A; i = 1, \ldots, m && (4.16)
\end{aligned}
$$

- Constraints 4.8 impose that each job J_j should be assigned to one and only one machine.
- Constraints 4.9 mean that each job completion time is at least equal to the total processing times of the jobs scheduled before it on the same machine plus its own processing time.
- Constrains 4.10 impose that if two jobs are performed on the same machine, then one must be scheduled before another, i.e. either job J_j scheduled before job J_k or job J_k scheduled before job J_j.
- Constraints 4.11 express that if J_j is executed on machine M_i and J_j is before J_k, then J_k must be scheduled on machine M_i also.
- Transitivity constraints are expressed by the formula 4.12, which mean that if J_j is executed before J_k, and J_k is executed before J_l than J_j is executed before J_l.
- Constraints 4.13 fixe the values of the binary variables U^B_j: if it equals to 0 then we have $C_j \leq d^B$; Otherwise the job is late and the constraint is still valid.
- Constraints 4.14 express the ε-constraint, i.e. the number of tardy jobs of agent B does not exceed Q_B.
- Constraints 4.15 express that the makespan is greater ou equal to the last completion time.
- Constraints 4.16 are the integrity ones.

4.5.2 Time-based formulation

To solve optimally the ε-constraint problem $Pm|ND, d^B, \sum U_j^B \leq Q_B|C_{max}^A$, we propose a second mathematical formulation where decision variables are time-indexed. Hence, new binary variables denoted $s_{j,t}$ are defined. $s_{j,t}$ takes value 1 if job J_j starts its processing at time t; 0 otherwise. t is in the interval $[0, UB]$ where $UB = \sum_{j \in \mathcal{J}} p_j$. Thereby, we have $n \times (UB + 1)$ binary variables, which is pseudo-polynomial number.

The mathematical formulation is then given as follow.

$$
\begin{aligned}
(MILP2) \quad & Minimize \;\; C_{max}^A \\
s.t. \quad & \sum_{t=0}^{T} s_{j,t} && = 1 && \forall J_j \in \mathcal{J}^A && (4.17)\\
& \sum_{J_j \in \mathcal{J}} \sum_{l=\max\{0, t-p_j+1\}}^{t} s_{j,l} && \leq m && \forall t = 0, \ldots, UB && (4.18)\\
& C_{max}^A - \sum_{t=0}^{T-p_j} (t + p_j) s_{j,t} && \geq 0 && \forall J_j \in \mathcal{J}^A && (4.19)\\
& \sum_{t=0}^{T-p_j} (t + p_j) s_{j,t} - HV U_j^B && \leq d^B && \forall J_j \in \mathcal{J}^B && (4.20)\\
& \sum_{J_j \in \mathcal{J}^B} U_j^B && \leq Q_B && && (4.21)\\
& s_{j,t} \in \{0,1\}, U_j^B \in \{0,1\} && && \forall J_j \in \mathcal{J}, \forall t = 0, \ldots, UB && (4.22)
\end{aligned}
$$

- Constraints 4.17 impose that each job J_j should be scheduled and hence starts at a time t.
- Constraints 4.18 ensure that no more than m jobs are scheduled simultaneously, it avoids the jobs overlapping at any time t.
- Constraints 4.19 determine the makespan value of jobs of agent A.
- Constrains 4.20 fixe the values of the decision binary variables U_j^B, which indicates if job is late or not: if $C_j \leq d^B$ then U_j^B equals to 0; Otherwise the job is late and the constrain is still valid.
- Constraints 4.21 express the ε-approach bound.
- Constraints 4.22 are the integrity ones.

4.6 Computational experiments

The two mixed integer linear programming formulations are executed on a workstation with a 2.4 GHz Intel Core i5 processor with 8 GB of memory running under Windows

10, using IBM Cplex version 12.6.2. To compute the optimal Pareto front by using the ε-constraint, computation time limit has been fixed to 3600 seconds for all different values of $Q_B \in \{0, n_B\}$.

4.6.1 Data generation

Dealing with two agents A and B, we have tested these the two mathematical formulations on data sets, which have been randomly generated with different settings described as follow:

- The number of identical machines $m = 2, 3$
- The number of jobs n is $\{10, 20, \ldots, 100\}$.
- Three types of processing times are randomly generated, denoted $Data_1$ and $Data_2$:
  - $Data_1$: processing times are randomly generated in $[1, 10]$.
  - $Data_2$: processing times are randomly generated in $[1, 100]$.
- For each (n, m), thirty instances are generated.
- The common due date d^B is randomly generated in $[0, 15P, 0, 3P]$ (to have more the differents solutions), where $P = \sum_1^n p_j$. This is a good compromise when we are dealing with at most three machines. Therefore, The common due date should be decreased when the number of machines increases to obtain interesting instances.
- For each instance, we determine the jobs of each agent and the common ones according to two parameters a_A and a_B with $a_A + a_B < 100\%$. Hence, we have: $n_A = a_A \times n$ and $n_B = a_B \times n$. The number of common jobs is thus equal to: $n_{AB} = n - (n_A + n_B)$. After several conducted experiments, the values used in the rest of this chapter are:
  - 40% of jobs in $\mathcal{J}$ are jobs of only agent A
  - 30% of jobs in $\mathcal{J}$ are jobs belong to only agent B
  - 30% of jobs in $\mathcal{J}$ are thus common jobs (belong to the two agents)

Dealing with $Data_1$:

Table 4.1 and Table 4.2 summarizes the results that indicate the performances of the two proposed MILPs to generate the exact Pareto front. We denote by $|S^*|$ the size of the exact Pareto front and CPU the needed time to compute the exact Pareto Front.

The results presented in Table 4.1 summarizes the performance of MILPs when dealing with $m = 2$. About MILPs performances, we can easily to see that the time indexed (MILP2) formulation is better than the assigned formulation (MILP1), since it solves instances with 70 jobs in 1 hour and 18 minutes on average, the average time spent to obtain the whole Pareto front. Unfortunately, the assigned indexed formulation shows weaker results since

(n, m)		MILP1		MILP2	
n	m	CPU	$\|S^*\|$	CPU	$\|S^*\|$
10	2	1,28	2,37	0,01	2,37
20	2	708,39	4,07	0,069	4,07
30	2			2,65	4,87
40	2			12,20	6,13
50	2			78,00	7,50
70	2			4664,16	9,76

Table 4.1: Computing exact Pareto front with $m = 2$ and $p_j \in [1, 10]$

it cannot solve instances with more than 20 jobs.

The results presented in Table 4.2 summarizes the performance of MILPs when dealing with $m = 3$. We remark that MILP1 can solve instances with up to 20 jobs with the time less than one hour. MILP2 seems to be better than MILP1. The MILP2 can find the optimal solutions up to 50 jobs.

(n, m)		MILP1		MILP2	
n	m	CPU	$\|S^*\|$	CPU	$\|S^*\|$
10	3	2,22	3,58	0,58	3,58
20	3	2117,22	4,71	8,85	4,71
30	3			52,31	5,25
40	3			408,01	6,09
50	3			1856,75	7,01

Table 4.2: MILP performances: exact Pareto front with $m = 3$ and $p_j \in [1, 10]$

Dealing with $Data_2$:

We note that the time indexed formulation has a pseudo-polynomial binary variables depending on the UB. With same data excepting processing times, which are generated in interval $[1, 100]$, Table 4.3 summarizes the obtained results to compute the exact Pareto front.

(n, m)		MILP1		MILP2	
n	m	CPU	$\|S^*\|$	CPU	$\|S^*\|$
10	2	87,67	3,01	46,24	3,01
20	2			587,15	4,57
30	2			2358,47	7,20
40	2				

Table 4.3: MILP performances: exact Pareto front with $m = 3$ and $p_j \in [1, 100]$

According to these results, MILP1 and MILP2 can solve instances with up to 10 jobs. In this case, MILP2 seems to be better than MILP1. However, in case where the number of jobs is greater than or equal to 20, during 1 hour running time, MILP1 cannot compute one optimal Pareto solution for a given value of Q_B, while MILP2 can calculate the exact Pareto edge for instances up to 30 jobs.

4.7 Conclusion

In this chapter, we first give a general structure of a Pareto optimal solution. This study provides subsequently some reflections to develop resolution methods and also justifies, in particular for proposed metaheuristics and matheuristics presented in the next chapters. Then, we have studied some particular cases. In the case of single machine scheduling problems, polynomial algorithms have proposed to determine Pareto optimal solution, and thus the exact Pareto front. For parallel machines, we first analyse the particular case where all jobs have the same processing times. We show that the studied problems are polynomial. These new results are summarized in Table 4.4.

Problem	Complexity	Section
$1\|ND, d^A, C^B_{max} \leq Q_B\|\sum U^A_j$	$O(n_A \log(n_A) + n)$	4.3.1
$1\|ND, d^B, \sum U^B_j \leq Q_B\|C^A_{max}$	$O(n_B \log(n_B) + n \log(UB))$	4.3.2
$1\|ND, d^A\|P(\sum U^A_j, C^B_{max})$	$O(n^2_B \log(n_B) + n_B n \log(UB))$	4.3.3
$Pm\|ND, d^A_j, p_j = p, C^B_{max} \leq Q_B\|\sum U^A_j$	$O(n^{5/2})$	4.4.1
$Pm\|ND, d^B_j, p_j = p, \sum U^B_j \leq Q_B\|C^A_{max}$	$O(n^{5/2} \log(UB))$	4.4.2
$Pm\|ND, d^A_j, p_j = p\|P(\sum U^A_j, C^B_{max})$	$O(n_A n^{5/2} log(UB))$	4.4.3

Table 4.4: New complexity results.

We propose then two mixed integer linear programming formulations to compute Pareto optimal solution for the general case. The first one is based on the assignment-precedence variables and the second is based on the time-indexed variables. To analyse their performances, some experiments are conducted where the running time is limited to one hour. The first MILP can compute an optimal solution for small instances (the number of jobs is less than or equal twenty). The second MILP is better, it solves instances with 70 jobs in 1 hour.

The next chapter is dedicated to resolution methods "heuristics" where a lower bound is proposed.

Chapter 5

Polynomial and Pseudo Polynomial Heuristics

Résumé: La complexité des problèmes étudiés suggère l'utilisation de méthodologies heuristiques. Plusieurs méthodes approchées peuvent être développées. Dans le but de développer des heuristiques efficaces pour résoudre les problèmes étudiés, nous proposons des heuristiques gloutonnes dédiées basées sur des règles de priorité et des heuristiques hybrides basées sur des méthodes exactes.

Ce chapitre est consacré à la résolution du problème $Pm|ND, d^B, \sum U_j^B \leq Q|C_{max}^A$ par heuristiques. Néanmoins, dans la section 5.1 nous montrons comment utiliser les méthodes proposées pour résoudre les autres problèmes qui font l'objet de notre étude. Nous avons donc décidé de nous concentrer sur un seul problème d'ordonnancement pour illustrer notre démarche et rendre donc ce chapitre plus facile et léger à lire. Dans la section 5.2 nous proposons une borne inférieure du makepan. Cette borne inférieure est utilisée pour d'une part, réduire le nombre d'itérations dans le calcul du front de Pareto où l'approche ε-constraint est utilisée. Dans la section 5.3, nous proposons trois heuristiques (la seconde est une amélioration de la première qui est basée sur la simple règle $LPT-FAM$). Dans la section 5.4, nous proposons une hybridation de ces méthodes avec deux méthodes exactes : la programmation dynamique et MILP. Le but de l'hybridation est donc d'améliorer la qualité des heuristiques gloutonnes. Toutes ces méthodes ont été mises en $\frac{1}{2}$uvre. Une discussion sur leurs performances est présentée dans la dernière section.

Abstract: The complexity of the studied problems suggests the use of heuristics methodologies. Several heuristics are candidates to be used. With the aim of covering different approaches, both a specific heuristic based on priority rules for this problem and hybrid heuristics based on exact methods have been developed. This chapter is devoted to solving the problem $Pm|ND, d^B, \sum U_j^B \leq Q|C_{max}^A$ by heuristics. Nevertheless, in section 5.1 we show how to use the proposed methods to solve the other problems that are the subject of our study. We have therefore decided to focus only on one scheduling problem to make this chapter easier and lighter to read. In Section 5.2 we propose a lower bound of the makespan. This lower bound is used to reduce the number of iterations to compute the Pareto front when the ε-constraint approach is used. In Section 5.3, we proposes three

heuristics (the second one is an improvement of the first one which is based on the simple $LPT - FAM$ rule). In section 5.4, we propose a hybridization of these methods with two exact methods: dynamic programming and MILP. The aim of the hybridization is thus to improve the greedy heuristics. All these methods have been implemented. A discussion on their performance is provided in the last section.

5.1 General approach for the studied problems

In this chapter, we propose specific heuristics to solve the non-disjoint multi-agent scheduling problems. However, to make this chapter easier and lighter to read, we will illustrate our resolution approaches by focusing only on the studied ε-constraint scheduling problem :

$$Pm|ND, d^B, \sum U_j^B \leq Q|C_{max}^A$$

All the methods developed in this chapter are easily generalized for the following two problems.

1. $Pm|ND, d^A, C_{max}^B \leq Q_B|\sum U_j^A$: This is the ε-constraint problem with two agents, where the objective function of agent A is minimizing the number of late jobs function ($\sum U_j^A$) while keeping the makespan for agent B less or equal to Q_B. To solve this scheduling problem it is sufficient to consider the following decision version of the scheduling problem denoted by:

   $$1|ND, d^A, C_{max}^B \leq Q_B, \sum U_j^A \leq y|-$$

   where y is a positive value in the set $\{0, n_A\}$, n_A is the number of jobs of agent A. Hence an optimal solution can be obtained by applying method proposed to solve the problem $Pm|ND, d^B, \sum U_j^B \leq Q|C_{max}^A$ as follow: by dichotomic procedure, choose $y \in \{0, n_B\}$; if there is a feasible (respectively no feasible) solution then the value of y is decreased (respectively increased). Indeed, the complexity of the algorithm increases by factor of $log(n_A)$, which is reasonable.

2. $Pm|ND, d^B, |P(C_{max}^A, \sum U_j^B)$: This is Pareto set enumeration problem. In this case we are looking for the whole set of strict Pareto optimal solutions. To determine this set of strict Pareto optimal solutions, an $\varepsilon-$constraint approach can be used, by modifying Q_B iteratively. Hence, to compute the Pareto front we can implement Algorithm 11. From Algorithm 11, UB^B is an upper bound and in worst scenario $UB^B = n_B$, LB^B is a lower bound and in worst scenario $LB^B = 0$, the function $Heuristic(Pm|ND, d^B, \sum U_j^B \leq Q_B|C_{max}^A)$ returns the solution by applying one of our proposed specific method developed for $Pm|ND, d^B, \sum U_j^B \leq Q_B|C_{max}^A$.

Algorithm 11 Enumeration of Pareto solutions of problem $Pm|ND, d^B|P(C^A_{max}, \sum U^B_j)$

1: $R = \emptyset$;
2: **for** each $Q_B = UB^B$ **down to** LB^B **step** 1 **do**
3: $\quad x = Heuristic(Pm|ND, d^B, \sum U^B_j \leq Q_B|C^A_{max})$
4: $\quad$ **if** not exit $x' \in R$ such that $C_{max}(x') \leq C_{max}(x)$ **then**
5: $\quad\quad R := R \bigcup \{x\}^*$
6: **end for**
7: Return R

Our choice was therefore to develop and analyze the performance of our approaches to solving $Pm|ND, d^B, \sum U^B_j \leq Q_B|C^A_{max}$. Computational experiments are conducted. Hence, we give the performance of proposed heuristics to compute the near Pareto front, i.e. solving problem $Pm|ND, d^B|P(C^A_{max}, \sum U^B_j)$.

5.2 Lower bound

The ε-constraint problem $Pm|ND, d^B, \sum U^B_j \leq Q_B|C^A_{max}$ is $\mathcal{NP}$-hard and exact methods proposed in chapter 3 can solve small instances. Hence, to analyse the performances of heuristics we define a lower bounds of agent's A makespan. Our proposed lower bound is based on the classical one introduced for mono-criterion scheduling problem $P_m||C_{max}$. In fact, minimizing makespan with n jobs, the lower bound is done by the formula: $LB = max\big(p_{max}; \frac{\sum_{j\in\mathcal{J}} p_j}{m}\big)$ where $p_{max} = max_{j\in\mathcal{J}}(p_j)$. Hence, to compute a lower bound of agent A makespan, we propose Algorithm 12.

Algorithm 12 Lower bound for problem $Pm|d^B, \sum U^B_j \leq Q_B|C^A_{max}$

1: Let Q_B be the number of jobs of agent B, which can be late
2: Let n_{AB} be the number of common jobs
3: $S = \mathcal{J}^B \setminus \mathcal{J}^{AB}$
4: $k = min(Q_B, n_B - n_{AB})$
5: Remove k biggest jobs from S
6: $p^S_{max} = max_{j\in S}(p_j)$
7: $p^A_{max} = max_{j\in\mathcal{J}^A}(p_j)$
8: $LB^A = max\big(p^S_{max}; p^A_{max}; \frac{\sum_{j\in\mathcal{J}^A\cup S} p_j}{m}\big)$
9: **if** $(LB^A \leq d^B)$ **then**
10: $\quad LB^A = max\big(p^A_{max}; \frac{\sum_{j\in\mathcal{J}^A} p_j}{m}\big)$
11: Return LB^A

According to Algorithm 12, it is clear that if $Q_B \geq n_B$, the minimum value of makespan of agent A is given by schedule all agent's A jobs before jobs from $\mathcal{J}^B \setminus \mathcal{J}^{AB}$, this case is treated by step (5) when $k = n_B - n_{AB}$. Otherwise we have $k = Q_B$. Therefore, at least only $(n_B - Q_B)$ smallest jobs from $\mathcal{J}^B \setminus \mathcal{J}^{AB}$ are scheduled before jobs in $\mathcal{J}^A$ where the value of the LB^A is hence given by step (8). Note that if the obtained value of LB^A

according to step (8) is less than or equal to the common due date, then jobs of agent A should be scheduled at first positions and before jobs in S according to Proposition 4. This is done by step (10). LB^A can be thus computed in $O(n_B \log n_B)$ time.

