# Polynomial Approximation Algorithms for Parallel Machine Problems in a Multi-objective or Constrained Context

Gais Alhadi Babikir Alhadi

# Polynomial Approximation Algorithms for Parallel Machine Problems in a Multi-objective or Constrained Context


**By: Gais ALHADI BABIKIR ALHADI**

**Supervisors:** Prof. Imed KACEM (Université de Lorraine, France)
Dr. Pierre LAROCHE (Université de Lorraine, France)
Prof. Izzeldin M. OSMAN (University of Gezira, Sudan)

Submitted to the IAEM Lorraine Doctoral School in Partial Fulfillment of the Requirements for the Degree of

**Doctor in Computer Science**

at the

**UNIVERSITÉ DE LORRAINE**

(The Jury Members):

Prof. Feng CHU (Rapporteur), Dr. Marie-Ange MANIER (Rapporteur), Prof. Mhand HIFI (Examinateur), Prof. Frédéric SAUBION (Examinateur), Dr. Awadallah M. AHMED (Examinateur), Prof. Imed KACEM (Supervisor), Prof. Izzeldin M. OSMAN (Supervisor), Dr. Pierre LAROCHE (Co-Supervisor).

July 2019

I would like to dedicate this thesis to my parents for their love, prayers, and support.
To my wife Sara and my daughter Lamees, thanks for being always beside me.

# Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

By: Gais ALHADI BABIKIR ALHADI
July 2019

# Acknowledgements

Foremost, I would like to express my sincere gratitude to my Supervisor at the University of Lorraine Prof. Imed Kacem for the continuous support of my Ph.D. study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

Besides my Supervisor, I would like to thank my Co-Supervisor Dr. Pierre Laroche, who was very generous with his time and knowledge, for his patience, motivation, enthusiasm, encouragement, insightful comments and immense knowledge. His guidance helped me in each step of research to complete this thesis.I can't thank you enough for all that you have done for me.

I would also like to extend my heartfelt thanks to my Supervisor at the University of Gezira Prof. Izzeldin M.Osman, for his continuous advice and support.His support helped me in all the time of research.

**Thank you, my Supervisors, It has had the pleasure and honor of working with you.**

I also must thank the thesis reporters and all the jury members for the thorough and critical judgment and evaluation of this thesis.

I also express my gratitude to the staff members and friends from the IUT-Metz for their help and support, especially, Chrystele, Sebastien, Nicolas, and Mireille.

I never forget the University of Gezira for providing me with this scholarship, especially, Prof. Mohammed W. Omer, Prof. Mohammed El-Sanosy and Dr. Murtada K. El-Bashir.

I greatly appreciate all kinds of assistance received from the Embassy of France in Khartoum, especially Michel Roy.

I deeply thank my wife and my sweetheart Sara, who was by my side at every moment and carried all the difficulties with me.

Finally and forever, I acknowledge the moral and emotional support I received from my family and friends during this challenging period and my life in general.

# Abstract

This thesis addresses the scheduling problems on parallel machines, with and without non-availability constraints, and the design of approximation methods dedicated to solving this problem in a multi-objective context (load balancing and minimization of the delivery times). These are critical logistical issues for the quality of service and the performance of such systems. These are NP-hard optimization problems. In this context, the carried work of this thesis leads to a contribution to solving and approximating the performance of these systems with parallel resources. Many optimization methods have been developed and tested in this context. These methods include different approaches such as heuristics of guaranteed performance, dynamic programming algorithms, polynomial time approximation scheme (PTAS), and fully polynomial time approximation scheme (FPTAS).

In particular, we thoroughly analyzed the basic substructure with two parallel machines. We have studied the scenario, with a constraint of unavailability on a machine, associated with this substructure. We have shown that the problem has a constant polynomial approximation algorithm. Thus, we presented a dynamic programming algorithm and an FPTAS, which has a strongly polynomial running time. Experimental tests have been performed and used to compare the performances of the proposed algorithms on several sets of instances.

The second important contribution of this thesis is related to the determination of the Pareto solutions for the same substructure (two parallel machines) but without non-availability constraint. Many methods have been proposed in this section: dynamic programming, PTAS, FPTAS in two versions, with detailed experimental comparisons.

The third contribution of the thesis concerns the extension of the multi-objective study to the problem of scheduling jobs on $m$ parallel machines. Different extensions and algorithms have been proposed: a dynamic programming, a PTAS and an FPTAS for fixed value of $m$. Experimental tests were conducted and allowed to evaluate and compare the performance of these methods.

# Table of contents

# List of figures

# List of tables

# Résumé en français

Cette thèse traite du problème d'ordonnancement sur machines parallèles, avec et sans indisponibilités, et de la conception de méthodes d'approximation dédiées à la résolution de ce problème dans un contexte multi-objectif (équilibrage des charges et minimisation de la date de livraison). Ce sont des problèmes cruciaux dans le domaine logistique pour améliorer la qualité de service et la performance de tels systèmes. Il s'agit des problèmes d'optimisation NP-difficiles. Dans cette optique, les travaux de cette thèse constituent une contribution à la résolution exacte et approchée de ces systèmes à ressources parallèles. Plusieurs méthodes d'optimisation ont été développées et testées dans ce cadre. De telles méthodes incorporent différentes approches de résolution ou d'optimisation, comme les heuristiques à garantie de performance, les algorithmes de programmation dynamique, les schémas d'approximation polynomiaux (PTAS) et entièrement polynomiaux (FPTAS).

En particulier, nous avons analysé de manière approfondie la sous-structure de base avec deux machines parallèles. Nous avons étudié le scénario associé à cette sous-structure dans lequel une contrainte d'indisponibilité est présente sur une machine, et nous avons montré que le problème admet une programmation dynamique de complexité pseudo-polynomiale et un FPTAS fortement polynomial. Des tests expérimentaux ont été effectués et ont permis de comparer les performances pratiques des algorithmes proposés sur plusieurs jeux d'instances (*Alhadi, Kacem, Laroche, Osman*, 2018, *IEEE/CoDIT*'18).