The main interest of this lower bound is the reduction of the values that makepan can take. Combined with an upper bound (heuristics), the execution time of the proposed methods to calculate the Pareto front is hence reduced, when the ε-constraint approach is used (see Algorithm 11). In other words, lower bound with upper bound allow us to approach the extreme points of the Pareto front.

5.3 List scheduling heuristics

In this section, we focus on minimizing the makespan of agent A where the total tardy jobs of agent B does not exceed a given value Q_B. Remember that one of the most effective heuristics based on the priority rule to solve the classical scheduling problem $Pm||C_{max}$ is based on the LPT rule. That is why we focus particularly on this rule by adapting it to our studied problem. We propose computation time efficient heuristics to find best solutions. The main idea of our polynomial time heuristics is:

- First, we schedule jobs of one agent to meet the $\varepsilon-$ constraints (such as $\sum U_j^B \leq Q_B$);
- Second, we try to effectively schedule the jobs of the second agent to minimize his objective function (such as C_{max}^A);
- Finally, we try to improve a solution by removing or switching certain jobs.

5.3.1 Heuristic $H1$

The heuristic presented in this section is made up of three phases, where $LPT-FAM$ rule is used (see Algorithm 13 illustrated by Figure 5.1). It starts by scheduling jobs of E, followed by the remaining jobs of agent A from $\mathcal{J}^A \setminus \{\mathcal{J}^A \cap E\}$, and finishes by sequencing jobs of agent B not already scheduled. The remaining jobs of agent B have no influence on the makespan, we can execute them at the end of any machine, but in order to optimize the number of tardy jobs it would be more efficient to use $LPT+FAM$. The time complexity of this heuristic presented in Algorithm 13 is $O(n_A \log n_A + n_B \log n_B)$.

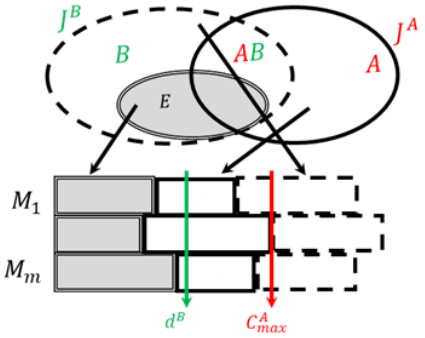

Figure 5.1: Heuristic of Upper bound

Algorithm 13 Heuristic 1 ($H1$) for problem $Pm|ND, d^B, \sum U_j^B \leq Q|C^A_{max}$

1: Sort jobs in $\mathcal{J}^B$ in SPT order.
2: Let $E = \{J_1^B, \ldots, J^B_{n_B - Q_B}\}$ be a set of potential jobs of agent B that can be early
3: Schedule jobs of E on m machines according tp $LPT - FAM$ rule
4: **if** at least one job is late **then**
5: STOP; // this heuristic cannot find a feasible solution
6: **else**
7: Schedule jobs of $\mathcal{J}^A \setminus E$ on m machines according to $LPT - FAM$ rule
8: Schedule jobs of $\mathcal{J}^B \setminus E$ on m machines according to $LPT - FAM$ rule
9: Return obtained sequence and $(C^A_{max}, \sum U_j^B)$

Example: Let's consider 6 jobs to be scheduled on 2 machines where common due date $d^B = 5$. The considered scheduled problem is denoted $P2|ND, d^B = 5, \sum U_j^B \leq 2|C^A_{max}$. The data problem are:

Job(j)	1	2	3	4	5	6
Processing time (p)	1	2	3	4	5	6
Agent	A	A,B	B	B	AB	B

First, we find the early job set E with $n_B - Q_B = 5 - 2 = 3$, That is J_2^B, J_3^B, J_4^B. Second, we schedule jobs of E on 2 machines according to $LPT - FAM$ rule.

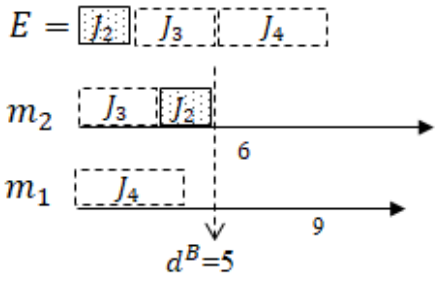

Figure 5.2: Heuristic $H1$: step 1

Third, we schedule jobs of $\mathcal{J}^A \setminus E(j_1^A, j_5^A)$ on 2 machines according to $LPT - FAM$ rule. We obtain thus the value $C^A_{max} = 9$

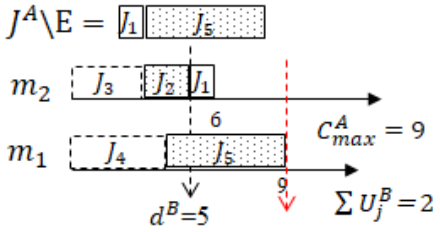

Figure 5.3: Heuristic $H1$: step 2

Finally, we schedule jobs of $\mathcal{J}^B \setminus E(j_6^B)$ on 2 machines according to $LPT - FAM$ rule.

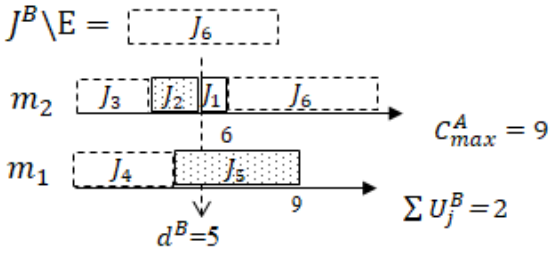

Figure 5.4: Heuristic $H1$: step 3

5.3.2 Heuristic $H2$

In this section we present a second heuristic, denoted $H2$. $H2$ is none other than an improvement of $H1$. In fact, according to $H1$, after scheduling early jobs $S = \{J_1^B, \ldots, J^B_{n_B - Q_B}\}$ maybe on a given machine M_i we have: $\delta = d^B - C^S_{ij} > 0$ (see Figure 5.5).

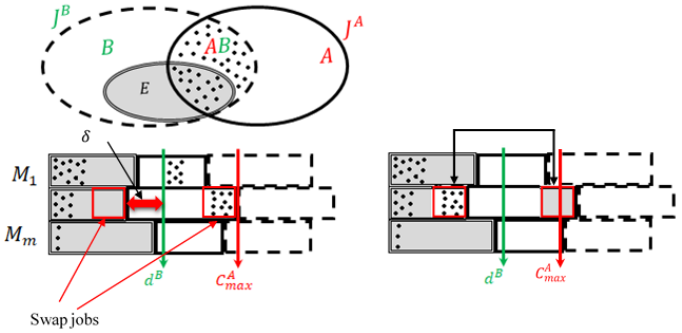

Figure 5.5: $H2$: Improving $H1$

Hence, the makespan of agent A can be improved by putting the last job of agent A

(defining his makespan) before the due date without degrading value $\sum U_j^B$. With this observation, improving $C_{max}(H1)$ can be done following two ways:

- Try to schedule late common job earlier, if possible.

- Try to schedule largest job of agent A earlier, if possible.

Let's consider previous example. We remark that the obtained solution by $H1$ (see Figure 5.4) can be improved as shown in Figure 5.6.

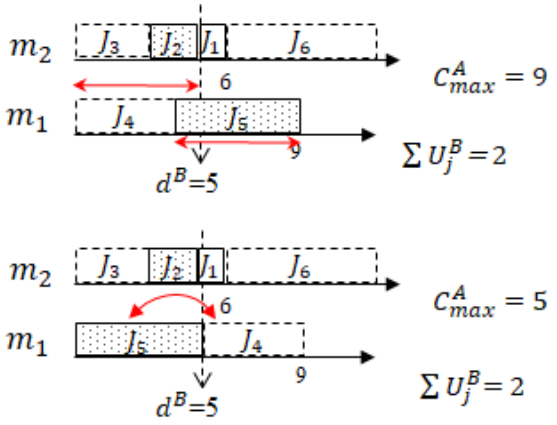

Figure 5.6: Example of improving $H1$

Unlike heuristic $H1$, $H2$ allows jobs reallocation. First of all, we schedule jobs belong to set $E_1 = E \cup \mathcal{J}^A B$ by using $LPT - FAM$ rule. At a given iteration, if the current job assigned to the first available machine M_i is late, then: if there exist largest scheduled job J_k from E_1 such that $J_k \in \mathcal{J}^B \setminus \mathcal{J}^{AB}$ (not necessary scheduled on M_i), then J_k is removed; If J_k does not exist, we remove the largest scheduled job. After, we run again $LPT - FAM$ rule by considering the set of all scheduled jobs with the new one.

This procedure is repeated until ε-constraint is satisfied (i.e. $\sum U_j^B \leq Q_B$). Then, jobs belong to E_1 and which are in $\mathcal{J}^B \setminus \mathcal{J}^{AB}$ are removed. Then we schedule the remaining jobs belong to $E_1 \cup \mathcal{J}^A$. Note that, if ε-constraint is not satisfied, this heuristic cannot find a feasible solution.

Due to the recalls of $LPT - FAM$ procedure, the time complexity of this algorithm is therefore bounded by $O(n^2)$.

Algorithm 14 Heuristic 2 ($H2$) for problem $Pm|ND, d^B, \sum U_j^B \leq Q_B|C_{max}^A$

Sort jobs belong to $\mathcal{J}^B$ in SPT order
Let $E = \{J_1^B, \ldots, J_{n_B - Q_B}^B\}$ be the potential early jobs of agent B
$E_1 = E \cup \mathcal{J}^{AB}$ sorted in LPT order.
Set $U^B = n_B$;
while $(E_1 \neq \emptyset)$ *and* $(U^B > Q_B)$ **do**
    Schedule the first job J_j in E_1 according to $LPT - FAM$ procedure
    if J_j is late **then**
        Remove scheduled job J_k belongs to $E_1 \setminus E$, where $p_k = max(p_j)$
        If such that job does not exist, remove the largest scheduled job belongs to E_1
        Recall $LPT - FAM$ procedure on the j^{th} first jobs of $E_1 \setminus \{J_k\}$
    Update U^B
if $(U^B = Q_B)$ **then**
    Schedule jobs of $\mathcal{J}^A$ not already scheduled according to $LPT - FAM$ rule
    Schedule the remaining jobs belong to $\mathcal{J}^B$ according to $LPT - FAM$ rule
    Return solution with $(C_{max}^A, \sum U_j^B)$
else
    STOP there is no feasible solution

Example: Let's consider the previous example. First, we find the early job set E with $n_B - Q_B = 5 - 2 = 3$ jobs, That is J_2^B, J_3^B, J_4^B. And $E^1 = E + \mathcal{J}^B \setminus E = J_2^B, J_3^B, J_4^B, J_5^B, J_6^B$. Second, we schedule jobs of $E1$ on 2 machines according to $LPT - FAM$ rule.

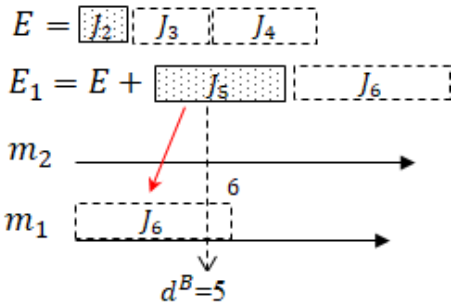

Figure 5.7: Heuristic $H2$: step 1

As job J_6^B is tardy, we remove scheduled job J_k belongs to E_1, where $p_k = max(p_j)$ and $J_k \notin E$. Hence $J_k = J_6^B$ and we continue with the next job in set $E1$.

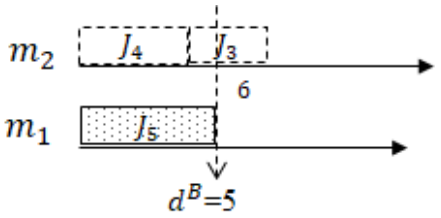

Figure 5.8: Heuristic $H2$: step 2

However, job J_3^B is tardy, we remove scheduled job J_k belongs to E_1, where $p_k = max(p_j)$ and $J_k \notin E$. Hence $J_k = J_4^B$ and we continue with the next job in set $E1$.

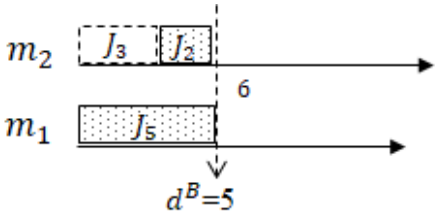

Figure 5.9: Heuristic $H2$: step 3

After that, we schedule jobs of $(\mathcal{J}^A \setminus E) = \{J_1^A\}$, $(\mathcal{J}^B \setminus E) = \{J_6^B, J_4^B\}$ on 2 machines according to $LPT - FAM$ rule. Thus we have: $C_{max}^A = 5$

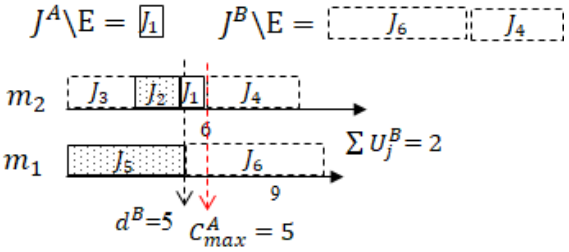

Figure 5.10: Heuristic $H2$: step 4

5.3.3 Heuristic $H3$

The basic idea of Heuristic $H3$ is based on Algorithm 12 introduced to determine the lower bound.

First, we try to find the minimum makespan of agent A where some smallest jobs of agent B are scheduled early. According to Algorithm 15, the $(n_{early} = n_B - Q_B)$ considered jobs at this step are belong to E.

Second, we schedule jobs of set E according to $LPT - FAM$ rule. If we have tardy jobs, we remove the job with largest processing time from E and replace it by smallest job from S^B. Then, we run again $LPT - FAM$ rule by considering the set of all jobs belong

E. This procedure is repeated until $C^A_{max}(E) \leq d^B$. At the end, we schedule the remaining jobs of agent A followed by the rest of jobs of agent B according to $LPT - FAM$ rule.

Due to the recalls of $LPT - FAM$ procedure, the time complexity of this algorithm is therefore bounded by $O(n^2)$.

Algorithm 15 Heuristic 3 ($H3$) for problem $Pm|ND, d^B, \sum U^B_j \leq Q_B|C^A_{max}$

Let $S^B = \{J^B_1, ..., J^B_{q_B}\}$ be the set of jobs belong $\mathcal{J}^B \setminus \mathcal{J}^{AB}$ numbred according to SPT order
Let Q_B be the maximum number of late jobs of agent B
$E = \mathcal{J}^{AB}$
$n_{AB} = |\mathcal{J}^{AB}|$
$n_{early} = n_B - Q_B$ (i.e. minimum number of early jobs)
if $n_{AB} < n_{early}$ **then**
    Add the first $(n_{early} - n_{AB})$ jobs from S^B to E
    Remove the first $(n_{early} - n_{AB})$ jobs belong to S^B from S^B
Schedule jobs of E according to $LPT - FAM$ rule
Deduce $C^A_{max}(E)$ given by the completion time of the last scheduled job belongs to E
while $C^A_{max}(E) > d^B$ **do**
    if $S^B = \emptyset$ **then**
        STOP, there is no feasible solution
    else
        $E = E \backslash \{J_k\}$ where $p_k = max_{j \in E}(p_j)$ (J_k is the largest job belongs to E)
        $E = E \cup \{J_l\}$ where $p_l = min_{j \in S^B}(p_j)$ (J_l is the smallest job belongs to S^B)
        $S^B = S^B \setminus \{J_j\}$
        Recall $LPT - FAM$ procedure by considering the new set of jobs E
        Deduce the new value of $C^A_{max}(E)$
Schedule the remaining jobs of agent A according to $LPT - FAM$ rule and deduce C^A_{max}
Schedule the remaining jobs of agent B according to $LPT - FAM$ rule and deduce U^B_j
return solution with $(C^A_{max}, \sum U^B_j)$

Example: Let's consider an example with 6 jobs and 2 machines with $d^B = 5$ and $Q_B = 1$. In this example we solve scheduling problem $Pm|ND, d^B, \sum U^B_j \leq 1|C^A_{max}$. The data problem are:

Job(j)	1	2	3	4	5	6
Processing time (p)	1	2	3	4	5	6
Due date (d)	5	5			5	5
Agent	AB	B	A	A	B	AB

First, we compute set $S^B = \{J^B_1, \ldots, J^B_{q_B}\}$. S^B is the set of jobs belong to $\mathcal{J}^B \setminus \mathcal{J}^{AB}$ numebred according to SPT order. $S^B = \{J^B_1, J^B_6\}$. Thus, $n_{AB} = 2$ and $n_{early} = n_B - Q_B = 3$. Hence $n_{AB} < n_{early}$. We add the first $(n_{early} - n_{AB}) = 1$ job belongs to S^B to

E. Hence $E = E \cup \{J_2^B\}$, and then we remove J_2^B from S^B.

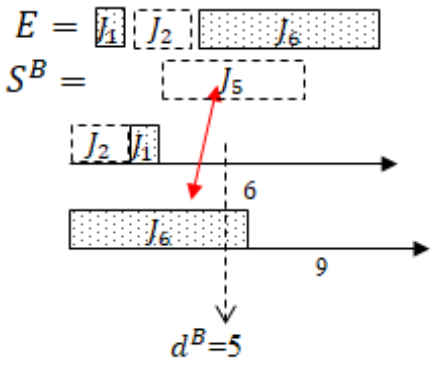

Figure 5.11: $H3$: Example step 1

Second, we apply $LPT - FAM$ procedure by considering the new set of jobs E. So, we have: $C^A_{max}(E) = 6 > d^B = 5$. We remove then job J_6^B (the largest job belongs to E). Third, we remove job J_5^B from set S^B and add it to E. We have: $C^A_{max}(E) = 5 = d^B$. S^B is now empty.