La deuxième contribution importante de cette thèse concerne la détermination des solutions dominantes au sens de Pareto pour la même sous-structure (deux machines parallèles) mais sans indisponibilité. Les méthodes proposées dans cette partie sont nombreuses : programmation dynamique, PTAS, FPTAS en deux versions, avec des comparaisons expérimentales détaillées (*Alhadi, Kacem, Laroche, Osman*, 2019, *Annals of Operations Research*).

La troisième contribution de la thèse porte sur l'extension de l'étude multi-objectif au problème d'ordonnancement sur $m$ machines parallèles. Différentes extensions et algorithmes ont été proposés : la programmation dynamique, un PTAS et un FPTAS pour $m$ fixé. Des tests expérimentaux ont été réalisés et ont permis de bien évaluer et comparer les performances de ces méthodes.

# Résumé du chapitre 1: Problèmes d'ordonnancement et concepts d'approximation

Dans ce premier chapitre, nous présentons les principales notions sur lesquelles repose cette thèse. Les problèmes d'ordonnancement sont tout d'abord présentés, dans leur grande diversité. Les différentes méthodes de résolution sont passées en revue, et nous détaillons également les problèmes d'ordonnancement sur machines parallèles. Des notions de base concernant la complexité des algorithmes sont ensuites données, puis nous nous concentrons sur la théorie d'approximation. Les classes d'approximation sont rappelées, puis nous détaillons les différentes approches possibles (PTAS, FPTAS), en indiquant les techniques généralement mises en œuvre dans ce type de méthodes. Le chapitre est ensuite conclu par une étude bibliographique de résolution de problèmes d'ordonnancement, en considérant les problèmes sur machines parallèles.

# Résumé du chapitre 2 : Optimisation multi-objectif

Ce deuxième chapitre est dédié à la présentation de l'étude des problèmes multiobjectif, qui est un des domaines les plus difficiles auxquels les chercheurs ont été confrontés depuis les années 1950. Ce chapitre présente toutes les notions liées à l'optimisation combinatoire multi-objectifs. Par exemple, nous illustrons le concept de solutions dominées et non dominées, l'optimalité de Pareto et l'approximation des solutions Pareto-optimales. La suite de ce chapitre est consacrée à la présentation des techniques importantes pouvant être utilisées pour résoudre des problèmes d'optimisation multi-objectifs. Nous présentons également les principaux problèmes de conception des métaheuristiques multi-objectifs. Enfin, ce chapitre se termine par les problèmes concrets liés à l'optimisation combinatoire multi-objectifs.

# Résumé du chapitre 3 : Minimisation de la latence maximale pour deux machines parallèles en présence d'un intervalle d'indisponibilité

Dans ce chapitre, nous traitons le problème d'ordonnancement de $n$ tâches sur deux machines parallèles identiques, dont l'une est affectée par un intervalle d'indisponibilité : à partir d'un temps $T_1$ fixé, la machine n'est plus disponible, aucune tâche de peut plus lui être affectée. Le problème auquel nous nous intéressons est celui de la minimisation de la date de livraison

maximale. Chaque tâche $j$ possède un temps de traitement $p_j$ et un temps de latence $q_j$ : une fois la tâche $j$ traitée sur une machine, il faut attendre le délai $q_j$ avant de pouvoir disposer de la tâche livrée. Ce temps de latence peut être induit par différents facteurs que l'on peut rencontrer dans les applications de l'ordonnancement :

- le temps de refroidissement d'une pièce après son usinage ;
- le temps de séchage après une opération de peinture ;
- le temps de livraison du produit par les services postaux ;
- etc.

La latence maximale de l'ordonnancement correspond à l'instant auquel la dernière tâche devient effectivement disponible, en additionnant sa date de fin d'exécution et son temps de latence : $max_j C_j + q_j$.

Nous présentons la résolution de ce problème d'ordonnancement de façon exacte, puis en introduisant un schéma d'approximation. Une étude bibliographique de l'application des schémas d'approximation à des problèmes similaires est présentée.

L'algorithme exact que nous proposons est un algorithme de programmation dynamique. Les tâches sont ordonnées en fonction de la règle de Jackson : par ordre de temps de latence décroissant. L'ensemble des ordonnancements possibles est ensuite construit de façon itérative, en plaçant successivement chaque tâche sur les deux machines, jusqu'à atteindre la date d'indisponinilité de la seconde machine. Cet algorithme de compléxité $O(2^n)$ est ramené à une complexité polynomiale $O(n \cdot T_1)$ en s'appuyant sur certaines caractéristiques des solutions possibles.

Pour résoudre notre problème de façon approchée, un schéma d'approximation est ensuite proposé : la décomposition de l'espace des solutions permet la mise en oeuvre d'un FPTAS (*Fully Polynomial Time Approximation Scheme*). Pour un $\varepsilon$ fixé, une FPTAS garantit de ne jamais s'éloigner de l'optimum de plus d'un facteur $(1+\varepsilon)$ en une complexité polynomiale dépendant à la fois de la taille de l'instance et de $1/\varepsilon$. Nous présentons la méthode de décomposition utilisée, puis le nouvel algorithme permettant de trouver une solution approchée en $O(n^2/\varepsilon^2)$. Cette complexité et la garantie de performance sont démontrées dans une preuve, avant qu'un exemple d'illustration soit présenté.

Ce chapitre se conclut par la présentation de résultats expérimentaux, en utilisant des instances générées aléatoirement. Ces instances contiennent jusqu'à 1000 tâches à ordonnancer, et les temps de traitement et de latence varient selon trois intervalles différents. Cela nous permet d'étudier l'influence de ces trois facteurs sur la qualité et les temps de calcul des

résultats obtenus. Nous faisons également varier les périodes d'indisponibilité et le facteur $\varepsilon$ afin de compléter notre étude.

Les résultats présentés sont de qualité très satisfaisante, que ce soit en terme de temps de calcul ou en terme de qualité des solutions obtenues.

## Résumé du chapitre 4 : Algorithmes d'approximation pour minimiser la date de livraison maximale et le *makespan* sur deux machines parallèles

Dans ce chapitre, nous traitons un problème d'ordonnancement multiobjectif, dans le but de minimiser à la fois la date de livraison maximale et la durée totale de l'ordonnancement (*makespan*). Cette durée totale correspond à la date de fin d'ordonnacement de la dernière tâche. Comme au chapitre précédent, les tâches $j$ possèdent un temps de traitement $p_j$ et un temps de latence $q_j$. En revanche, contrairement au chapitre précédent, aucune indisponibilité n'est à gérer sur les machines.

Nous présentons la résolution de ce problème d'ordonnancement de façon exacte, puis en introduisant plusieurs schémas d'approximation. Une étude bibliographique de l'application des schémas d'approximation à des problèmes multiobjectif similaires est présentée ; le problème étudié ici ne l'a jamais été auparavent.

Ici aussi, notre algorithme exact est un algorithme de programmation dynamique. Le front de Pareto est obtenu itérativement, en construisant l'ensemble des solutions possibles. Les tâches, ordonnées selon la règle de Jackson, sont successivement affectées sur chaque machine. En fin d'algorithme, seules les solutions non dominées sont conservées afin d'obtenir le front de Pareto. Cet algorithme possède une complexité de $O(2^n)$. Néanmoins, à chaque étape, certaines solutions partielles sous-optimales peuvent être éliminées (sans perdre le caractère exact de l'algorithme), afin de réduire la complexité à $O(n \cdot P)$, $P$ étant la somme des temps de traitement des différentes tâches à ordonnancer.

La recherche de solutions approchées est ensuite abordée avec l'introduction d'un premier schéma d'approximation de type PTAS (*Polynomial Time Approximation Scheme*). Dans ce type de schémas, l'instance initiale est transformée en une instance plus facile à résoudre, puis la solution obtenue est appliquée à l'instance initiale. Nous appliquons ici une technique de fusion des tâches selon la durée de leur temps de traitement, en fusionnant les "petites" tâches. L'algorithme exact présenté en début de ce chapitre est alors appliqué sur cette nouvelle instance. Les ordonnancements obtenus sont ensuite transposés à l'instance de départ, et seules les solutions non dominées sont conservées pour obtenir le front de Pareto.

Nous présentons en détail la technique utilisée, puis l'algorithme obtenu, avant de démontrer formellement la complexité et la garantie de performance de notre PTAS.

Un second schéma d'approximation est ensuite présenté, sous forme de FPTAS. L'espace des solutions est décomposé en zones, et, lors de la résolution du problème grâce à notre algorithme de programmation dynamique, nous ne retenons qu'une solution par zone de cet espace. Notre méthode de décomposition est détaillée, puis nous donnons une preuve de complexité de cet FPTAS.

Une troisième méthode permettant d'obtenir des solutions approchées est ensuite présentée, en combinant les deux approches précédentes. Après avoir appliqué le PTAS en fusionnant les "petites" tâches, l'instance modifiée est alors résolue en utilisant le FPTAS (et non plus la programmation dynamique comme précédemment).

La dernière partie du chapitre est dévolue à la présentation des résultats expérimentaux. Nous utilisons des instances générées aléatoirement, d'une taille comprise entre 5 et 200 taĉhes, avec divers temps de traitement et de latence. Nous comparons ensuite nos différentes approches, en faisant également varier le facteur $\varepsilon$ afin d'étudier son influence sur les propriétés de nos algorithmes. Afin de comparer la qualité des résultats obtenus, nous utilisons notamment la notion d'hypervolume.

## Résumé du chapitre 5 : Schémas d'approximation pour minimiser la date de livraison maximale et le *makespan* sur $m$ machines parallèles

Dans ce chapitre, nous nous intéressons au même problème qu'au chapitre précédent, mais le nombre de machines n'est plus limité à deux. Nous proposons une résolution exacte et une résolution approchée, en utilisant les techniques de FPTAS et PTAS vues précédemment.

Dans un premier temps, une étude bibliographique est proposée. Puis, un algorithme de programmation dynamique est introduit, afin de résoudre le problème de manière exacte. Sa complexité primale, même si elle est réduite comme lors des chapitres précédemment en éliminant les solutions sous-optimales, nous a conduit à l'étude d'heuristiques permettant de réduire encore l'espace de recherche.

Nous proposons quatre heuristiques qui ont toutes le même but : obtenir très rapidement une solution du problème. Cette solution est alors utilisée à chaque étape de l'algorithme de programmation dynamique : si la solution partielle obtenue est dominée par la solution trouvée par l'heuristique, alors cette solution partielle est éliminée.

Nous comparons ensuite l'efficacité de ces heuristiques, pour trois et cinq machines, en utilisant un sous-ensemble des instances créées au chapitre précédent. En effet, la complexité accrue nous a conduit à ne conserver que des instances de petites tailles, entre 5 et 20 tâches, et avec des temps de traitement réduits.

La suite de ce chapitre est consacrée aux schémas d'approximation PTAS et FPTAS proposés. Ceux-ci sont similaies à ceux du chapitre précédent, mais les peuves sont données dans le cas du nombre de machines $m$ fixé.

# Chapter 1

# Scheduling Problems and Approximation Concepts

*This chapter presents the main generalities about scheduling problems and the approximation algorithms that are the focus of our study. For instance, scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries. In this chapter, we first recall the concept of scheduling and the notations related to the scheduling problems. Then, we will present the complexity of the scheduling problems and the standard methods for solving the scheduling problems. The rest of this chapter is devoted to presenting the concept of approximation schemes and worst-case performance analysis. Finally, this chapter concludes with concrete problems related to scheduling problems.*

## 1.1 Introduction

The scheduling problems are one of the well-studied problems for more than sixty years. It is also related to the management of production systems, planning of large projects, optimizing the service activities and assigning the activities to available resources. Indeed, the scheduling problems can be considered as organizing activities of limited resources during the time, with the aim of optimizing one or more objectives. In addition, there may be some constraints limiting/reducing the scheduling process (release times, deadlines, operator/machine non-availability, due dates, etc.).