Finally, we schedule the rest jobs of agent A followed by the rest jobs of agent B on the 2 machines according to $LPT - FAM$ rule. We obtain: $C^A_{max} = 12$.

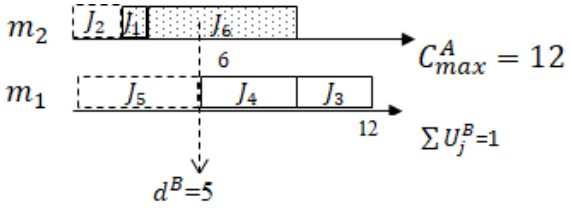

Figure 5.12: $H3$: Example step 2

5.4 Two-step heuristic methods

The complexity of the problem suggests the use of heuristics methodologies. Some specific heuristics applied to solve scheduling problems are greedy algorithms based on priority rules. These approaches ensure an efficient computational time. However, the deviation from an optimal solution can be significant. In this section we try to overcome this deficiency by optimally scheduling some subset of jobs. We propose two-step heuristic methods based on the decomposition of the problem into two decision steps.

By the first step, we call greedy specific heuristic (greedy heuristic) to determine the blocks of jobs, which they respect the structure of the optimal solution (see Propositions 3, 4, 5). In fact, according to Proposition 3, an optimal sequence is defined by three job blocks (see section 3.1.3.1): the early jobs of agent B form the first block; the remaining

jobs of agent A belong to the second jobs block; and the last jobs block contains the rest of jobs of agent B.

During the second step, the job blocks are fixed according to the preceding step. However, scheduling jobs belong each block is equivalent to the classical scheduling problem $Pm||C_{max}$. Thus, in our study we propose two exact methods to schedule the second block of jobs (dealing with remaining jobs of agent A): dynamic programming algorithm and MILP. These two exact methods are hence used to improve heuristic solution.

5.4.1 Hybrid heuristics

In this section, we propose a pseudo-polynomial time heuristic, denoted HDP, where an adapted dynamic programming algorithm is used to minimize the makespan of some jobs of agent A. The main steps of this HDP are:

- Apply heuristic ($H1, H2$, or $H3$) to determine the set of early jobs of agent B (determine the first block of jobs); Then call $LPT - FAM$ procedure to schedule these jobs. Let $(r_1, \ldots, r_i, \ldots, r_m)$ the makespan of each machine, i.e. r_i is the completion time of the last job performed on M_i.
- Optimally solve the problem $Pm|r_i|C_{max}$ by considering the remaining jobs of agent A. Note that the machines are not available at time zero due to scheduled jobs of agent B. Thus, we introduce $r_i, i = 1, \ldots, m$ in β field to indicate this constraint.

To illustrate our approach, let's consider the case of two machines. Without lost of generality, suppose that the machines are numbered according to non-decreasing order of r_i. Let $P^{\mathcal{J}}$ be the total procession times given by: $P^{\mathcal{J}} = \sum_{j \in \mathcal{J}} p_j$. To optimally solve $P2|r_i|C_{max}$, we propose the following dynamic program DP.

Let t_1 be the completion time of the last scheduled job on machine M_1, $t_1 = r_1, r_1 + 1, \ldots, P^{\mathcal{J}}$.

Let $f(j, t_1)$ be the recursion version to be minimized, where $f(j, t_1)$ gives the completion time of the last job scheduled on M_2.

The initial values are:

- $f(j, t_1) = \infty$ if $j < 0$ or $t_1 < r_1$
- $f(j, r_1) = r_2 + \sum_{l=1}^{j} p_l$

$$f(j, t_1) = min \begin{cases} f(j-1, t_1 - p_j) & (1) \\ f(j-1, t_1) + p_j & (2) \end{cases}$$

There is two decisions: jobs J_j is assigned to machine M_1 (case (1)); job J_j is assigned to machine M_2 (case (2)). The makespan is hence given by: $C_{max} = min(max_{t_1}(t_1, f(n, t_1)))$

Therefore, heuristic HDP is defined by Algotithm 16.

The time complexity of this dynamic program is $O(n_A U B^{m-1})$. Therefore, the time complexity of Algorithm 16 depends on the complexity of used heuristic. In worst case,

this complexity is bounded by $O(n^2 + n_A UB^{m-1})$, where UB is the upper bound of the makespan and m is the number of machines.

Algorithm 16 Heuristic HDP for problem $Pm|ND, d^B, \sum U_j^B \leq Q|C_{max}^A$

1: Call heuristic to solve ε-contraint problem $Pm|ND, d^B, \sum U_j^B \leq Q|C_{max}^A$
2: Let $E \subseteq \mathcal{J}^B$ be the set of jobs scheduled before d^B
3: Call $LPT - FAM$ procedure to solve $P_m||C_{max}$ by considering only set E
4: Let $(r_1, r_2, \ldots, r_m)$ be the completion times of the last jobs scheduled on $(M_1, M_2, \ldots, M_m)$.
5: Call dynamic programming algorithm to solve $P_m|r_i|C_{max}$ by considering the rest jobs of agent A
6: Schedule the tardy jobs of agent B at the end of the schedule
7: return solution with $(C_{max}^A, \sum U_j^B)$

5.4.2 Heuristics and MILP

Makespan is equivalent to the decision problem involving a common deadline (i.e. whether a feasible schedule can be obtained such that all jobs finish before a common deadline). The set of optimal solutions can give the decision-maker important information on whether jobs for one agent can be finished by a given time, and thus can give the decision-maker the resulting compromise on the number of tardy jobs for the other agent. Therefore, by applying the proposed heuristics presented in section 5.3, the set of early jobs can be computed. That means, this result satisfies the objective function that minimizes the total number of tardy jobs of agent B. The next step, the second objective function (makespan defined by agent A) should optimized. To minimize the makespan, in this section we propose to replace $LPT - FAM$ procedure by an other more effective method. In fact, we propose to use a mixed integer linear programming (MILP) to schedule the third block of jobs belong to agent A (the rest of jobs of agent A). In this case, we propose MIPL model to solve classical scheduling problem ($Pm|r_i|C_{max}$) by considering only jobs from the second jobs block.

Assignment-based formulation:

Let's consider the scheduling problem $Pm|r_i|C_{max}$, where r_i is the avalaible time of machine M_i. We define decision variables $x_{i,j}$. $x_{i,j}$ is a binary variable that takes the value 1 if job J_j is scheduled on machine M_i; 0 otherwise. The MILP model is then given as follow:

$$
\begin{aligned}
(MILP3) \quad & Minimize \ \ C_{max} & & & & \\
s.t. \quad & \sum_{j=1}^{n}(r_i + p_j x_{ij}) & \leq C_{max} & \quad \forall i = 1, \ldots, m & (5.1)\\
& \sum_{i=1}^{m} x_{ij} & = 1 & \quad \forall j = 1, \ldots, n & (5.2)\\
& x_{ij} \in \{0, 1\} & & \quad \forall i = 1, \ldots, m, \forall j = 1, \ldots, n & (5.3)
\end{aligned}
$$

- Constraints 5.1 express that the makespan is greater ou equal to the last completion time of job on each machine.
- Constraints 5.2 impose that each job J_j should be assigned to one and only one machine.

To compute a solution of ε-constraint $Pm|ND, d^B, \sum U_j^B \leq Q|C_{max}^A$, we follow the same idea as presented in section 5.4.1. This MILP3 is thus used to improve returned solution in term of C_{max}^A. The main steps of this $H - MILP$ are:

- Apply heuristic ($H1, H2$, or $H3$) to determine the set of early jobs of agent B (determine the first block of jobs); Then call $LPT - FAM$ procedure to schedule these jobs. Let $(r_1, \ldots, r_i, \ldots, r_m)$ the makespan of each machine, i.e. r_i is the completion time of the last job performed on M_i.
- Optimally solve the problem $Pm|r_i|C_{max}$ by considering MILP3 on the remaining jobs of agent A, where r_i is the availability date of machine M_i.

Thus, heuristic $H - MILP$ is defined by Algotithm 17.

Algorithm 17 Heuristic $H - MILP$

1: Call heuristic to solve ε-contraint problem $Pm|ND, d^B, \sum U_j^B \leq Q|C_{max}^A$
2: Let $E \subseteq \mathcal{J}^B$ be the set of jobs scheduled before d^B
3: Call $LPT - FAM$ procedure to solve $P_m||C_{max}$ by considering only set E
4: Let $(r_1, r_2, \ldots, r_m)$ be the completion times of the last jobs scheduled on $(M_1, M_2, \ldots, M_m)$.
5: Call MILP3 to solve $P_m|r_i|C_{max}$ by considering the rest jobs of agent A
6: Schedule the tardy jobs of agent B at the end of the schedule
7: return solution with $(C_{max}^A, \sum U_j^B)$

5.5 Computational experiments

The proposed algorithms are coded in Python language and executed on a workstation with a 2.4 GHz Intel Core i5 processor with 8 GB of memory, under Windows 10. Cplex

version 12.6.2 is used to solve the MILP models. The computation time limit has been fixed to 3600 seconds for all different values of $Q_B \in \{0, n_B\}$ (getting Pareto front).

To analyse the performances of proposed heuristics we use the same generated instances as presented in Section 4.6.1. Indeed, and according to the processing times, there are two sets of data $Data_1$ (processing times are randomly generated in $[1, 10]$) and $Data_2$ (processing times are randomly generated in $[1, 100]$). The considered numbers of machines m are: $m = 2, 3$.

5.5.1 Used performance measures

In the tables of results presented in this section, we denote by: CPU the needed time to compute the exact Pareto Front; $|S^*|$ the size of the exact Pareto front; GD is the average minimum Euclidian distance, it is used to calculate the average of the minimum Euclidian distances between an exact Pareto front and his approximate front (see section 2.5.2); And H as Hypervolume metric (see 2.5.1).

GD measure is appropriate for our situation since it requires a reference set (which in our case is the exact Pareto front) and allows comparisons in terms of closeness to the reference set. Therefore, even when the convex hull of the Pareto front is near to the convex hull of the optimal Pareto front, the average distance may be a poor indicator of the quality of the approximated front. Thus, the metric H calculates the area dominated by some front.

Note that for each table of results, each line is defined by (n, m). Thus, for each performance measure, the reported value corresponds to the average value over 30 instances.

5.5.2 Performance analyses of LB

To analyze the performance of LB in regard to an optimal Pareto front, we use MILP2. Indeed, according to the results presented at the end of the previous chapter, this model is more effective than MILP1. Here, we only consider Data_1 instances.

Table of Figure 5.13 and table of Figure 5.14 summarizes the results that indicate the performances of the $LB = (LB^A, Q_B)$ of the exact Pareto front (recall that ε-constraint approach is used).

No Jobs	No machine	MILP 2		LB			
		CPU	|S*|	CPU	%S	GD	H
10	2	0,0	2,37	0,0	77%	-0,33	-5,37
20	2	0,7	4,07	0,0	98%	-0,04	-0,87
30	2	2,7	4,87	0,0	100%	0,00	0,00
40	2	12,2	6,13	0,0	100%	0,00	0,00
50	2	78,0	7,50	0,0	100%	0,00	0,00
60	2	562,4	8,12	0,0	100%	0,00	0,00
70	2	4664,2	9,80	0,0	100%	0,00	0,00

Figure 5.13: Performance of LB: $m = 2$, $p_j \in [1, 10]$

No Jobs	No machine	MILP 2		LB			
		CPU	|S*|	CPU	%S	GD	H
10	3	0,6	3,58	0,0	48%	-0,79	-12,57
20	3	8,9	4,71	0,0	87%	-0,20	-0,63
30	3	52,3	5,25	0,0	93%	-0,05	-0,13
40	3	408,0	6,09	0,0	96%	-0,04	-0,07
50	3	1856,8	7,01	0,0	97%	-0,02	-0,05

Figure 5.14: Performance of LB: $m = 3$, $p_j \in [1, 10]$

For problem $Pm|ND, d^B|P(C^A_{max}, \sum U^B_j)$, the number of Pareto solutions increases with an increasing number of jobs when $m = 2$, while the number of Pareto solutions is rather constant and varies between 1 and 3 when $m = 3$. However, in the case of $m = 2$, dealing with instances with more than 30 jobs, the gap between the lower bound of the Pareto front and an exact Pareto front is then zero. It means that for a given value of Q_B, we have: $(C^A_{max}(LB), Q_B) = (C^A_{max}(S^*), Q_B)$.

Dealing with distance measures (GD and H) and according to the tables, it can be concluded that the average distances from the reference set are small. It can be concluded that the average distances from the reference set are very small. Even if the reported values correspond to average values, this observation remains valid when considering each dataset (note that LB gives optimal values in an average of 93% of instances such that $m = 3$ and $n \geq 20$).

5.5.3 Performance analyses of greedy heuristics

Data$_1$ instances: In table of Figure 5.15 we summarize the performance of $H1$ and $H2$ by given the deviation from lower bound of Pareto front.

No Jobs	|S(LB)|	H1				H2			
		CPU	%S	GD	H	CPU	%S	GD	H
10	2,35	0,0	43%	1,4	8,4	0,0	49%	1,1	7,1
20	4,06	0,0	34%	2,7	29,5	0,0	40%	1,4	16,4
30	4,87	0,0	24%	4,4	66,8	0,0	33%	1,6	32,4
40	6,13	0,0	16%	5,6	128,7	0,0	26%	2,0	61,6
50	8,50	0,0	14%	7,4	200,9	0,0	28%	2,2	85,2
60	9,76	0,0	12%	8,2	254,5	0,0	27%	2,1	108,8
70	15,91	0,0	11%	10,6	377,5	0,0	25%	2,4	145,9
80	20,89	0,0	9%	11,2	478,3	0,0	22%	2,5	187,3
90	24,68	0,0	7%	13,2	642,1	0,0	26%	2,5	219,0
100	31,38	0,0	7%	14,8	783,8	0,0	19%	3,1	287,4

Figure 5.15: $H1$ vs $H2$ performances with $m = 2$

As $H2$ is none other than an improvement of $H1$, by this table, we just want to indicate the efficiency of the improvement phase added in $H2$ compared to simple rules that have been used to define $H1$. We can see that the performance of $H2$ is better and better than $H1$ when the size of instances increase. Hence, in the rest of this section, we leave out $H1$.

Let's consider now only $H2$ and $H3$ heuristics where the reference set corresponds to the lower bound of Pareto front.

In table of Figure 5.16 and table of Figure 5.17 we introduce a third column which summarizes the best approximate Pareto front formed by non-dominated solutions belong to both Pareto front computed by $H2$ and Pareto front computed by $H3$.

No Jobs	|S(LB)|	H2				H3				Best(H2 & H3)			
		CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H
10	2,35	0,0	49%	1,1	7,1	0,0	54%	0,78	4,7	0,0	62%	0,63	3,6
20	4,06	0,0	40%	1,4	16,4	0,0	65%	0,55	4,1	0,0	74%	0,37	3,4
30	4,87	0,0	33%	1,6	32,4	0,0	78%	0,18	2,4	0,0	90%	0,13	1,9
40	6,13	0,0	26%	2,0	61,6	0,0	76%	0,19	2,6	0,0	87%	0,16	2,3
50	8,50	0,0	28%	2,2	85,2	0,0	83%	0,07	1,4	0,0	95%	0,06	1,3
60	9,76	0,0	27%	2,1	108,8	0,0	89%	0,00	0,0	0,0	100%	0,00	0,0
70	15,91	0,0	25%	2,4	145,9	0,0	88%	0,02	0,8	0,0	99%	0,02	0,8
80	20,89	0,0	22%	2,5	187,3	0,0	92%	0,01	0,0	0,0	100%	0,00	0,0
90	24,68	0,0	26%	2,5	219,0	0,0	92%	0,01	0,8	0,0	99%	0,01	0,8
100	31,38	0,0	19%	3,1	287,4	0,0	90%	0,00	0,0	0,0	100%	0,00	0,0

Figure 5.16: Data_1: $H2$ vs $H3$ performances with $m = 2$

No Jobs	|S(LB)|	H2				H3				Best(H2,H3)			
		CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H
10	3,58	0,0	28%	1,7	9,6	0,0	34%	1,5	7,5	0,0	36%	1,2	6,0
20	4,71	0,0	29%	1,5	15,3	0,0	47%	0,9	7,6	0,0	53%	0,6	5,7
30	5,25	0,0	36%	1,5	23,4	0,0	71%	0,6	6,4	0,0	73%	0,4	5,5
40	6,09	0,0	37%	1,6	37,7	0,0	70%	0,5	5,6	0,0	75%	0,3	4,7
50	7,01	0,0	47%	1,3	44,2	0,0	88%	0,2	3,6	0,0	90%	0,1	2,5
60	10,48	0,0	47%	1,4	54,7	0,0	93%	0,1	2,2	0,0	94%	0,1	1,8
70	13,59	0,0	46%	1,5	73,7	0,0	94%	0,1	2,6	0,0	94%	0,1	2,5
80	15,35	0,0	43%	1,5	81,7	0,0	95%	0,1	1,7	0,0	96%	0,0	1,3
90	19,08	0,0	56%	1,3	93,9	0,0	98%	0,0	1,1	0,0	98%	0,0	1,1
100	21,57	0,0	48%	1,8	129,3	0,0	96%	0,1	1,9	0,0	97%	0,0	1,4

Figure 5.17: Data_1: $H2$ vs $H3$ performances with $m = 3$

From these results, we remark that the number of Pareto solutions increases with the number of jobs when $m = 2$ (resp. $m = 3$), to reach 90% (resp. 96%). However, the distances remain significantly small. It should also be noted that these heuristics struggle to find optimal solutions for small instances (for $n = 10$, only 54% of optimal solutions are obtained by $H3$ with $m = 2$ and 34% with of optimal solutions are obtained by $H3$ with $m = 3$).