It is worth-noting that solving these problems is an important issue for many domains and applications. Therefore, many techniques have been elaborated to solve these problems to produce feasible solutions close to the optimum. Indeed, the approximation theory can provide many tools to carry out such an analysis [7, 48].

Let suppose $P$ an optimization problem, $A$ an approximation algorithm, $I$ an instance of problem $P$, $A(I)$ is the value for the instance $I$ generated by algorithm $A$ and $A^*(I)$ the optimal value for instance $I$.

***Definition*** *A is an approximation algorithm for problem P if for any given instance I of that problem, algorithm A returns an approximate solution, which is a feasible solution.*

**Different types of approximation can be illustrated [55]:**

- **First one, absolute approximation:** this is done by comparing the heuristic solution in the worst case with the optimal solution.
  ***Definition*** *A is an absolute approximation algorithm if there exists a constant c such that, for every instance I of problem P, we have the following relation:* $A^*(I) - A(I) \leq c$.

- **Second, the differential approximation:** this kind of approximation consists in establishing a ratio, between 0 and 1, associated with the heuristic solution in the worst case and the optimal solution (i.e., $f(A(I)) \leq \alpha \cdot f(opt(I)) + (1-\alpha) f(wst(I))$, where $\alpha$ is the differential ratio and it belongs to (0, 1).

- **Third, the probabilistic approximation:** here we calculate the expected performance of the solution yielded by the approximation-algorithm.

It is worth-mentioning that the diversity of scheduling problems makes it difficult to design a generic method for solving optimally all the problems. Thus, the design of approximation algorithms can represent an adequate alternative according to the time complexity and solution effectiveness.

In this thesis, we will deal with a deterministic scheduling problem on parallel machines. Here, the term "deterministic" means that all the parameters of the problem are known in advance. The resources refer to machines, and the activities refer to jobs that can be performed by the machines.

We aim at studying a family of scheduling problems in which, an optimal schedule can be reduced to a finite number of subsets. This is the case for many real applications well-known in the literature. More precisely, we will consider the objective of minimizing the maximum lateness. At the methodological level, we will deal with the design of approximation algorithms for these problems.

## 1.2 Scheduling Problems

Scheduling problems are one of the most common problems that can arise in many industries and services. Scheduling a set of tasks consists in determining their execution times, under some constraints, by allocating the necessary resources to the tasks in order to optimize one or more objectives. [7, 48]. In the following, we will explain the concept of resources, tasks, constraints, and objectives.

**Resources:** can be a human or technical tool to be used for performing one task or more. Different types of resources can be distinguished in scheduling problems. Hence, we can classify them according to their availability over time to renewable resources and consumable resources. Renewable resources are resources available for a period of time to perform some tasks after their use (for example: solar energy, wind energy, hydropower, biomass energy, etc.). While the consumed resources are a resource that can be used only once (example: processors, I/O channel, raw materials, etc.).

**Tasks:** represent all the processes that need to be performed, each task may have a processing time, delivery time, etc. Worth mentioning, we can classify tasks into two categories: the preemptive tasks which can be interrupted and continued later, and non-preemptive tasks which cannot be interrupted before it is completed.

**Constraints:** are restrictions that affect the project, we can distinguish between two types of constraints: first, the endogenous constraints, representing some of the conditions internally associated with the scheduling system (for example, resource capacity, resources availability dates, and number of robots). The second type is exogenous constraints imposed by external elements independent of the scheduling system. In addition, we can also classify the constraints into other types: the temporal constraints that lead to a limitation of the potential values for the starting and completion times, and the precedence constraints which are equivalent to a relative positioning between tasks.

**Objective:** in scheduling problems, it is important to evaluate and compare feasible solutions to improve the objective function. In some cases, we may look forward to improving more than one objective function. Here, the scheduling problem becomes a multiobjective problem, as we see that in Chapters 4 and 5.

### 1.2.1 Notations in scheduling problems

The machine scheduling problems can be described by the triplet $\alpha|\beta|\gamma$ [48], which describes the machine environment, job characteristics, and scheduling objectives.
**The $\alpha$ field** can be decomposed into two sub-parameters: $\alpha = \alpha_1\alpha_2$ where $\alpha_1$ is dedicated to the machine environment and $\alpha_2$ gives the number of machines. The machine environment indicates the type of machine available for processing jobs. The machine may be continuously available or have breakdowns [5].

The existence of diversity in scheduling problems makes it difficult to elaborate a generic and efficient solution to all scheduling problems and various environments. We can classify the machine environments specified in **field** $\alpha_1$ as follows:

- **Single machine shop** $(1)$: we have one machine, and $n$ jobs to be executed.
- **Flow shop** $(F)$: each job should be performed on each machine in the same order.
- **Flexible flow shop** $(FF)$: it is an extension of a flow shop. Indeed, each operation of job must be handled on one possible machine from a subset of candidate machines, in the same order.
- **Open shop** $(O)$: there are $m$ machines without restrictions in terms of routing each job through the machine environment. Indeed, each job (its operations) must be processed on a set of resources.
- **Job shop** $(J)$: each job has its own and fixed route to follow on the machines. It is possible that some jobs can pass by one machine one or more times.
- **Identical parallel machine** $(P)$: all the machines are identical in terms of speed.
- **Uniform parallel machine** $(Q)$: each machine has its own speed. These speeds are independent of the jobs.
- **Unrelated parallel machine** $(R)$: each machine has its specific speeds/performances, but these speeds depend on the jobs to be performed.

**The $\beta$ field** provides details of processing properties. Some of them are defined as follows:

- $r_j$: indicates that a release date is associated with each job.
- *pmtn*: indicates that the preemption is allowed.
- *prec*: indicates some precedence relations between jobs.

- *nwt*: indicates a no-wait constraint on machines.
- *brkdwn*: machine breakdowns means that machines will not be available during some one or more intervals for performing jobs.

**The $\gamma$ field** gives the information on the objective(s) to be optimised. For example, we can mention the following potential objectives that can be minimized:

- Makespan ($C_{max}$): is the completion time of the last performed job $j$, and can be defined as follows for a schedule $S$: $C^S_{max} = \max_j C^S_j$.
- Maximum Lateness ($L_{max}$): it measures the maximum value of the job delivery dates and it can be defined as follows: $L_{max} = \max_j (C^S_j + q_j)$.
- The total completion time $(\Sigma C_j)$ in schedule $S$ is the sum of all completion times of processed jobs (i.e, $\sum_{j=1}^{n} C^S_j$).
- The total weighted completion time $(\Sigma w_j C_j)$ in schedule $S$ is the sum of the weighted completion times of performed jobs (i.e, $\sum_{j=1}^{n} w_j C^S_j$).

### 1.2.2 Computational complexity and scheduling problems

Computational complexity theory aims at characterizing the necessary efforts or number of arithmetic resources required to solve a given task. This aspect is very important when elaborating scheduling algorithms. Indeed, the main purpose of scheduling is to find efficient algorithms for scheduling problems. Thus, it is important to evaluate the computational complexity of these algorithms. For any off-line non-preemptive deterministic scheduling problem, the total number of feasible sequences is finite. Although they may be very large, it is possible to find the optimal sequence by enumerating all the possible sequences. Indeed, this is not a practical approach when the search space is large. For many combinatorial optimization problems, it may take thousands/millions of years for all feasible solutions to be examined, even if the most powerful computing systems are used.

It should be noted that a scheduling/combinatorial problem cannot well-solved until a polynomial-time algorithm is elaborated for it. Indeed, polynomial-time algorithms typically require much less running time than non-polynomial time algorithms. Usually, the algorithm is a polynomial-time, only if the encoding scheme (the instance size of the problem is related to the used encoding scheme) is binary. The algorithm, which runs in polynomial time with respect to a unary encoding scheme, is called a pseudo-polynomial algorithm.

There are several algorithms that can be used to solve scheduling problems. Nevertheless, there are many hard scheduling problems for which it is still very difficult to obtain the optimal solution. Thus, there is no universal method to solve all scheduling problems. Indeed, the theory of complexity can help to study these problems in a mathematical and computational framework. In this theory, some major classes of problems have been identified. Some of them (the most important classes) are defined as follows [7, 25]:

- **P (Polynomial time)**: is a set of all decision problems that can be solved in a polynomial time.
- **NP (Non-deterministic polynomial time)**: is the set of all decision problems for which the solution can be verified in a polynomial time.
- **NP-complete**: are the decision problems such that if we can solve one of them (quickly) in a polynomial time, then we can solve any problem from NP.
- **NP hard**: are generally associated to the optimization version of the NP-complete problems (decision version).

Fig. 1.1 shows the Euler diagram for P, NP, NP-complete, and NP-hard problems. Under the assumption that $P \neq NP$, and $P = NP$. As we can see, under the speculation that $P = NP$, most optimization problems are NP-hard.

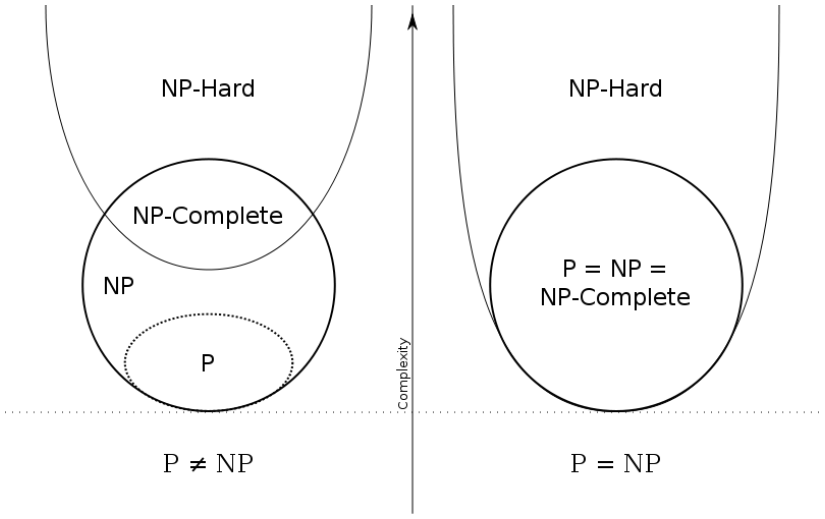



Fig. 1.1 Euler diagram for P, NP, NP-complete, and NP-hard problems [62]