From column 3, we can conclude that the combination of the two heuristics to compute the optimal Pareto front is interesting. Even if $H2$ is much less efficient than $H3$, it happens that $H2$ dominates $H3$. For example, in the case of 100 jobs and $m = 2$, the combination of $H2$ and $H3$ allows to find 100% of non-dominated solutions belong to the optimal Pareto front. This rate is reduced to 97% when $m = 3$.

Data_2 instances: In table of Figure 5.18 and table of Figure 5.19 we summarize the best approximate Pareto front formed by non-dominated solutions belong to both Pareto front computed by $H2$ and Pareto front computed by $H3$, where $p_j = [1, 100]$. The reference set corresponds to the lower bound.

No Jobs	|S(LB)|	H2				H3				Best(H2,H3)			
		CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H
10	3,02	0,0	10%	11,6	89,5	0,0	15%	6,7	57,2	0,0	15%	6,5	47,0
20	4,57	0,0	10%	11,4	229,5	0,0	20%	4,3	65,6	0,0	24%	3,7	53,9
30	7,02	0,0	10%	8,9	316,8	0,0	19%	3,0	52,5	0,0	23%	2,5	41,0
40	10,25	0,0	14%	11,6	594,3	0,0	29%	2,9	87,3	0,0	36%	2,4	75,0
50	12,59	0,0	16%	10,3	746,8	0,0	34%	2,5	59,0	0,0	40%	1,9	42,3
60	15,13	0,0	17%	11,2	1110,4	0,0	28%	2,1	62,3	0,0	36%	1,7	48,6
70	17,08	0,0	18%	10,8	1283,0	0,0	30%	1,8	57,5	0,0	38%	1,4	42,9
80	18,87	0,0	23%	11,0	1614,9	0,0	40%	1,4	43,7	0,0	51%	0,9	31,6
90	20,98	0,0	25%	10,2	1925,7	0,0	46%	1,4	114,2	0,0	58%	0,9	101,4
100	22,56	0,0	21%	12,7	2667,2	0,0	36%	1,4	87,8	0,0	46%	0,9	72,6

Figure 5.18: Data_2: $H2$ vs $H3$ performances with $m = 2$

No Jobs	|S(LB)|	H2				H3				Best(H2,H3)			
		CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H
10	2,98	0,0	1%	11,4	97,1	0,0	1%	9,0	69,6	0,0	2%	8,2	53,4
20	3,71	0,0	3%	8,1	161,7	0,0	2%	5,9	106,8	0,0	4%	5,6	83,1
30	5,75	0,0	3%	6,4	234,4	0,0	3%	4,8	111,7	0,0	4%	4,2	88,2
40	7,19	0,0	4%	7,3	362,3	0,0	11%	3,4	101,1	0,0	13%	3,0	77,9
50	8,56	0,0	7%	6,5	374,4	0,0	10%	2,8	87,5	0,0	14%	2,3	66,8
60	10,22	0,0	6%	6,5	535,2	0,0	10%	2,9	104,8	0,0	13%	2,5	85,8
70	12,44	0,0	5%	5,9	559,4	0,0	9%	2,8	107,3	0,0	12%	2,4	82,6
80	14,25	0,0	9%	4,9	558,4	0,0	21%	2,2	83,4	0,0	26%	1,7	61,9
90	16,28	0,0	11%	5,0	633,2	0,0	30%	1,5	70,3	0,0	36%	1,2	56,3
100	19,88	0,0	11%	5,6	913,2	0,0	8%	2,8	155,7	0,0	15%	2,1	122,5

Figure 5.19: Data_2: $H2$ vs $H3$ performances with $m = 2$

Here, when processing times varying between 1 and 100, we make the same observation regarding the performance of $H3$ compared to $H2$. Nevertheless, the performance of both methods is deteriorating as the number of jobs increases. Only 15% of the strict non-dominated solutions are generated by the combining $H2$ and $H3$. We note that the distances from the lower bound remain small.

5.5.4 Performance analyses of hybrid heuristics

Let's consider now $DP(H2\&H3)$ and $MILP2(H2\&H3)$ heuristics. $DP(H2\&H3)$ (resp. $MILP2(H2\&H3)$) is the proposed hybrid pseudo-polynomial dynamic programming (resp. MILP2) heuristic. By the notation $DP(H2\&H3)$ we mean that the initial solutions correspond to the non-dominated solution obtained by $H2$ or $H3$, for each value of Q_B. In fact, these two heuristics give the same result (compute the same approximate Pareto front). Here we are focused on the CPU times needed by each method to compute this Pareto front. And, in the same times, we study the performance of $H2-H3$ by analyse the deviation from lower bound.

It should be remembered that the running time of these two methods depends strongly on the processing times. If the processing times are long, the methods require significant

computation times. That is why we were only interested on the data set $Data_1$.

In table of Figure 5.20 and table of Figure 5.17, the second column gives the average of the size of the exact Pareto front, denoted $|S^*|$; The thrid column indicates the gabs between approximate Pareto front obtained by ($H2$ or $H3$) and lower bound of the exact Pareto front; The fourth (resp. fifth) column are dealing with $DP(H2\&H3)$ (resp. $MILP2(H2\&H3)$).

No Jobs	|S(LB)|	Best(H2 & H3)				DP(H2 & H3)				MILP2(H2 & H3)			
		CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H
10	2,35	0,0	62%	0,63	3,6	0,3	74%	0,5	2,4	0,3	74%	0,5	2,4
20	4,06	0,0	74%	0,37	3,4	0,5	95%	0,1	1,1	0,4	95%	0,1	1,1
30	4,87	0,0	90%	0,13	1,9	0,7	99%	0,0	0,5	0,9	99%	0,0	0,5
40	6,13	0,0	87%	0,16	2,3	0,8	100%	0,0	0,0	0,8	100%	0,0	0,0
50	8,50	0,0	95%	0,06	1,3	1,7	100%	0,0	0,0	1,3	100%	0,0	0,0
60	9,76	0,0	100%	0,00	0,0	0,0	100%	0,0	0,0	0,0	100%	0,0	0,0
70	15,91	0,0	99%	0,02	0,8	3,5	100%	0,0	0,0	0,8	100%	0,0	0,0
80	20,89	0,0	100%	0,00	0,0	0,0	100%	0,0	0,0	0,0	100%	0,0	0,0
90	24,68	0,0	99%	0,01	0,8	4,1	100%	0,0	0,0	0,8	100%	0,0	0,0
100	31,38	0,0	100%	0,00	0,0	0,0	100%	0,0	0,0	0,0	100%	0,0	0,0

Figure 5.20: Data$_1$: $H2-3$ vs $DP(H2\&H3)$ or $MILP2(H2\&H3)$ with $m=2$

No Jobs	|S(LB)|	Best(H2 & H3)				DP(H2 & H3)				MILP2(H2 & H3)			
		CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H
10	3,02	0,0	15%	6,5	47,0	0,01	20%	5,6	35,4	0,00	20%	5,6	35,4
20	4,57	0,0	24%	3,7	53,9	0,02	57%	1,6	19,8	0,02	57%	1,6	19,8
30	7,02	0,0	23%	2,5	41,0	0,06	92%	0,2	1,6	0,05	92%	0,2	1,6
40	10,25	0,0	36%	2,4	75,0	0,11	97%	0,6	33,4	0,12	97%	0,6	33,4
50	12,59	0,0	40%	1,9	42,3	0,15	99%	0,2	2,4	0,11	99%	0,2	2,4
60	15,13	0,0	36%	1,7	48,6	0,18	99%	0,2	13,2	0,12	99%	0,2	13,2
70	17,08	0,0	38%	1,4	42,9	0,25	100%	0,0	0,0	0,13	100%	0,0	0,0
80	18,87	0,0	51%	0,9	31,6	0,32	100%	0,0	0,0	0,16	100%	0,0	0,7
90	20,98	0,0	58%	0,9	101,4	0,38	99%	0,3	74,1	0,22	99%	0,3	74,1
100	22,56	0,0	46%	0,9	72,6	0,57	100%	0,0	0,0	0,31	100%	0,0	0,0

Figure 5.21: Data$_2$: $H2-3$ vs $DP(H2\&H3)$ or $MILP2(H2\&H3)$ with $m=2$

The most important lesson here is that heuristics often find the right job blocks (those that are early and those that are tardy). However, machine assignment remains a problem since it is given by solving the classical parallel machine scheduling problem to minimize the makepan. By optimizing locally on a subset of jobs, we can improve heuristics results. However, this improvement is at the expense of the calculation time, which is getting significantly longer.

5.5.5 Conclusion

This chapter is devoted to solving the problem $Pm|ND, d^B|P(C^A_{max}, \sum U^B_j)$ by heuristics where ε- constraint approach is used. We propose a lower bound of the makespan. And

thus, a lower bound of the Pareto front can be computed. To compute this Pareto front, three greedy heuristics are proposed, where the second heuristic $H2$ is an improvement of the first one. The experiment results show that the third heuristic $H3$ outperforms $H2$. Then, we propose a hybridization of these methods with two exact methods: dynamic programming and MILP. The aim of the hybridization is thus to improve the greedy heuristics by a local optimization.

Chapter 6

Iterative methods to solve the studied scheduling problems

Résumé : L'application de méthodes exactes sur des problèmes combinatoires $\mathcal{NP}$-difficile au sens fort reste inefficace. Par conséquent, des méthodes heuristiques ont été développées dans le chapitre précédent, basées sur des algorithmes de listes SPT et LPT. Ces méthodes restent très rapides mais ne garantissent pas l'optimalité, d'autre part, elles sont généralement conçues pour construire une solution de compromis unique. C'est pourquoi nous nous sommes intéressés au développement de "Métaheuristiques". Ces méthodes constituent des algorithmes d'optimisation globale stochastiques, avec une exploration partielle de l'espace de recherche des solutions. Elles s'appuient sur des mécanismes d'intensification et de diversification de la recherche afin de converger vers la ou les solutions optimales globales. Dans ce chapitre, nous proposons deux types de métaheuristiques : Recherche Tabu et les algorithmes NSGA-II. Puis un troisième, connu sous le nom de Matheuristique, est également développée. Nous utilisons ces méthodes pour générer le front de Pareto des problèmes d'ordonnancement étudiés dans les chapitres précédents.

L'organisation de ce chapitre est la suivante : nous présentons brièvement les méthodes itératives dans la section 6.1. La recherche Tabu est décrite dans la section 6.2. La section 6.3 est consacrée à la présentation de la NSGA-II. Dans la section **??** nous présentons la matheuristique que nous avons développée. L'analyse de performance de toutes ces méthodes est exposée dans la section 6.5.

Abstract: The application of exact methods on hard combinatorial problems remains ineffective. Therefore, heuristic methods have been developed in the previous chapter, based on SPT and LPT list algorithms. These methods remain very fast but do not guarantee optimality, on the other hand, they are generally designed to build a single compromise solution. That is why we have taken an interest in "Metaheuristics". These methods constitute stochastic global optimization algorithms, with a partial exploration of the research space of solutions. They are based on mechanisms for intensifying and diversifying research in order to converge towards the overall optimal solution(s). In this chapter we propose two types of metaheuristic: Tabu Search and NSGA-II algorithms. Then a third one known as Matheuristic is also developed. We use these methods to

generate the Pareto front of the studied scheduling problem.

The organization of this chapter is as follows: we briefly introduce iterative methods in the section 6.1. The Tabu Search is described in the section 6.2. The section 6.3 is dedicated to the presentation of NSGA-II. In section 6.4 we present the developed matheuristic. Performance analyses of these methods are developed in the section 6.5.

6.1 Introduction

The principle of a traditional “iterative improvement" algorithm can be summarized as follows. Given an initial solution σ_0, which can be randomly generated, we build new solution σ_1 by locally modifying certain elements, i.e. certain decisions that were chosen according to the initial solution. The objective function values of each solution (initial and new one) are then compared to analyse the impact of this modification. If the change has led to an improvement in the value of the objective function, then σ_1 is generally accepted. σ_1 is a called ”neighbor” solution to the previous σ_0. Therefore, σ_1 becomes the new initial solution and is considered as a starting point to generate a new solution.

Otherwise, σ_1 is rejected and we try again to build another solution from σ_0. The process is made iterative until the modification process does not lead to the improvement of the last best solution (or until stopping conditions). This iterative improvement algorithm (also indicated as a classical method, or descent method) does not lead, in general, to an global optimum, but only to a local optimal solution, which constitutes the best accessible solution taking into account the initial assumptions.

To improve the efficiency of the method, it can of course be applied several times, starting again with different initial solutions, and retaining as the final solution the best obtained solution. However, this procedure significantly increases the computing time of the algorithm and may not find an optimal solution, and the gap maybe non-negligible. In general, this procedure is particularly ineffective when the number of local minima increases exponentially with the size of the problem.

To overcome the obstacle of local minima, another idea has proven to be very cost-effective, to such an extent that it is the basic step of any neighbourhood-based metaheuristics such as simulated annealing and tabu search algorithm. It is a question of sometimes accepting movements that degrade a current solution, i.e. accepting a temporary degradation of the solution. A mechanism for controlling degradation, specific to each metaheuristic, makes it possible to avoid process divergence. It then becomes possible to be extracted from the ”trap” that represents the optimal local solution, and thus to explore another more promising "valley". The "based population" metaheuristics (such as evolutionary algorithms) also introduce mechanisms for exploring different sub-areas of solutions. As mechanisms we can mention the mutation procedure in evolutionary algorithms [Dréo *et al.*, 2006].

To solve our studied scheduling problem, we develop two metaheuristics: tabu search algorithm (TS) and a non-dominated sorting genetic algorithm (NSGA-II). We develop also a matheuristic algorithm.

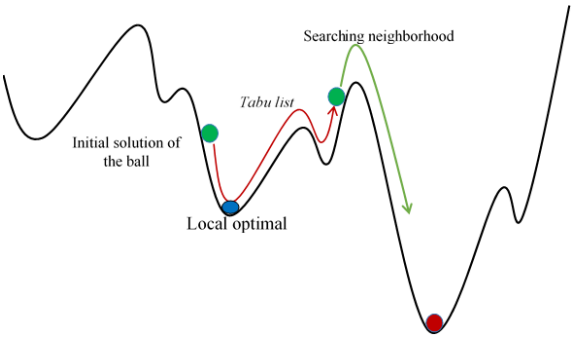

Figure 6.1: Conceptual scheme of Tabu search

6.2 Tabu search

Tabu search algorithm was introduced by Glover [Glover, 1989, Glover, 1990]. This method has been successfully applied to solve many difficult optimization problems, such as Bin Paking problem [Gourgand *et al.*, 2007], lot-sizing problems [Gourgand *et al.*, 2010], machine scheduling problems [Venditti *et al.*, 2010], RCPSP [Laurent *et al.*, 2017], vehicle routing problems [Kergosien *et al.*, 2011].

The principle of Tabu search is similar to that of iterative local search methods. It is based on the generation and evaluation of a neighborhood of solutions. Its main feature is the implementation of a mechanism combining a local search procedure with rules allowing it to overcome the obstacle of local optimal solution and at the same time avoid being trapped in a cycle (see Figure 6.1).

The tabu search deals with only one configuration at a time, which is updated at each iteration. At each iteration, the transition from one S configuration to another S' is done in two steps:

- All the neighbors $N(S)$ of S is constructed from the elementary movement of S,
- The objective function f of the problem is evaluated in each of the configurations of $N(S)$, the configuration S' that follows S is the configuration of the set $N(S)$ where f takes the minimum value.

Figure 6.2 presents the general structure of TS algorithm.

To avoid the appearance of a cycle and the return to a configuration already selected, a list of tabu movements or tabu list TL is updated. This list contains the last $|TL|$ movements performed. It means that a visited solution it is marked as "Tabu" (forbidden) so that the algorithm does not consider this solution during a certain period. So, it costs memory structures that describe the visited solutions or user-provided sets of mouvements. Through an aspiration criterion, the tabu status of solution can, however, be revoked if that would allow the search process to return to a previously visited solution and thus allows the exploration of other areas and obtain better solutions. Moreover, in its basic form, the TS method comprises less parameters of adjustment than simulated annealing, for example. This method makes it easier to use. However, the various additional mechanisms, like the intensification and diversification, bring a notable complexity.