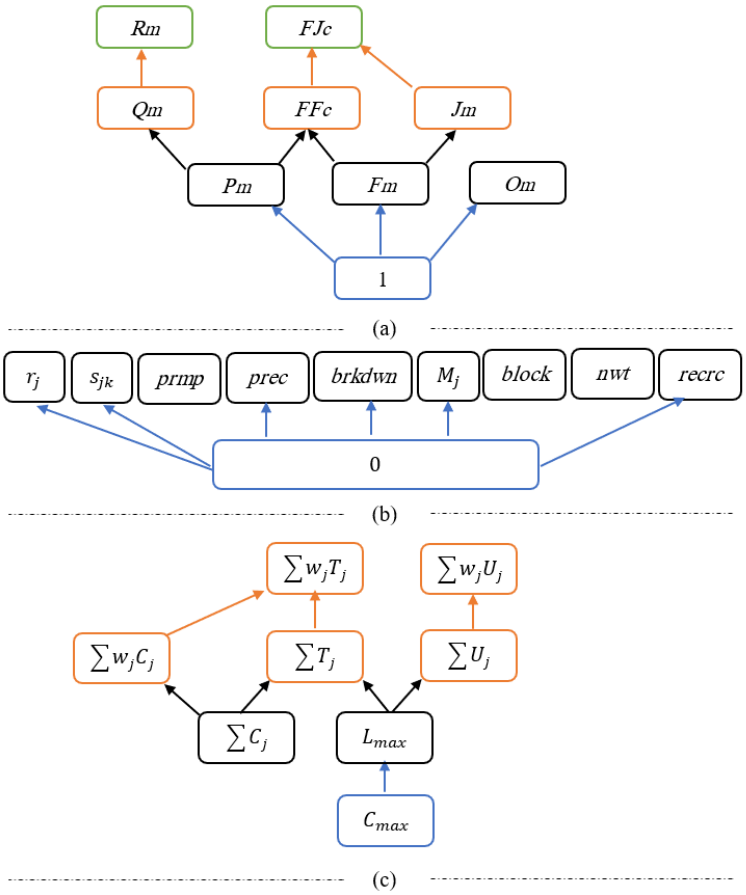



Fig. 1.2 Complexity hierarchies of deterministic scheduling problems: (a) Machine environments (b) Processing restrictions and constraints (c) Objective functions [48]

It is important to mention that a scheduling problem can be reduced to another decision problem. For example, $1||U_j$ can be reduced to $1||w_jU_j$. Fig. 1.2 depicts a number of graphs that help to understand the hierarchy of some important deterministic scheduling problems [48].

Scheduling problems are classified according to machine environments, such as single machine problems, parallel machine problems, etc. Given a scheduling problem, first, we need to determine whether the problem is polynomial or NP-hard, and if it is in the NP-hard class, we need to further investigate whether it is weakly or strongly NP-hard. It should be reminded that strongly NP-hard problems are NP-hard even for unary encoding. The weakly NP-hard problems are polynomial in the size of the instance when the instance is encoded in unary (which implies a pseudo-polynomial time complexity).

For NP-hard problems, optimization algorithms may work for small problems only. Thus, for large-scale problems, we typically aim to get a near-optimal or "good" feasible solution, since finding the optimal solution may be "expensive" in terms of running time. Noteworthy,

to evaluate the performance of the approximation algorithms, we need to analyse them in depth. Regarding the terminology, we use the *worst-case ratio* for off-line algorithms and the *competitive ratio* for on-line algorithms.

### 1.2.3 Solving the scheduling problems

As we have seen before, if the studied problem belongs to class P, there exists a polynomial algorithm to solve it, and if the problem belongs to the NP-hard class, we can distinguish between two approaches:

- The first approach is to find an approximation algorithm which produces in polynomial time a solution hopefully "near" to the optimal solution, i.e., a heuristic.
- In the second approach, we can propose an algorithm that calculates the optimal solution of the problem with exponential complexity in the worst case. In this case, the challenge is to elaborate an algorithm performing within a reasonable time.

#### $1^{st}$ Approach: Heuristic Methods

A heuristic is a technique that can be used for solving an NP-hard problem quickly compared with the exact methods. Indeed, there is no known method for solving the problem to optimality. Therefore, the use of a heuristic method for finding an approximate solution of an NP-hard problem is legitimated/pertinent. The heuristic methods are more flexible compared to the exact methods. Indeed, they can be applied to solve problems that have a high level of hardness/complexity. But, it is important to mention that any powerful heuristic requires a minimum intelligence in its design. More precisely, in a good heuristic algorithm we expect the following advantages:

- The yielded solution should be computed within a reasonable running time.
- The solution should be "near" to the optimum.

Noteworthy, there are many types of heuristic methods that are very different in nature. Therefore, it is difficult to find a unified classification for the heuristics. Sometimes, to solve a specific problem, many heuristic methods can be designed without the possibility of generalizing them to other similar problems. For the sake of illustration, we can mention the following types of heuristic methods [46]:

***Decomposition methods:*** the original problem is divided into more simple sub-problems to solve, taking into account that subproblems belong to the same problem class.

***Inductive methods:*** is the process of generalizing smaller or simpler versions to the entire case.

***Reduction methods:*** the reduction is a process of reducing the space of the solutions by simplifying the problem. These involve identifying properties that are mainly fulfilled by good solutions and introducing them as boundaries to the problem.

***Constructive methods:*** the idea behind these methods is to elaborate a solution to the problem step by step based, in general, on some priority rules.

***Local Search methods:*** these methods are based on the exploration of an existing solution of the problem and try to improve it gradually. We stop when there is a stagnation (which means that no new better solution is detected). Many variants of these methods exist (e.g., Tabu Search, Simulated Annealing ...). It is worth mentioning that the constructive and Local Search methods form the foundations of the meta-heuristic procedures.

***Population Based methods:*** these methods consist in applying iterative improvements of a set of feasible solutions (e.g., Genetic Algorithms, Swarm Particle Optimization, Evolutionary Algorithms, Ant Colonies,etc.).

### $2^{nd}$ Approach. Exact Methods

Exact methods are algorithms that can solve the optimization problems to optimality. The use of exact methods becomes impossible for the NP-hard problems when the size of the instances reach a certain limit. This is due to the computing time which is steadily increasing with the number of variables to be fixed/determined. Therefore, we can apply these methods to reach the optimal solution, under some instance size limitations, when the structure of the problem is suitable.

In this kind of methods, the aim is to explore the search space implicitly. Hence, this exploration allows us to reduce the space of solutions visited in the associated sub-space which has at least one optimal solution. Thus, we can ignore the other sub-spaces (of the non dominating solutions) in order to reduce the computation time. These methods can be applied based on the following adapted tools: dominance rules, heuristic solutions, valid cuts, exploration strategies, lower bounds, etc.
The exact methods include mainly: Branch and Bound methods, Integer Linear Programming Based methods, Dynamic Programming, etc [49].