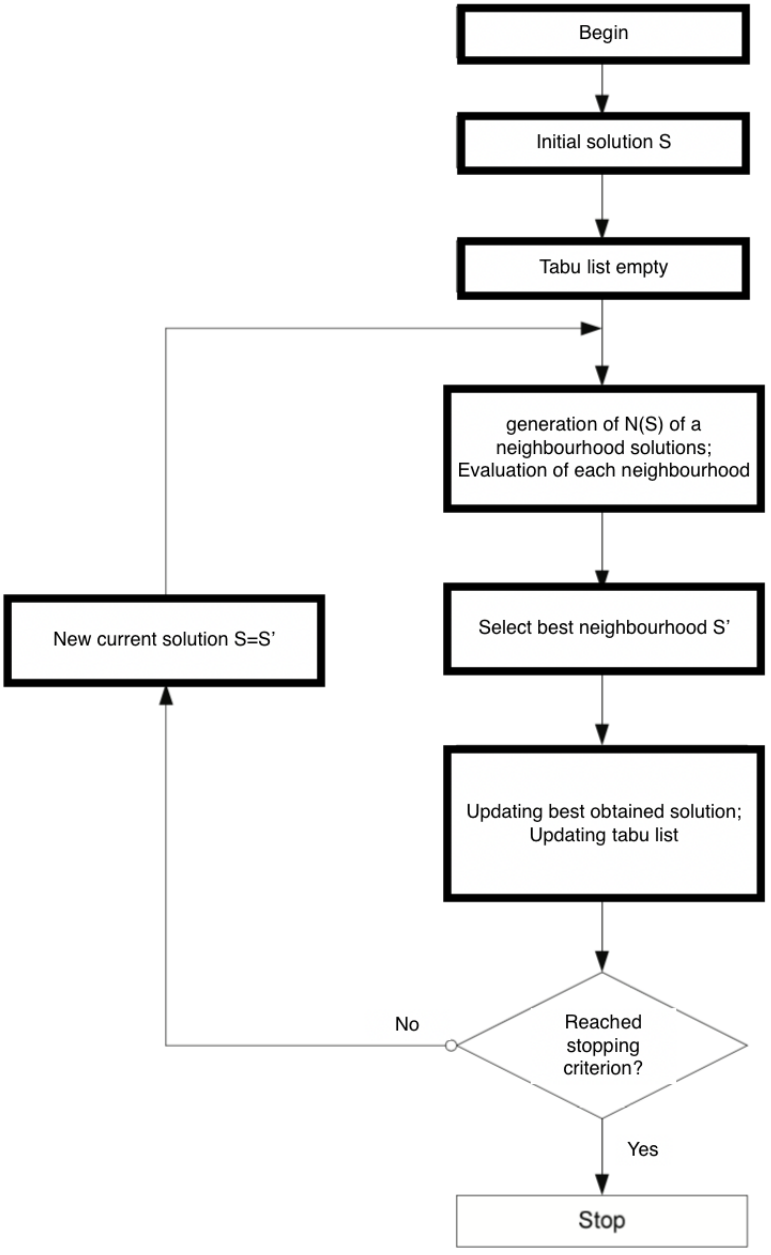

Figure 6.2: General scheme of Tabu search

In following, we show how this TS algorithm was implemented to solve the studied scheduling problems by justifying our choices.

6.2.1 Encoding mechanism

According to Propositions 3, 4 and 5, we know that given the assignment of jobs, we can compute the objective function values of a non-dominated solution for problem $Pm|ND, d^B, \sum U_j^B \leq Q_B|C_{max}^A$ in polynomial time. In fact, this can be done by adaptation of Algorithm 9 or Algorithm 10. Therefore, the proposed encoding is based on a machine-assignment scheme described as follow.

Machine-assignment encoding: A solution is an n-vector where each element i stores the number of the machine that performs job J_j. Let us consider an example with $m = 3$ machines, $n = 7$ jobs and $n_B = 5$. Let the processing times be $\{1; 2; 3; 4; 5; 3; 1\}$. Where the five first jobs are in $\mathcal{J}^B$ are numbered according to SPT order; the two last jobs are in $\mathcal{J}^A \setminus \mathcal{J}^{AB}$ (black color) are numbered according to LPT order; jobs J_2, J_3 are in $\mathcal{J}^{AB}$ (bleu color). Figure 6.3 shows the encoding and decoding of a solution. Note that in this step, we don't have call Algorithm 9 yet.

6.2.2 Decoding mechanism

Dealing with values of Q_B and d^B, from an encoding solution, we can compute the corresponding optimal values of C_{max}^A and $\sum U_j^B$ by applying algorithm 9. Note that jobs have been already sorted. Then this step can be done in $O(n)$.

Let us consider the previous example. Suppose that: $(Q_B, d^B) = (1, 5)$. Thus, Algorithm 9 returns a solution as showing in Figure 6.3.(a) where $(C_{max}^A, \sum U_j^B) = (10, 1)$.

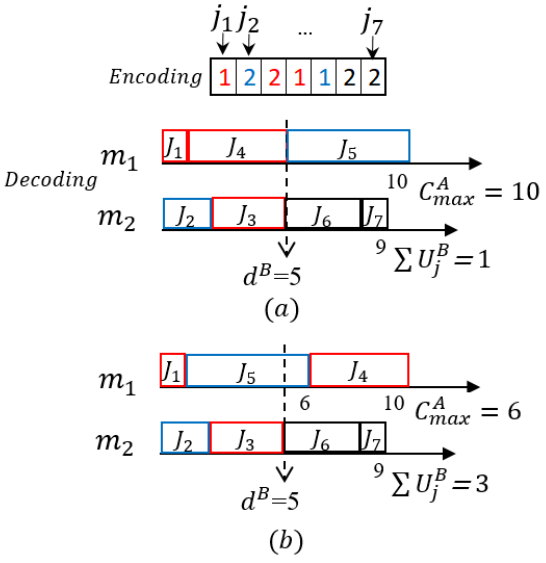

Figure 6.3: TS encoding and decoding of a solution

Let's now consider $(Q_B, d^B) = (3, 5)$. Thus, according to encoding solution presented in Figure 6.3, Algorithm 9 returns a solution as showing in Figure 6.3.(b) where $(C^A_{max}, \sum U^B_j) = (6, 3)$.

6.2.3 Initial solution

There are several ways to generate an initial solution when dealing with metaheuristic approaches: random generation, greedy heuristics, local search methods, metaheuristics, etc. In our case, the tabu search heuristic is initialized with a solution S_0 constructed by the best obtained solution given by $H2$ or $H3$ heuristics.

6.2.4 Neighborhood function

The set N(s) is generally defined as the set of solutions related to the current configuration, which we consider from a practical point of view as the set of modifications we can make to S. We call movement, a modification made to a solution. The set $N(S)$ is the set of valid solutions that can be obtained by applying a movement l belonging to the set L of possible movements. In the literature, among the possible movements that could generate eligible neighboring solutions, we quote for example: insertion, i.e. move an element from its original place to a new place; swapping of two successive elements in the initial solution; swapping of any two distinct elements.

In our study, the generation of neighbors $N(S)$ is given by modifying the assignment of job J_j executed on M_i to a machine $M_{i'}$ such that $i' = 1, \ldots, m$ and $i \neq i'$; $\forall j$. This movement allow to have a size neighborhood of $(n \times (m-1))$.

6.2.5 Tabu list

The recent selected solutions are stored in a short-term memory. In any context, there are several possibilities regarding the specific information that is recorded. For example, we can record: complete solutions; the last few transformations performed on the current solution and prohibiting reverse transformations; key characteristics of solutions.

In our study, we record the last performed mouvement on the current solution which led us to obtain a new solution.

Concerning the tabu liste size, even if some authors have proposed varying the tabu list length during the search, it has been shown that fixed-length tabu list can prevent cycling. In our case, the size of tabu list has been fixed to n, where n is the number of jobs. According to computational experiments, this offers a good compromise value with the size of instance.

6.2.6 Stopping criteria

The tabu search procedure is stopped according to two criteria:

- Number of iterations without improving the best computed solution. This value is fixed to 100 iterations that offers a good compromise according to the conducted

computational experiments.

- An optimal solution is reached (value of makespan equals to the value of lower bound)

6.2.7 Implemented tabu search algorithm

The resulting non-disjoint multi-agent scheduling algorithm to compute Pareto front is summarized in the pseudo-code of Algorithm 18. The approach ε-constraint approach is then used with $Q_B \in [LB^B, UB^B]$. Of course, to reduce the number of iterations, we first compute a lower bound by Algorithm 12 to determine $LB^A \leq C^A_{max}$.

Algorithm 18 Tabu search algorithm for $Pm|ND, d^B|P(C^A_{max}, \sum U^B_j)$

$iteration_{max} = 100$; $LT_{size} = n$
Let ND be the set of non-dominated solutions; $ND = \emptyset$
for $Q_B = n_B$ **down to** 0 **step** 1 **do**
    Compute an initial solution S_0 (non-dominated solution belongs to $H2$ and $H3$)
    $S_{best} = S_0$
    $S = S_0$
    $LT = \emptyset$
    Compute LB^A
    $iteration = 0$
    while $C^A_{max}(S) > LB^A$ AND $Iteration < iteration_{max}$ **do**
        Determine $N(S_{iteration})$ the neighbor solutions of $S_{iteration}$
        Let S be the best neighbor solution belongs to $N(S_{iteration})$
        $LT = LT \cup \{S\}$
        if $LT_{size} > n$ **then**
            Remove the oldest solution from LT
        if S *dominates* S_{best} **then**
            $S_{best} = S$
            $iteration = 0$
        else
            $iteration = iteration + 1$
    $ND = ND \cup \{S_{best}\}$
    $S_0 = S$
return set of non-dominated solutions ND

6.3 NSGA-II algorithm

Genetic Algorithm have been originally proposed by Holland and further developed by Goldberg [Holland, 1992]. This is a general search technique where a population composed by individual evolves according to some genetic operators such as: selection, crossover, mutation. Finally, each individual presents a good fitness and he is one of the population, where their organisms are more or less optimal adaptation to their environment. Several evolutionary algorithms are proposed in the literature, such as genetic algorithms, NSGA or

NSGA-II. All these approaches operate on a set of candidate solutions. An iterative process based on diversification and intensification of solutions is applied. Diversification is ensured by genetic operators, crossover and mutation which are applied to certain solutions chosen according to elitist rules. The intensification process builds on existing solutions to find those that will be considered better. At each iteration, solutions from genetic operators are inserted into the population. All solutions are then assessed on their fitness (quality of the criteria values). The method stops thanks to a stopping criterion, which can be defined by the number of iterations, the execution time or the quality of the solutions found. In the end, the best solutions are selected and presented as the result of the method. The general structure of such algorithms is illustrated in Figure 6.4.

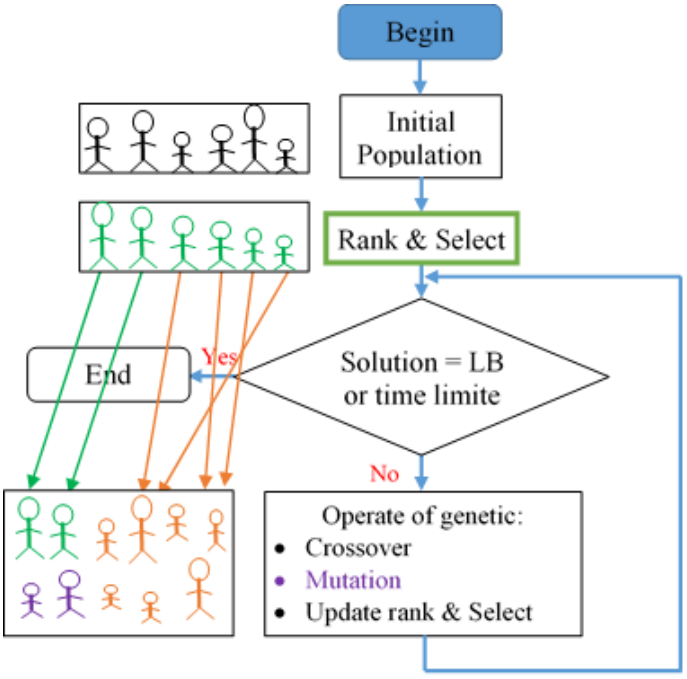

Figure 6.4: General structure of an evolutionary algorithm

In this section, we propose a second metaheuristic to compute the Pareto front of scheduling problem $Pm|ND, d^B|P(C^A_{max}, \sum U^B_j)$ based on ε-constraint approach. It is about NSGA-II algorithm. The NSGA-II procedure "Non-dominated Sorting Genetic Algorithm", is an evolutionary method often used to solve multi-criteria optimization problems. The basic idea of this method is to establish a total order between the solutions. NSGA-II was introduced by Deb et al. [Deb *et al.*, 2002]. The authors proposed a procedure for sorting solutions with a complexity in $O(K|P|^2)$ (K being the number of criteria and $|P|$ the size of the population to sort). The solutions are thus hierarchized into ranks: rank 1, rank 2, etc. It means that solutions belonging to rank k are non-dominated among themselves and cannot be dominated by a solution that belongs to rank k' such that $k' > k$. This procedure is called at each iteration.

6.3.1 Encoding

Each individual is encoded (represented) by a chromosome consisting of a gene, that is, a gene sequence. Hence, the same encoding used for tabu search algorithm is used (see Figure 6.3). It is thus machine-assignment encoding. This coding can be justified by the previous results and makes it possible to set up particularly simple crossing and mutation operators.

In the case of the m parallel machine scheduling problem with n jobs, an individual is thus defined by n-vector which gives the permutation of machines. An example resulting from this procedure is presented in Figure 6.5.

Job	J_1	J_2	J_3	J_4	...	J_n
Machine	1	3	2	2	...	1

That the corresponding solution is represented by Figure 6.5.

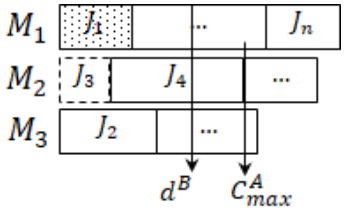

Figure 6.5: Used encoding for NSGA-II

6.3.2 Initial population

This step consists in building the initial population. The mechanism that will be adopted must be able to produce a non-homogeneous population of individuals that will serve as a basis for future generations. The first individuals were obtained by the simple gluttonous algorithms presented in Chapter 4 ($H1$, $H2$ and $H3$). To generate other individuals, we applied our crossover operators to these initial solutions.

We also generated randomly some individuals. This consists of going through each chromosome, and assigning the job one by one on a randomly decided machine.

6.3.3 Crossover operator

The crossing allows the mixing of two parental genes, the resulting genes partially inherit the characteristics of the parents. This function of operator is to enrich the diversity of the population by manipulating the structure of chromosomes in order to generate new individuals that are potentially better than individuals in the current population.

The crosses considered in our study are made between two individuals in a population and generate two children. We note $\mathcal{PC}^k$ the resulting population in step k by applying crossover operator. During the crossover operator, two parents 1 and 2 are selected from the individuals of the current population $\mathcal{P}^{k-1}$. We adopt the 2-point crossover operator.

This procedure therefore generates two new individuals. An example resulting from this crossover operator is presented in Figure 6.6.

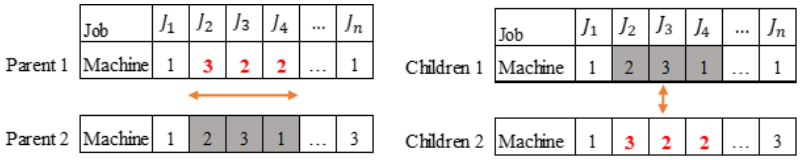

Figure 6.6: 2-point crossover operator

6.3.4 Mutation operator

For the application of this operator a random number is generated, if this number is less than the probability of mutation ρ (fixed to 5%) an individual is selected to carry out a mutation. The new individuals resulting from this operation will constitute a part of the new population.

Our mutation operator consists of randomly changing the value of one or more genes in a chromosome, as shown in Figure 6.7. This operator is described by Algorithm 19.

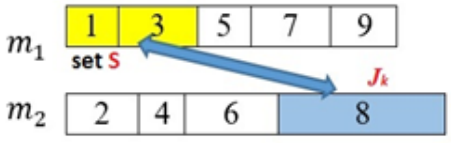

Figure 6.7: Mutation operator

Algorithm 19 Mutation operator

1: Randomly choose job J_k performed on machine M_i.
2: Randomly choose machine $M_{i'} \neq M_i$.
3: Let S_{swap} be the set of jobs scheduled on machine $M_{i'}$ such that $\sum_{j \in S_{swap}} p_j \leq p_k$
4: Assign J_k to machine $M_{i'}$
5: Assign each job belongs to S_{swap} to machine M_i
6: Add this solution to current population

6.3.5 Parameters

Parameters such as population size, mutation probability or number of crossing points are often difficult to determine, and the success of the method depends heavily on them. The mutation rate has been set at 5%. The population size was set at 100. About the stopping creteria, we use the same as introduced for tabu search algorithm. In particular, after $n_G = 100$ generations of population, the algorithm returns the found Pareto front.

6.4 Matheuristic algorithms

Matheuristic is a new type of metaheuristc algorithms, which can be defined as an hybridation of exact methods (generally based on mathematical programming models) and heuristic/metaheuristic approaches. This type of approach offers best performances when $\mathcal{NP}$-hard optimization problems are tackled. Notice that a matheuristic method can be used with any initial solution. Thus, to compute a Pareto front of the studied scheduling problem, we propose a matheuristic method where scheduling a subset of jobs of agent A is done by applying $MILP3$ introduced in section 5.4.2 and is iteratively called.

The main idea of our implemented of the proposed matheuristic is presented as follow (see Figure 6.8):

- Compute an initial solution S_0 (S_0 can be done by applying one of proposed greed heuristics)
- Compute neighbor solutions according to a neighboring function. And thus, set of early jobs of agent B is deduced (respectively the rest jobs of agent A) (see Figure 6.9.
- Call $MILP3$ to schedule rest jobs of agent A and deduce C^A_{max}

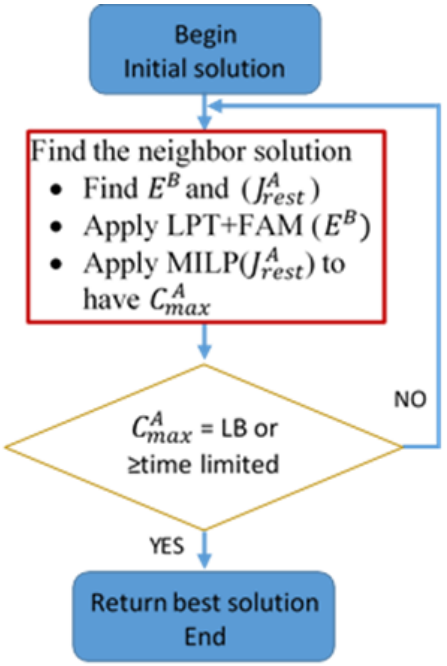

Figure 6.8: General structure of our matheuristic algorithm

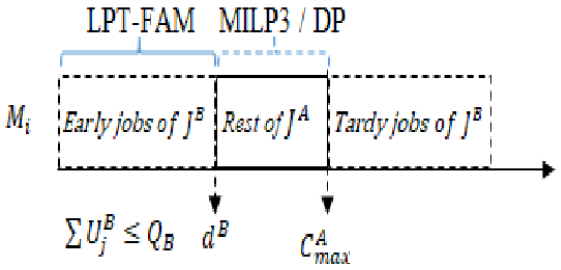

Figure 6.9: The main idea of the proposed matheuristic

6.4.1 Encoding

Dealing with this method, we introduce a new encoding procedure. Here, a solution is represented by a matrix with the size $(2 \times n)$ as showing by Figure 6.10. The first line contains the position number of job (position-job encoding) and the second one contains the machine number to indicate on witch job J_j is executed (assignent-machine encoding). Figure 6.10 shows the encoding of an instance with 7 jobs and tree machines.

Job	1	2	3	4	5	6	7
Position	1	1	1	2	2	2	3
Machine	2	3	1	1	2	3	1

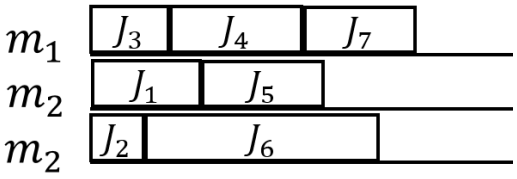

Figure 6.10: Combining position-job encoding with assignent-machine encoding

6.4.2 Initial solution

Initial solution S_0 corresponds to the best solution obtained by applying proposed greedy heuristics $H2$ and $H3$.

6.4.3 Neighborhood function

The set $N(S)$ is the set of solutions related to the current solution. In our study, the generation of neighbors $N(S)$ of solution S is given by two different movements: swapping of any two distinct elements; swapping of two successive elements.

1. **Swapping of any two distinct elements**: From the encoding matrix, we interchange the two columns J_{j_1} and J_{j_2}, i.e. copy the column J_{j_1} (respectively J_{j_2})

into the column J_{j_2} (respectively J_{j_1}). This procedure is denoted $Swap_{jobs}(J_{j_1}, J_{j_2})$, $\forall (J_{j_1}, J_{j_2}) \in \mathcal{J}^2$. This procedure builds n^2 new solutions (see Figure 6.11).

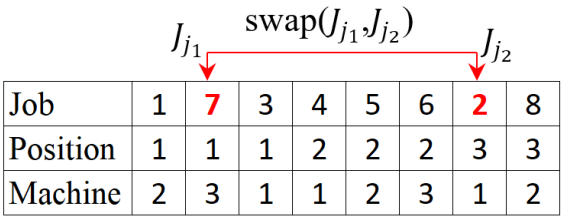

Figure 6.11: Swapping of any two distinct elements

2. **Swapping of two successive elements**: This is done only on the assignent-machine encoding (line 2 of encoding matrix). We interchange the first half part of this line with the second one, i.e. copy the $n/2$ first (respectively last) elements into the last (respectively first) half part. This procedure is denoted $Swap_{machines}(S)$. Dealing with one solution, this procedure builds a new one (see Figure 6.12).

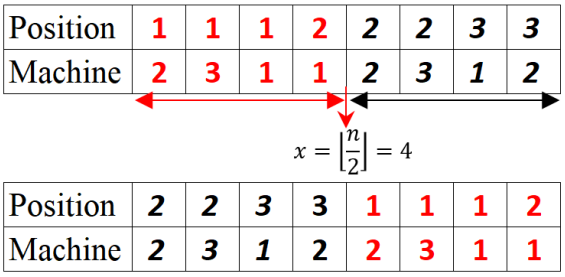

Figure 6.12: Swapping of two successive elements

The $2n^2$ solutions belong $N(S)$ are then generated by applying Algorithm 20.

Algorithm 20 Computing neighbor set

Let S_0 an initial solution
$N(S_0) = \emptyset$
for each $(J_{j_1}, J_{j_2}) \in \mathcal{J}^2$ **do**
    $S = Swap_{jobs}(J_{j_1}, J_{j_2})$
    $N(S_0) = N(S_0) \cup \{S\}$
for each $S \in N(S_0)$ **do**
    $S = Swap_{machines}(S)$
    $N(S_0) = N(S_0) \cup \{S\}$
return $N(S_0)$

6.4.4 Implemented matheuristic algorithm

Our matheuristic algorithm has been coded with the following parameters:

- Initial solution S_0 corresponds to the non-dominated solution belongs to solutions computed by $H2$ and $H3$
- Two stopping criteria are considered: First one when a solution is optimal, it means that makespan is equal to LB^A. The second one is the time-limit. Time-limit is thus fixed to: $time - limit = n^8/(5.10^{11}) + 0,09$. This value is approached with the time needed by exact method to compute solution for 10 to 40 jobs.

Algorithm 21 gives the pseudo-code of the coded matheuristic.

Algorithm 21 Mathheuristic for $Pm|ND, d^B|P(C^A_{max}, \sum U^B_j)$

$iteration_{max} = timeLimit$
Let ND be the set of non-dominated solutions; $ND = \emptyset$
for $Q_B = n_B$ **down to** 0 **step** 1 **do**
    Compute an initial solution S_0 (non-dominated solution belongs to $H2$ and $H3$)
    $S_{best} = S_0$
    $S = S_0$
    Compute LB^A
    $iteration = 0$
    while $C^A_{max}(S) > LB^A$ AND $Iteration < iteration_{max}$ **do**
        Call Algorithm 20 to generate $N(S_{iteration})$, neighbor solutions of $S_{iteration}$
        for all $S \in N(S)$ **do**
            Let $E(S)$ be the early jobs belong $\mathcal{J}^B$
            $RA(S) = \mathcal{J}^A \setminus E(S)$ ($RA(S)$ is the rest of jobs belong $\mathcal{J}^A$)
            Schedule jobs belong to $E(S)$ according to $LPT - FAM$ rule
            Call $MILP3$ on jobs $RA(S)$
            Schedule the remaining jobs of agent B to obtain new solution S
            According to S, update S_{best} and ND
        endfor
        Update $Iteration$
    endwhile
endfor
return set of non-dominated solutions ND

6.5 Computational experiments

The proposed algorithms are coded in Python language and executed on a workstation with a 2.4 GHz Intel Core i5 processor with 8 GB of memory, under Windows 10.

Getting Pareto front, all methods have been tested with different values of $Q_B \in \{0, n_B\}$. To analyse the performances of proposed methods we use the same generated instances as presented in Section 4.6.1. Indeed, and according to the processing times, there are two sets of data: $Data_1$ (processing times are randomly generated in $[1, 10]$) and $Data_2$ (processing times are randomly generated in $[1, 100]$). The considered numbers of machines m are: $m = 2, 3$.

The tables of results presented in this section are defined as follow:

1. The first column corresponds to the number of jobs, denoted n
2. The second column gives the average on size of the Pareto front, denoted $|S(LB)|$. Here we consider the lower bound of the Pareto front computed according to Algorithm 12 where $Q_B = 0, \ldots, n_B$.
3. The last four columns indicate the performance of following heuristics: Best($H2\&H3$), $NSGA-II$, tabu search algorithm TS and Matheuristic, in this order. For each method, we give: CPU (the needed time to compute the exact Pareto Front); $\%S$ (the average percentage of strict Pareto solutions); GD (the averaged of an Euclidien distance); and H (the average of the hypervolume), in this order.

Note that for each dataset (n, m), 30 instances were generated. The average values are therefore over 30 instances.

GD measure is appropriate for our situation since it requires a reference set (which is the lower bound of Pareto front) and allows comparisons in terms of closeness to this reference set. Therefore, even when the convex hull of the Pareto front is near to the convex hull of the optimal Pareto front, the average distance may be a poor indicator of the quality of the approximated front. Thus, the metric H calculates the area dominated by some front.

Data_1 instances:

Tables of Figure 6.13 and Figure 6.14 we summarize the performances of the considered resolution methods by given the deviation from the lower bound. Here, processing times are randomly generated in $[1, 10]$.

No Jobs	|S(LB)|	Best(H2 & H3)				NSGA-II				TS				Matheuristic			
		CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H
10	2,35	0,0	62%	0,63	3,57	0,2	74%	0,37	1,77	0,2	74%	0,33	1,60	0,09	76%	0,29	1,37
20	4,06	0,0	74%	0,37	3,40	1,2	94%	0,08	1,07	0,9	96%	0,06	0,93	0,12	96%	0,06	0,93
30	4,87	0,0	90%	0,13	1,93	2,3	98%	0,04	1,23	1,4	99%	0,02	0,50	1,39	99%	0,02	0,50
40	6,13	0,0	87%	0,16	2,33	3,0	100%	0,00	0,00	2,5	100%	0,00	0,00	0,00	100%	0,00	0,00
50	8,50	0,0	95%	0,06	1,27	4,3	100%	0,00	0,00	4,7	100%	0,00	0,00	1,27	100%	0,00	0,00
60	9,76	0,0	100%	0,00	0,00	0,0	100%	0,00	0,00	0,0	100%	0,00	0,00	0,00	100%	0,00	0,00
70	15,91	0,0	99%	0,02	0,83	32,5	99%	0,01	0,77	23,9	100%	0,00	0,00	0,83	100%	0,00	0,00
80	20,89	0,0	100%	0,00	0,00	0,0	100%	0,00	0,00	0,0	100%	0,00	0,00	0,00	100%	0,00	0,00
90	24,68	0,0	99%	0,01	0,77	126,9	100%	0,00	0,00	68,1	100%	0,00	0,00	0,77	100%	0,00	0,00
100	31,38	0,0	100%	0,00	0,00	0,0	100%	0,00	0,00	0,0	100%	0,00	0,00	0,00	100%	0,00	0,00

Figure 6.13: Data_1 with $m = 2$: Iterative methods performances

TS algorithm offers almost the same results as the Matheuristic method, except for small size instances. For exemple, dealing with 10 jobs and $m = 2$ (see table of Figure 6.13, the average percentage of strict Pareto solutions is improved by Matheuristic algorithm to reach 76% versus 74% for strict Pareto solutions computed by TS algorithm or by NSGA-II algorithm. Note that the average percentage of the the initial size of the approximate Pareto front (initial solutions computed by $H2\&H3$) is 62%, which is improved by the iterative methods.

We also notice that the two performance measures GD and H are improved regarding to the initial Pareto front, which means that the approached Pareto front is almost identical to the lower bound (the maximum average deviation is $1,4$ in term of hypervolume H and $0,3$ in term of GD). It is worth noting the performance of NSGA-II, even if the Pareto front obtained by the NSGA-II does not dominate those obtained by TS and Matheuristic. Also note the performance (as already seen) of the combination of the two heuristics ($H2\&H3$). As ($H2\&H3$) provide strict Pareto solutions for some values of Q_B, the computation times of these iterative methods are significantly improved (0 sec, indicates that the initial solution is an optimal solution in the Pareto sense).

No Jobs	|S(LB)|	Best(H2 & H3)				NSGA-II				TS				Matheuristic			
		CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H
10	3,58	0,0	36%	1,19	6,00	0,3	43%	1,03	5,17	0,6	43%	1,06	5,30	0,09	46%	0,91	4,33
20	4,71	0,0	53%	0,63	5,67	22,7	68%	0,43	4,17	22,1	68%	0,42	4,10	1,41	87%	0,20	2,73
30	5,25	0,0	73%	0,43	5,47	105,6	83%	0,30	4,10	55,6	84%	0,28	4,00	1,38	94%	0,11	2,20
40	6,09	0,0	75%	0,33	4,67	34,7	89%	0,17	2,57	65,3	89%	0,17	2,53	8,55	99%	0,02	1,03
50	7,01	0,0	90%	0,13	2,50	119,6	95%	0,07	1,37	49,3	94%	0,08	1,41	1,77	100%	0,00	0,00
60	10,48	0,0	94%	0,07	1,83	250,7	95%	0,06	1,63	53,7	95%	0,06	1,65	2,30	100%	0,00	0,00
70	13,59	0,0	94%	0,08	2,47	370,6	96%	0,05	2,20	114,0	97%	0,05	2,13	71,82	99%	0,01	1,13
80	15,35	0,0	96%	0,05	1,33	479,1	97%	0,03	0,87	113,2	98%	0,03	0,80	3,10	100%	0,00	0,00
90	19,08	0,0	98%	0,02	1,07	714,8	99%	0,01	0,93	95,2	100%	0,00	0,00	3,20	100%	0,00	0,00
100	21,57	0,0	97%	0,03	1,40	829,2	97%	0,03	1,40	401,2	97%	0,23	0,10	3,20	100%	0,00	0,00

Figure 6.14: Data_1 with $m = 3$: Iterative methods performances

From results presented in table of Figure 6.14 when $m = 3$ we remark that NSGA-II and TS algorithms offer the same performances. However, the observation remains the same as in the case of 2 machines: Matheuristique slightly dominates the other methods, but with a slightly longer computation time than the other methods, for some instances.

Data_2 instances:

Tables of Figure 6.15 and Figure 6.16 summarize the performances of the considered resolution methods by given the deviation from the lower bound. Here, processing times are randomly generated in $[1, 100]$.

No Jobs	|S(LB)|	Best(H2 & H3)				NSGA-II				TS				Matheuristic			
		CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H
10	3,02	0,0	15%	6,5	47,0	0,8	24%	4,53	26,13	0,8	26%	4,32	24,43	32,13	29%	4,2	20,6
20	4,57	0,0	24%	3,7	53,9	3,7	58%	1,68	19,00	1,1	73%	1,20	15,47	87,57	84%	0,9	11,7
30	7,02	0,0	23%	2,5	41,0	39,7	82%	0,40	3,20	95,7	99%	0,13	0,90	75,15	99%	0,1	0,7
40	10,25	0,0	36%	2,4	75,0	69,7	86%	0,75	36,03	104,0	97%	0,59	33,43	68,88	97%	0,5	28,4
50	12,59	0,0	40%	1,9	42,3	71,6	89%	0,41	6,27	235,5	99%	0,23	2,37	71,12	99%	0,2	2,4
60	15,13	0,0	36%	1,7	48,6	83,9	83%	0,50	19,20	201,7	99%	0,24	13,20	118,25	99%	0,2	12,4
70	17,08	0,0	38%	1,4	42,9	105,8	82%	0,33	9,67	324,1	100%	0,00	0,00	2,51	100%	0,0	0,0
80	18,87	0,0	51%	0,9	31,6	114,2	80%	0,31	10,57	309,0	100%	0,00	0,00	3,11	100%	0,0	0,0
90	20,98	0,0	58%	0,9	101,4	559,2	83%	0,51	80,20	388,5	99%	0,29	74,10	26,89	99%	0,2	54,1
100	22,56	0,0	46%	0,9	72,6	587,3	81%	0,33	44,03	220,0	100%	0,00	0,00	6,27	100%	0,0	0,0

Figure 6.15: Data_2 with $m = 2$: Iterative methods performances

No Jobs	|S(LB)|	Best(H2 & H3)				NSGA-II				TS				Matheuristic			
		CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H	CPU	%S	GD	H
10	2,98	0,0	2%	8,20	53,37	18,8	9%	7,57	44,60	0,9	10%	7,40	46,43	38,91	13%	7,23	42,70
20	3,71	0,0	4%	5,56	83,10	59,2	16%	4,08	54,23	24,1	16%	4,16	56,20	106,55	24%	2,52	33,23
30	5,75	0,0	4%	4,23	88,23	95,0	21%	2,87	46,10	81,3	27%	2,81	46,23	102,01	60%	0,73	12,30
40	7,19	0,0	13%	3,00	77,87	141,6	35%	2,01	47,03	115,8	43%	1,90	46,67	79,59	98%	0,20	10,47
50	8,56	0,0	14%	2,34	66,83	274,8	40%	1,45	35,97	186,0	46%	1,39	35,33	40,46	100%	0,00	0,00
60	10,22	0,0	13%	2,46	85,77	119,7	35%	1,59	56,37	196,2	42%	1,51	54,27	38,25	99%	0,18	17,87
70	12,44	0,0	12%	2,43	82,57	495,1	38%	1,48	45,17	236,4	44%	1,37	42,13	3,30	100%	0,00	0,00
80	14,25	0,0	26%	1,74	61,87	342,1	48%	1,13	37,60	176,6	53%	0,98	32,67	3,70	100%	0,00	0,00
90	16,28	0,0	36%	1,24	56,30	451,7	60%	0,77	33,63	316,8	67%	0,69	31,30	5,20	100%	0,00	0,00
100	19,88	0,0	15%	2,12	122,47	503,2	68%	1,40	86,70	157,0	73%	1,27	81,13	6,90	100%	0,00	0,00

Figure 6.16: $Data_2$ with $m = 3$: Iterative methods performances

Here, when processing times varying between 1 and 100, we make the same observation regarding to the performance of Matheuristic compared to other methods. In the case of $m = 2$ (resp. $m = 3$) with instances of size $n \geq 30$ jobs (resp. $n \geq 40$ jobs), Matheuristic allows to find more than 97% in average of non-dominated solutions.