***Branch and Bound methods:*** Branch and Bound (BB) is a general method based on the exploration of a tree search, associated to the problem, in order to find optimal solutions. In this tree, in any node, the algorithm must take a specific decision and specify one of the unbound variables. Clearly, the BB algorithm is based on the following strategies:

- **The bounding procedure:** this strategy is based on evaluation nodes, which allows us to reduce the search space by removing some subsets that cannot lead to an optimum.
- **The branching scheme:** this strategy is to divide the problem into sub-problems, and to solve or to remove these subsets. We keep the best solution that can solve the original problem.
- **The exploration strategy:** this strategy defines the order of visiting the nodes (which nodes must be considered before the others).

***Integer Linear Programming:*** Integer Linear Programming (ILP) is the type of combinatorial optimization problems with integer variables, and linear inequality constraints. The integer linear program can be stated as the following:

$$\begin{aligned} &\text{Maximize } CX, && (1)\\ &\text{Subject to: } AX \leq B\,, && (2)\\ &X \in Z^n\,, && (3) \end{aligned}$$

where the solution $X \in Z^n$ is a vector of $n$ integer variables: $X = (x_1, x_2, ..., x_n)^T$ and the data are given by matrix $A : A_{m\times n}$, matrix $C : C_{1\times n}$, and matrix $B : B_{m\times 1}$. This formula also includes constraints on equality, since each equality constraint can be reduced to two constraints of inequality (e.g., in (2)).
Many real life problems include the constraint that the variables of the program must be integers in any feasible solution. The combinatorial problems, for example, the knapsack problem, traveling salesman problem, set covering problems, spanning trees problems and many scheduling problems can be solved as integer linear optimization problems [47]. These methods include other advanced mathematical techniques such as branch-and-price or branch-and-cut techniques (based on column generation procedures, polyedral analysis, facets and valid inequalities, etc.).

***Dynamic Programming:*** gives us a way to design algorithms that systematically search all pertinent partial possibilities and store the consequences of all possible useful decisions and using this information in a systematic/recurrent way.
Dynamic Programming is a general algorithm design technique that can be applied for

solving a wide variety of discrete optimization problems (e.g., Scheduling, Packaging, and Inventory Management, etc.) defined by or formulated as recurrences, which allows us to recombine the partial solutions, associated to sub-instances [57].

## 1.3 Approximation Techniques

The standard definitions of approximation algorithms with a constant performance ratio have been presented in the beginning of this chapter. In the rest of this section, we will focus on the used techniques and on the approximation schemes. Two types of schemes can be distinguished in absolute approximation: Polynomial Time Approximation Schemes (PTAS) and Fully Polynomial Time Approximation Schemes (FPTAS). Noteworthy, an approximation scheme is a suboptimal approach based on an input parameter: the accepted error bound. It works quickly, whith the aim of getting a solution for an NP-hard problem by respecting the accepted error bound [54].

It is important to mention that these techniques can use linear programming tools or sophisticated algorithmic techniques which lead to difficult implementation problems. Furthermore, some approximation algorithms have impractical running times even though they are polynomial time [63].

***Definition*** Let $\varepsilon > 0$. An algorithm $A$ is called a polynomial-time approximation scheme (PTAS) for an NP-hard problem, if for any instance $I$ of that problem the algorithm $A$ yields, within a polynomial time for the fixed $\varepsilon$, a feasible solution with an objective value $A(I)$ such that [54]:

$$|A(I) - OPT(I)| \leq \varepsilon.OPT(I),$$

where, $OPT(I)$ is the optimal value of instance $I$. For the minimization problems, this leads to inequality $A(I) \leq (1+\varepsilon)OPT(I))$. For the maximization problems, it leads to inequality $A(I) \geq (1-\varepsilon)OPT(I))$.

When a PTAS runs polynomially in $1/\varepsilon$ and the instance size, it becomes an FPTAS (fully polynomial approximation scheme). Such a scheme is the most powerful approximation algorithm that we can expect for an NP-hard problem.

From these definitions, it is clear that approximation schemes are clearly stronger than constant approximation algorithms. As an illustration, a possible complexity of an PTAS would be $|I|^{2/\varepsilon}, (|I|^{2/\varepsilon})^{1/\varepsilon}, (|I|^{2/\varepsilon})^{10}$ , i.e., exponential in $1/\varepsilon$, but polynomial in the input size of $I$ (denoted by $|I|$) for a fixed value of $\varepsilon$. For an FPTAS, this time complexity would be for example $|I|^8/\varepsilon^3, |I|^2/\varepsilon, |I|/\varepsilon^2$...

### 1.3.1 Approximation-based problems classification

Since the seventies, the structure of approximation classes has been a major issue. The existence of such algorithms is important when studying an NP-hard problem since through the approximation analysis, we can identify optimization problems of good properties or bad approximability behavior. Hence, we can classify the NP-hard problems as follows [21]:

- Ratios depending on the size of $I(|I|)$:
  - **Exp-APX** (e.g., general Travelling Salesman Problem).
  - **Poly-APX** (e.g., Graph Coloring Problem).
  - **Log-APX** (e.g., Set Covering Problem).
- Constant ratios (independent of $|I|$):
  - **APX** (e.g., $R||C_{max}$).
- Ratios $1+\varepsilon$, for any $\varepsilon > 0$:
  - **PTAS** (e.g., $P_m||C_{max}$).
  - **FPTAS** (e.g., $2||C_{max}$).