Nevertheless, the performances of both TS algorithm and NSGA-II algorithm is deteriorating as the number of jobs increases in term of the number of strict Pareto solutions found. Only in average 10% (resp. 9%) over 3 strict Pareto solutions (note that here their corresponding criteria values are lower bounds) are computed. However, we note that the average distances (GD and H) from the lower bound remain small.

6.6 Conclusion

In this chapter, we have focused on studying job scheduling problems with two agents, some of which are common, using a Pareto approach. We presented the Tabu search algorithm and then the $NSGA-II$ method. The chosen encoding depends on the structure of the problem solutions. Indeed, for problems minimizing the makepan and the number of late jobs, we have previously shown that from an assignment of jobs to machines we can calculate a Pareto solution in polynomial time. The encoding of such a solution is thus done with a $n-$vector.

Another encoding was proposed giving information on the positions of jobs on the machines. Thus, the proposed encoding is defined by a matrix $(n \times 2)$. This coding is used within the third developed method: Matheuristic. The initial solution (initial population) is performed using proposed greedy heuristics ($H1, H2$ and $H3$). Dealing with $NSGA-II$, a part of initial population is (semi-)randomly performed.

In order to analyse the performance of the proposed methods, we have taken up the instances presented in the previous chapters. The comparison between the lower bounds of the Pareto fronts and the approximated fronts reveals that the proposed methods are effective, in particular the Matheuristic method, since the generational distances are very small compared to the values of the criteria.

Chapter 7

Conclusions and future research directions

In this thesis we tackle multiagent scheduling problems, with two objective functions. The first is minimizing the makespan of one agent and the second is minimizing the total number of tardy jobs with common due date for the other agent. More precisely, we study the non-disjoint scenario. Only two agents are considered in our study. Each one has his own jobs. May happen that some jobs are common. The agents share the same resources to schedule their own jobs. The considered resources in our study are identical parallel machines.

This problem is shown to be $\mathcal{NP}$-hard. To compute the Pareto front, the ε-constraint approach is investigated.

However, in the case of single machine, we show that the Pareto front can be computed in polynomial time. The case of equal length jobs is also studied. We show that the problem is equivalent to the minimum cost maximum matching problem, which is polynomially solvable. Table 7.2 summarizes the polynomial cases.

Problem	Complexity	Section
$1\|ND, d^A, C^B_{max} \leq Q_B\|\sum U^A_j$	$O(n_A \log(n_A) + n)$	4.3.1
$1\|ND, d^B, \sum U^B_j \leq Q_B\|C^A_{max}$	$O(n_B \log(n_B) + n \log(UB))$	4.3.2
$1\|ND, d^A\|P(\sum U^A_j, C^B_{max})$	$O(n^2_B \log(n_B) + n_B n \log(UB))$	4.3.3
$Pm\|ND, d^A_j, p_j = p, C^B_{max} \leq Q_B\|\sum U^A_j$	$O(n^{5/2})$	4.4.1
$Pm\|ND, d^B_j, p_j = p, \sum U^B_j \leq Q_B\|C^A_{max}$	$O(n^{5/2} \log(UB))$	4.4.2
$Pm\|ND, d^A_j, p_j = p\|P(\sum U^A_j, C^B_{max})$	$O(n_A n^{5/2} log(UB))$	4.4.3

Table 7.1: New complexity results.

Dealing with general case, several resolutions methods were developed:

- Two exact methods $MILP1$ and $MILP2$.

- Three greedy heuristics: $H1, H2$ and $H3$.
- Two hybrid heuristics combining greedy heuristics and exact methods (dynamic programming algorithm and $MILP$).
- Three iterative improvement methods: Tabu search, $NSGA-II$ and Matheuristic.

We use the different metrics to compare the performances of these methods such as: size of Pareto front, generational distance and Hypervolume. The greedy algorithm $H3$ gives best results than the proposed greedy heuristics, where hybridization method of heuristic with $MILP$ outperforms the other one based on dynamic programming algorithm. About the iterative improvement methods, the experiment results show that the best method is Matheuristic.

Several research directions can be considered for the future work:

- The first idea is to embed the resolution of the MILP model in the NSGA-II algorithm to have better solutions.
- A second idea is to provide et test other crossover operator and mutation operator.
- A third idea is analyse which result can be generalized to the case of more than two agents.
- Finally, it mill be interisting to study the case with arbitrary due dates, and consider unrelated parallel machines.

The study carried out has been published in national conferences and international conferences. We summarize below all our publications.

- Van Ut Tran, Ameur Soukhal. *Multi-agent, two-criteria: Optimization total the number of tardy jobs and the makespan on identical parallel processors, iFors 2017, jully 2017, Quebec, Cannada.*
- Van Ut Tran, Ameur Soukhal. *Multi-agent scheduling problem and apply them in application. CanTho Software Park, 01/2017, CanTho, Vietnam.*
- Faiza Sadi, T. Van Ut, Nguyen Huynh Tuong, Ameur Soukhal:*Non-disjoint Multi-agent Scheduling Problem on Identical Parallel Processors. Conference, Future Data and Security Engineering, 10018, Springer, pp.400414, 2016, Lecture Notes in Computer Science, 978-3-319-48057-2, CanTho, Vietnam, November 2016.*
- Van Ut Tran, Ameur Soukhal, Huynh Nguyen. *Minimisation de la date d'achèvement et du nombre de travaux en retard pour l'ordonnancement multiagent non-disjoint. 18ème Conférence ROADEF de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, Feb 2017, Metz, France.*

- Van Ut Tran, Faiza Sadi, Ameur Soukhal. *Minimisation de la date d'achèvement et du nombre de travaux en retard pour lórdonnancement multiagent. 17ème congrès de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF2016), Feb 2016, Compiègne, France.*

- Van Ut Tran, Ameur Soukhal. *Ordonnancement de travaux interférants avec la date d'achèvement et contraint nombre de retards des travaux. 16 ème conférence ROADEF Société Française de Recherche Opérationnelle et Aide à la Décision, Feb 2015, Marseille, France.*

Conclusions et perspectives

Dans cette thèse, nous abordons les problèmes d'ordonnancement multi-agents, avec deux fonctions objectifs. Le premier est de réduire au minimum la durée d'achèvement totale des travaux d'un agent et le second est de réduire au minimum le nombre total de travaux en retard pour une date d'échéance commune. Plus précisément, nous étudions le scénario non disjoint. Seuls deux agents sont pris en compte dans notre étude. Chacun a son propre travail. Il peut arriver que certains travaux soient communs. Les agents partagent les mêmes ressources pour ordonnancer leurs propre travaux. Les ressources considérées dans notre étude sont des machines parallèles identiques.

Ce problème est $\mathcal{NP}$-difficile. Pour calculer le front de Pareto, l'approche ε-contrainte est utiliusée.

Cependant, dans le cas d'une seule machine, nous montrons que le front de Pareto peut être calculé en temps polynomial. Le cas des travaux de même durée opératoire est également étudié. Nous montrons que le problème est équivalent au problème de couplage maximum à coût minimum, qui est polynomial. Le tableau 7.2 résume les cas polynomiaux montrés dans notre étude.

Problem	Complexity	Section
$1\|ND, d^A, C^B_{max} \leq Q_B\|\sum U^A_j$	$O(n_A \log(n_A) + n)$	4.3.1
$1\|ND, d^B, \sum U^B_j \leq Q_B\|C^A_{max}$	$O(n_B \log(n_B) + n \log(UB))$	4.3.2
$1\|ND, d^A\|P(\sum U^A_j, C^B_{max})$	$O(n^2_B \log(n_B) + n_B n \log(UB))$	4.3.3
$Pm\|ND, d^A_j, p_j = p, C^B_{max} \leq Q_B\|\sum U^A_j$	$O(n^{5/2})$	4.4.1
$Pm\|ND, d^B_j, p_j = p, \sum U^B_j \leq Q_B\|C^A_{max}$	$O(n^{5/2} \log(UB))$	4.4.2
$Pm\|ND, d^A_j, p_j = p\|P(\sum U^A_j, C^B_{max})$	$O(n_A n^{5/2} log(UB))$	4.4.3

Table 7.2: New complexity results.

En ce qui concerne le cas général, plusieurs méthodes de résolution ont été mises au point :

- Deux méthodes exactes $MILP1$ et $MILP2$.
- Trois heuristiques gloutonnes : $H1, H2$ et $H3$.
- Deux heuristiques hybrides combinant heuristiques gloutonnes et méthodes exactes (algorithme de programmation dynamique et $MILP$).
- Trois méthodes itératives d'amélioration : Recherche Tabu, $NSGA-II$ et Matheuristique.

Nous avons utilisé différentes métriques pour comparer et analyser les performances de ces méthodes telles que : taille du front de Pareto, distance générationnelle et Hypervolume. L'algorithme gourmand $H3$ donne les meilleurs résultats que les deux autres

heuristiques gloutonnes. La méthode d'hybridation de l'heuristique $H3$ avec $MILP$ est plus performante que l'autre basée sur l'algorithme de programmation dynamique. En ce qui concerne les méthodes itératives d'amélioration, les résultats des expérimentations menées montrent que la meilleure méthode est la Matheuristique.

Plusieurs axes de recherche peuvent être envisagés pour les travaux futurs :

- La première idée est d'intégrer la résolution du modèle MILP dans l'algorithme NSGA-II pour avoir de meilleures solutions.
- Une deuxième idée est de fournir et de tester d'autres opérateurs de croisement et de mutation.
- Une troisième idée est d'analyser et déduire quels résultats peuvent être généralisés au cas de plus de deux agents.
- Enfin, il serait intéressant d'étudier le cas avec des dates d'échéance arbitraires, et d'envisager des machines parallèles non reliées.

Les études menées ont donné lieu à des publications dans des conférences nationales et internationales. Nous résumons ci-dessous toutes nos communications scientifiques.

- Van Ut Tran, Ameur Soukhal. *Multi-agent, two-criteria: Optimization total the number of tardy jobs and the makespan on identical parallel processors, iFors 2017, jully 2017, Quebec, Cannada.*
- Van Ut Tran, Ameur Soukhal. *Multi-agent scheduling problem and apply them in application. CanTho Software Park, 01/2017, CanTho, Vietnam.*
- Faiza Sadi, T. Van Ut, Nguyen Huynh Tuong, Ameur Soukhal:*Non-disjoint Multi-agent Scheduling Problem on Identical Parallel Processors. Conference, Future Data and Security Engineering, 10018, Springer, pp.400414, 2016, Lecture Notes in Computer Science, 978-3-319-48057-2, CanTho, Vietnam, November 2016.*
- Van Ut Tran, Ameur Soukhal, Huynh Nguyen. *Minimisation de la date d'achèvement et du nombre de travaux en retard pour l'ordonnancement multiagent non-disjoint. 18ème Conférence ROADEF de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, Feb 2017, Metz, France.*
- Van Ut Tran, Faiza Sadi, Ameur Soukhal. *Minimisation de la date d'achèvement et du nombre de travaux en retard pour lórdonnancement multiagent. 17ème congrès de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF2016), Feb 2016, Compiègne, France.*
- Van Ut Tran, Ameur Soukhal. *Ordonnancement de travaux interférants avec la date d'achèvement et contraint nombre de retards des travaux. 16 ème conférence ROADEF Société Française de Recherche Opérationnelle et Aide à la Décision, Feb 2015, Marseille, France.*

Bibliography

[Agnetis *et al.*, 2014] Agnetis, A., Billaut, J.-C., Gawiejnowicz, S., Pacciarelli, D. et Soukhal, A. (2014). *Multiagent Scheduling, Models and Algorithms*. Springer. 1, 1, 2.3.4, 2, 2.4.2.2, 3.3.3, 3.3.4.2, 3.4

[Agnetis *et al.*, 2009a] Agnetis, A., de Pascale, G. et M.Pranzo (2009a). Computing the nash solution for scheduling bargaining problems. *International Journal of Operational Research*, 1:54–69. 3.3, 3.4

[Agnetis *et al.*, 2000] Agnetis, A., Mirchandani, P., Pacciarelli, D. et Pacifici, A. (2000). Nondominated Schedules for a Job-Shop with Two Competing Users. *Computational & Mathematical Organization Theory*, 6(2):191–217. 3.3, 3.3.1.1

[Agnetis *et al.*, 2004] Agnetis, A., Mirchandani, P., Pacciarelli, D. et Pacifici, A. (2004). Scheduling problems with two competing agents. *Operations Research*, 52:229–242. 3.3, 3.3.1.1, 3.4

[Agnetis *et al.*, 2009b] Agnetis, A., Pacciarelli, D. et de Pascale, G. (2009b). A Lagrangian approach to single-machine scheduling problems with two competing agents. *Journal of Scheduling*, 12:401–415. 3.3

[Agnetis *et al.*, 2007] Agnetis, A., Pacciarelli, D. et Pacifici, A. (2007). Multi-agent single machine scheduling. *Annals of Operations Research*, 150:3–15. 3.3, 3.4

[Alharkan, 1997] Alharkan, I. M. (1997). *Algorithms for Sequencing and Scheduling*. Industrial Engineering Department College of Engineering King Saud University Riyadh, Saudi Arabia. 3.1.3.3

[Baker et Smith, 2003] Baker, K. et Smith, J. (2003). A multiple-criterion model for machine scheduling. *Journal of Scheduling*, 6:7–16. 2.2, 3.4

[Balasubramanian *et al.*, 2009a] Balasubramanian, H., Fowler, J., A.Keha et Pfund, M. (2009a). Scheduling interfering job sets on parallel machines. *European Journal of Operational Research*, 199:55–67. 3.4

[Balasubramanian *et al.*, 2009b] Balasubramanian, H., Fowler, J., Keha, A. et Pfund, M. (2009b). Scheduling interfering job sets on parallel machines. *European Journal of Operational Research*, 199(1):55–67. 4.5

[Baptiste et Brucker, 2004] Baptiste, P. et Brucker, P. (2004). *Scheduling equal processing time jobs, in : J.Y. Leung (Ed.), Handbook of Scheduling : Algorithms, Models and Performance Analysis.* CRC Press, Boca Raton, FL, USA. 3.3

[Bellman et Kalaba, 1957] Bellman, R. et Kalaba, R. (1957). On the role of dynamic programming in statistical communication theory. *IRE Trans. Information Theory*, 3(3): 197–203. 2.4.2.2

[Beume *et al.*, 2009] Beume, N., Fonseca, C., Lopez-Ibanez, M., Paquete, L. et Vahrenhold, J. (2009). On the Complexity of Computing the Hypervolume Indicator. *IEEE Transactions on Evolutionary Computation*, 13(5):1075–1082. 2.5.1

[Beyer et Deb, 2001] Beyer, H.-G. et Deb, K. (2001). On self-adaptive features in real-parameter evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 5(3):250–270. 2.4.4.5

[Blazewicz *et al.*, 2007] Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G. et Weglarz, J. (2007). *Handbook on scheduling. From theory to applications.* Springer-Verlag, Berlin Heidelberg. 2.1, 3.1

[Bowman et Edward, 1959] Bowman et Edward, H. (1959). The Schedule-Sequencing Problem. *Operations Research*, 7(5):621–624. 2.4.2.1

[Brucker, 2007] Brucker, P. (2007). *Scheduling Algorithms.* Springer-Verlag, Berlin, 5th édition. 2.1, 3.1, 3.1.2

[Carlier et Chrétienne, 1988] Carlier, J. et Chrétienne, P. (1988). *ProblÃ¨mes d'ordonnancement : modÃ©lisation, complexitÃ©, algorithmes.* Collection Gestion, Masson, Paris. 2.1, 3.1

[Cheng *et al.*, 2008] Cheng, T. C. E., Ng, C. et Yuan, J. J. (2008). Multi-agent scheduling on a single machine with max-form criteria. *European Journal of Operational Research*, 188:603–609. 3.4

[Chibante, 2010] Chibante, R. (2010). *Simulated Annealing Theory with Applications.* India. 2.4.4.2

[Cordeiro *et al.*, 2011] Cordeiro, D., Dutot, P., Mounié, G. et Trystram, D. (2011). Tight analysis of relaxed multi-organization scheduling algorithms. *In 25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May, 2011 - Conference Proceedings*, pages 1177–1186. 3.1, 3.4

[Cyzak et Jaszkiewicz, 1998] Cyzak, P. et Jaszkiewicz, A. (1998). Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multicriteria Decision Analysis*, 7:34–47. 2.5.2

[Dauzère-Pérès et Pavageau, 2003] Dauzère-Pérès, S. et Pavageau, C. (2003). Extensions of an integrated approach for multi-resource shop scheduling. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 33(2):207–213. 3.3

[Dauzère-Pérès et Sevaux, 2003] Dauzère-Pérès, S. et Sevaux, M. (2003). Using lagrangean relaxation to minimize the weighted number of late jobs on a single machine. *Naval Research Logistics*, 50:273–288. 3.3

[Deb *et al.*, 2002] Deb, K., Anand, A. et Joshi, D. (2002). A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation*, 10(4):345–369. 6.3

[Dhib *et al.*, 2011] Dhib, C., Kooli, A., Soukhal, A. et Néron, E. (2011). Lower bounds for a multi-skill project scheduling problem. *In Operations Research Proceedings 2011, Selected Papers of the International Conference on Operations Research (OR 2011), August 30 - September 2, 2011, Zurich, Switzerland*, pages 471–476. 3.3.4.3

[Dileepan et Sen, 1988] Dileepan, P. et Sen, T. (1988). Bicriterion static scheduling research for a single machine. *Omega*, 16(1):53–59. 3.2

[Dréo *et al.*, 2006] Dréo, J., Pétrowski, A. et Taillard, E. (2006). *Metaheuristics for Hard Optimization*. Springer Berlin Heidelberg New York. 6.1

[Du et Leung, 1990] Du, J. et Leung, J. (1990). Minimizing total tardiness on one machine is np-hard. *Mathematics of operations research*, 15:483–495. 3.1

[Elvikis *et al.*, 2011] Elvikis, D., Hamacher, H. et T'Kindt, V. (2011). Scheduling two agents on uniform parallel machines with makespan and cost functions. *Journal of Scheduling*, 14:471–481. 3.4

[Garey *et al.*, 1976] Garey, M., Johnson, D. et Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129. 3.1.2, 3.1.3.2

[Geoffrion, 1968] Geoffrion, A. M. (1968). Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22(3):618–630. 2.3.1

[Glover, 1989] Glover, F. (1989). Tabu search - part I. *INFORMS Journal on Computing*, 1(3):190–206. 2.4.4.3, 6.2

[Glover, 1990] Glover, F. (1990). Tabu search - part II. *INFORMS Journal on Computing*, 2(1):4–32. 2.4.4.3, 6.2

[Glover et Laguna, 1997] Glover, F. et Laguna, M. (1997). *Tabu search*. Kluwer Academic Publishers. 2.4.4.3

[Gourgand *et al.*, 2007] Gourgand, M., Grangeon, N. et Norre, S. (2007). Metaheuristics based on bin packing for the line balancing problem. *RAIRO - Operations Research*, 41(2):193–211. 6.2

[Gourgand *et al.*, 2010] Gourgand, M., Lemoine, D. et Norre, S. (2010). Metaheuristic for the capacitated lot-sizing problem: a software tool for MPS elaboration. *IJMOR*, 2(6):724–747. 6.2

[Graham *et al.*, 1979] Graham, R. L., Lawler, E. L., Lenstra, J. K. et Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326. 3.1.1

[Holland, 1992] Holland, J. H. (1992). Adaptation in natural and artificial systems. *MIT Press.* 2.4.4.4, 2.4.4.4, 6.3

[Hoogeveen, 2005] Hoogeveen, H. (2005). Multicriteria scheduling. *European Journal of Operational Research*, 167:592–623. 3.2, 3.3

[Hoogeveen, 1992] Hoogeveen, J. (1992). Single-machine multi criteria scheduling. *Phd thesis, Technische Universiteit Eindhiven, The Netherlands.* 3.3.3

[Hoogeveen, 1996] Hoogeveen, J. (1996). Single-machine scheduling to minimize a function of two or three maximum cost criteria. *Journal of Algorithms*, 21:415–433. 3.2

[Hopcroft et Karp, 1973] Hopcroft, J. E. et Karp, R. M. (1973). An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 4:225–231. 4.4.1, 4.4.2

[Huynh Tuong et Soukhal, 2009] Huynh Tuong, N. et Soukhal, A. (2009). Interfering job set scheduling on two-operation three-machine flowshop. *In International Conference on Computing and Communication Technologies, 2009*, pages 1–5, IEEE press. 3.4

[Huynh Tuong *et al.*, 2011] Huynh Tuong, N., Soukhal, A. et Billaut, J.-C. (2011). Single-machine multi-agent scheduling problems with a global objective function. *Journal of Scheduling*, 15(3):311–321. 3.1.3.1, 3.4

[Huynh Tuong *et al.*, 2012] Huynh Tuong, N., Soukhal, A. et Billaut, J.-C. (2012). Single-machine multi-agent scheduling problems with a global objective function. *Journal of Scheduling*, 15:311–321. 3.4

[Jarboui *et al.*, 2013] Jarboui, B., Siarry, P. et Teghem, J. (2013). Métaheuristiques pour lòrdonnancement multicritère et les problèmes de transport. *Management science research*, pages 1–323. 2.5.1

[Jozefowska, 2007] Jozefowska, J. (2007). *Just-in-Time Scheduling. Models and algorithms for computer and manufacturing systems.* Springer-Verlag, Berlin. 2.1, 3.2, 3.3

[Kergosien *et al.*, 2017] Kergosien, Y., Gendreau, M. et Billaut, J.-C. (2017). A benders decomposition-based heuristic for a production and outbound distribution scheduling problem with strict delivery constraints. *European Journal of Operational Research*, 262(1):287–298. 2.4.5

[Kergosien *et al.*, 2011] Kergosien, Y., Lente, C., Piton, D. et Billaut, J.-C. (2011). A tabu search heuristic for the dynamic transportation of patients between care units. *European Journal of Operational Research*, 214(2):442–452. 6.2

[Ketan et Balasubramanian, 2014] Ketan, K., F. J. K.-A. et Balasubramanian (2014). Single machine scheduling with interfering job sets. *Computers and Operations Research*, 45(0):97–107. 3.4

[Kirkpatrick *et al.*, 1983] Kirkpatrick, S., Gelatt, C. D. et Vecchi, M. P. (1983). Optimization by Simulated Annealing. *New Science*, 220:671–680. 2.4.4.2

[Kovalyov *et al.*, 2012] Kovalyov, M., Oulamara, A. et Soukhal, A. (2012). Two-agent scheduling with agent specific batches on an unbounded serial batching machine. *In The 2nd International Symposium on Combinatorial Optimization, ISCO 2012, LNCS 7422, Athens.* 3.3

[Kovalyov *et al.*, 2015] Kovalyov, M. Y., Oulamara, A. et Soukhal, A. (2015). Two-agent scheduling with agent specific batches on an unbounded serial batching machine. *J. Scheduling*, 18(4):423–434. 3.3

[Laurent *et al.*, 2017] Laurent, A., Deroussi, L., Grangeon, N. et Norre, S. (2017). A new extension of the RCPSP in a multi-site context: Mathematical model and metaheuristics. *Computers & Industrial Engineering*, 112:634–644. 6.2

[Lawler, 1973] Lawler, E. (1973). Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(8):544–546. 3.2

[Lee et Kim, 1999] Lee, D.-H. et Kim, Y.-D. (1999). Scheduling algorithms for flexible manufacturing systems with partially grouped machines. *Journal of Manufacturing Systems*, 18(4):301–309. 3.1

[Lee *et al.*, 2009] Lee, K., Choi, B.-C., Leung, J. Y.-T. et Pinedo, M. (2009). Approximation algorithms for multi-agent scheduling to minimize total weighted completion time. *Information Processing Letters*, 109:913–917. 3.4

[Lee *et al.*, 2013] Lee, W.-C., Chung, Y.-H. et Huang, Z.-R. (2013). A single-machine bi-criterion scheduling problem with two agents. *Applied Mathematics and Computation*, 219(23):10831–10841. 3.4

[Lee et Wang, 2014] Lee, W.-C. et Wang, J.-Y. (2014). A scheduling problem with three competing agents. *Computers and Operations Research*, 51:208–217. 3.4

[Leung *et al.*, 2010] Leung, J. Y.-T., Pinedo, M. et Wan, G. (2010). Competitive two-agent scheduling and its applications. *operations Research*, 58:458–469. 3.4

[Lopez et Roubellat, 2008] Lopez, P. et Roubellat, F. (2008). *Production Scheduling.* Wiley-ISTE. 3.1

[Marco et Voss, 2010] Marco, C. et Voss, S. (2010). *Matheuristics*, volume 10 de *Annals of Information Systems.* Springer US, MA Boston. 2.4.5

[Metropolis, 1953] Metropolis (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092. 2.4.4.2

[Moore, 1968] Moore, J. M. (1968). An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15:102–109. 1

[Morton et Pentico, 1993] Morton, T. E. et Pentico, D. W. (1993). *Heuristic scheduling systems.* A Wiley Interscience. 1, 1, 3.1

[Nagar *et al.*, 1995] Nagar, A., Haddock, J. et Heragu, S. (1995). Multiple and bicriteria scheduling : A literature survey. *European Journal of Operational Research*, 81(1):88–104. 3.2

[Navet, 2006] Navet, N. (2006). Systï¿½mes temps rï¿½el 2 : Ordonnancement, rï¿½seaux et qualitï¿½ de service (traitï¿½ ic2, sï¿½rie informatique et systï¿½mes d'information). *Hermes Science Publications.* 3.1

[Ng *et al.*, 2006a] Ng, C., Cheng, T. et Yuan, J. (2006a). A note on the complexity of the problem of twoagent scheduling on a single machine. *Journal of Combinatorial Optimization*, 12(4):387–394. 3.3, 3.4

[Ng *et al.*, 2006b] Ng, C., Cheng, T. C. E. et Yuan, J. J. (2006b). A note on the complexity of the problem of two-agent scheduling on a single machine. *Journal of Combinatorial Optimization*, 12:387–394. 3.4

[Norre, 1993] Norre, S. (1993). Static allocation of tasks on multiprocessor architectures with interprocessor communication delays. *In PARLE '93, Parallel Architectures and Languages Europe, 5th International PARLE Conference, Munich, Germany, June 14-17, 1993, Proceedings*, pages 488–499. 3.1

[Oulamara *et al.*, 2009] Oulamara, A., Finke, G. et Kuiten, A. K. (2009). Flowshop scheduling problem with batching machine and task compatibilities. *Computers & Operations Research*, 36:391–401. 3.1

[Oulamara *et al.*, 2005] Oulamara, A., Kovalyov, M. et Finke, G. (2005). Scheduling a no-wait flowshop with unbounded batching machines. *IIE Transactions on Scheduling and logistics*, 37:685–696. 3.1

[Papadimitriou et Harilaos, 1976] Papadimitriou et Harilaos, C. (1976). *The complexity of combinatorial optimization problems.* Princeton University. 2.4.4.1, 2.4.4.1

[Papadimitriou et Steiglitz, 1982] Papadimitriou, C. H. et Steiglitz, K. (1982). *Combinatorial optimization : algorithms and complexity.* Prentice Hall. 2.4.4.1, 2.4.4.1

[Papadimitriou Christos H. , 1994] Papadimitriou Christos H. (1994). *Computational Complexity.* Addison Wesley Publishing Company. 3.1.2

[Pareto, 1897] Pareto, V. (1897). The new theories of economics. *The University of Chicago Press*, 5:485–502. 2.2.1

[Peha, 1995] Peha, J. (1995). Heterogeneous-criteria scheduling: Minimizing weighted number of tardy jobs and weighted completion time. *Journal of Computers and Operations Research*, 22(10):1089–1100. 3.1, 3.4

[Peha et Tobagi, 1990] Peha, J. et Tobagi, F. (1990). Evaluating scheduling algorithms for traffic with heterogeneous performance objectives. *In [Proceedings] GLOBECOM '90: IEEE Global Telecommunications Conference and Exhibition*, pages 21–27. IEEE. 3.3.4.3

[Pinedo, 2008] Pinedo, M. (2008). *Scheduling. Theory, algorithms, and systems.* Springer-Verlag, Berlin, 3rd édition. 2.1, 3.1

[Pinedo, 2016] Pinedo, M. L. (2016). *Scheduling Theory, Algorithms, and Systems.* Springer Cham Heidelberg New York Dordrecht London, 5th édition. 3.1

[Posner, 1985] Posner, M. E. (1985). Minimizing weighted completion times with deadlines. *Operations Research*, 33(3):562–574. 3.4

[Rios-Mercado et Rios-Solis, 2012] Rios-Mercado, R. Z. et Rios-Solis, Y. A. (2012). *Just-in-Time Systems.* Springer-Verlag, London. 2.1, 3.2

[Sadi et Soukhal, 2017] Sadi, F. et Soukhal, A. (2017). Complexity analyses for multi-agent scheduling problems with a global agent and equal length jobs. *Discrete Optimization*, 23:93–104. 3.4

[Sadi *et al.*, 2014] Sadi, F., Soukhal, A. et Billaut., J.-C. (2014). Solving multi-agent scheduling problems on parallel machines with a global objective function. *RAIRO - Operations Research*, 48(2):255–269. 2.4.2.2, 3.4

[Sadi *et al.*, 2015] Sadi, F., Van Ut, T., Tuong, N. H. et Soukhal, A. (2015). Non-disjoint Multi-agent Scheduling Problem on Identical Parallel Processors. *In Future Data and Security Engineering*, numéro FDSE 2016, chapitre Emerging D, pages 400–414. Springer, Cham. 3.1

[Santiago *et al.*, 2014] Santiago, A., Huacuja, H. J. F., Dorronsoro, B., Pecero, J. E., Santillan, C. G., Barbosa, J. J. G. et Monterrubio, J. C. S. (2014). A Survey of Decomposition Methods for Multi-objective Optimization. pages 453–465. Springer. 2.2.1

[Smith, 1956] Smith, W. E. (1956). Various optimizer for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66. 3.2

[Soukhal, 2012] Soukhal, A. (2012). *Modèles et Algorithmes pour l'Ordonnancement des Travaux Indépendants et Concurrents.* Habilitation à diriger des recherches, Université François-Rabelais de Tours, Tours. 2.4.2.2

[Srinivas et Deb, 1994] Srinivas, N. et Deb, K. (1994). Multiobjective function optimization using nondominated sorting genetic algorithms. *Journal of association for computing machinery*, pages 221–248. 2.4.4.5

[Talbi, 2013] Talbi, E.-G. (2013). *Hybrid Metaheuristics*, volume 434 de *Studies in Computational Intelligence.* Springer Berlin Heidelberg, Berlin, Heidelberg. 2.4.5

[T'Kindt et Billaut, 2006] T'Kindt, V. et Billaut, J.-C. (2006). *Multicriteria scheduling. Theory, models and algorithms.* Springer-Verlag, Berlin Heildelberg New York, 2nd édition. 2.1, 2.2.2, 2.3, 3.1.2, 3.2, 3.2.1, 3.3, 3.4

[Venditti *et al.*, 2010] Venditti, L., Pacciarelli, D. et Meloni, C. (2010). A tabu search algorithm for scheduling pharmaceutical packaging operations. *European Journal of Operational Research*, 202(2):538–546. 6.2

[Wagner, 1959] Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140. 2.4.2.1

[Wan *et al.*, 2010] Wan, G., Leung, J.-Y. et Pinedo, M. (2010). Scheduling two agents with controllable processing times. *European Journal of Operational Research*, 205:528–539. 3.4

[Wu *et al.*, 2013] Wu, C. C., W.-H. Wu andChen, J.-C. Y., Y. et Wu, W.-H. (2013). A study of the singlemachine two-agent scheduling problem with release times. *Applied Soft Computing*, 13(2):998–1006. 3.4

[Yuan *et al.*, 2005] Yuan, J. J., Shang, W. et Feng, Q. (2005). A note on the scheduling with two families of jobs. *Journal of Scheduling*, 8:537–542. 3.4

[Zitzler et Thiele, 1998] Zitzler, E. et Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms: A comparative case study. *In Springer, Berlin, Heidelberg*, pages 292–301. Springer, Berlin, Heidelberg. 2.5.1

Résumé :

Nous étudions des « problèmes d'ordonnancement multiagent non-disjoint ». Ces modèles considèrent différents agents associés à des sous-ensembles de travaux non nécessairement disjoints, chacun d'eux vise à minimiser un objectif qui ne dépend que de ses propres travaux. Deux types de critères sont considérés : minimisation du makespan et du nombre de travaux en retard. Nous cherchons donc les meilleurs compromis entre les critères des agents. Ces problèmes sont une classe particulière des problèmes d'ordonnancement « multi-agents » qui ont connu une grande expansion par leurs intérêts dans le domaine de l'ordonnancement et l'optimisation combinatoire. Dans nos travaux, nous nous sommes intéressés aux problèmes à machines parallèles identiques. Dans une première partie, nous étudions le cas d'une seule machine et celui où les travaux ont des durées identiques. Ainsi, des problèmes polynomiaux sont identifiés. Dans une seconde partie de nos travaux, nous abordons le problème à machines parallèles identiques avec deux agents. Notre étude porte sur l'énumération du front de Pareto par l'approche ε-contrainte en utilisant des modèles mathématiques à variables mixtes. Les résultats des expérimentations montrent les limites de ces méthodes. Pour résoudre des problèmes de grande taille, des heuristiques gloutonnes, hybrides, des métaheuristiques ou encore une matheuristique sont développées. Des expérimentations sont menées afin de montrer leurs performances par rapport aux méthodes exactes ou par rapport à la borne inférieure proposée.

Mots clés :

Recherche Opérationnelle, Ordonnancement multi-agent, Machines parallèles, Fronts de Pareto, Programmation mathématique, Metaheuristiques, Matheuristique.

Abstract :

We are studying "non-disjoint multi-agent scheduling problems". These models consider different agents associated with non-disjoint subsets of jobs, each of them aims to minimize an objective function that depends only on its own jobs. Two types of criteria are considered: minimization of the makespan and minimization of the number of late jobs. We are therefore looking for the best compromise solution between the agents. These problems constitute particular class of "multi-agent' scheduling problems that have developed considerably due to their interests in scheduling and combinatorial optimization domains. In our work, we focused on problems with identical parallel machines. In a first part, we study the case of a single machine, then the case where the jobs are equal length. Thus, polynomial scheduling problems are identified. In a second part, we address the identical parallel machine scheduling problems with two agents. Our study focuses on the enumeration of the Pareto front by the ε-constraint approach using mathematical mixed integer programming. The computational results show the limitations of these methods. To solve large instances, greedy heuristics, hybrid heuristics, metaheuristic or even matheuristic are developed. The computational results show their performance compared to exact methods or to the proposed lower bound.

Keywords :

Operational research, Multi-agent scheduling, Parallel machines, Pareto fronts, Mathematical programming, Metaheuristics, Matheuristic.