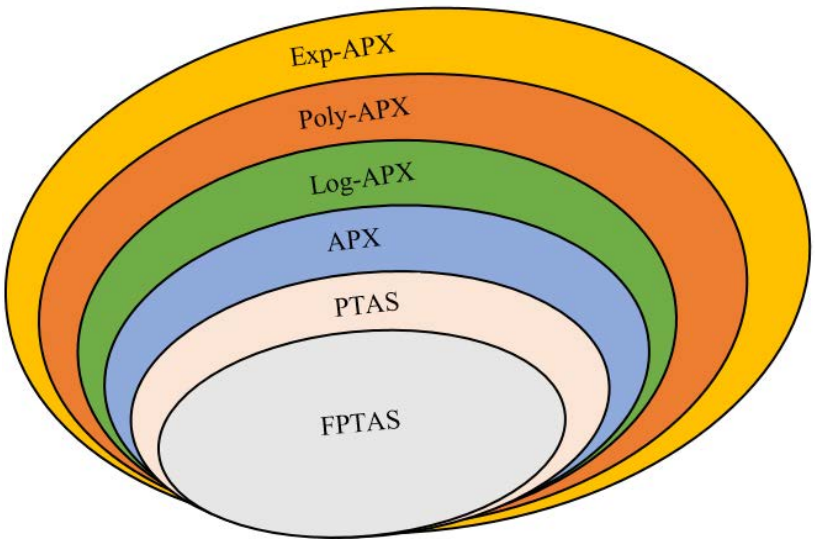



Fig. 1.3 Approximability classes for NP-hard problems (under the assumption $P \neq NP$) [21]

### 1.3.2 Approximation design techniques

It is important to mention that there exist many standard techniques that can be applied to establish an approximation algorithm. These include the following ones.

- Greedy algorithm.
- Local search.
- Enumeration and dynamic programming.
- Use a convex programming relaxation to get a fractional solution. Then, we transform this fractional solution into a close feasible solution by applying some rounding procedures at a subset of variables. Noteworthy, the standard relaxations include:
  - Linear programming relaxation.
  - Semidefinite programming relaxation.

One of the well-known approaches used in the elaboration of approximation schemes is based on identifying a specific structure of the input data. It consists in applying the following steps:

- Establishing a simplified instance based on the indicated precision $\varepsilon$.
- Solving the simplified instance, which needs to be carried out in polynomial time.
- Exploiting the optimal solution of the simplified instance to derive a good solution for the original instance.

In the approximation area, we are interested in analyzing the polynomial time approximation algorithms/schemes to achieve the best possible performance. To establish approximation schemes, several techniques and methodologies can be applied, mainly classified into three approaches as follows:

- The first approach is based on the modification of the input in order to reduce it to an easier instance.
- The second approach consists in reorganizing the output (solution) space.
- The third approach is based on truncating the execution of an exact algorithm.

To explain these approaches, let us recall that any instance $I$ of the studied problem $X$ needs that we apply an algorithm $A$ and processes it to produce its solution $A(I)$. This solution will be optimal if $A$ is exact. If the NP-hard problem $X$ is difficult to solve, then the exact algorithm $A$ will have a bad time complexity. Therefore, to improve the behavior of such an algorithm and possibly convert it to a PTAS, we can do one of the previous three approaches. In the next paragraphs, we will discuss these three methods [54] in the remainder of this section.

### Modification of the input

This approach can be described in the following three steps:

1. **Simplification**: simplifying the hard instance $I$ into a "simpler" instance $I^{\#}$, which needs less computational efforts to be solved. Such a simplification will be depending on the expected precision $\varepsilon$. The time complexity of this step must be polynomial in the input size for a fixed value of the precision parameter.

2. **Solve**: the simpler instance $I^{\#}$. We must identify the optimal solution $OPT^{\#}$ (or a very good approximation) for $I^{\#}$.

3. **Project**: finally, we have to "project" solution $OPT^{\#}$, obtained for instance $I^{\#}$, into a feasible solution $APP$ for the original instance $I$. This projection must induce a good level of closeness to the optimal solution (see Fig. 1.4).

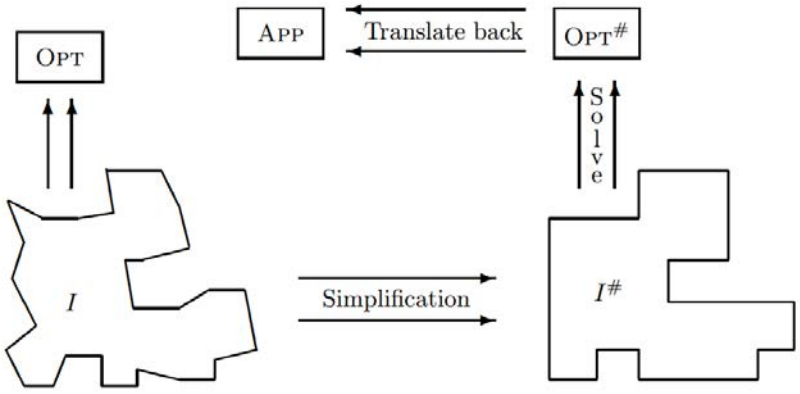



Fig. 1.4 Structuring the input [54]

It is important to mention that there are several known techniques that allows us to achieve the instance modification. These techniques includes mainly the following possibilities:

1. **Rounding**: here, the values of some data are rounded. In general, these data are augmented or decreased to the next or the previous multiple or power of a specific unit (see Fig. 1.5).

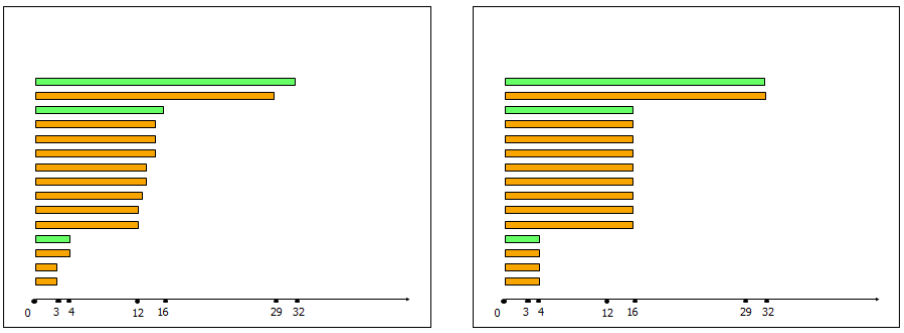



Fig. 1.5 Example of rounding

2. **Merging**: here, we can for example merge or put together some small pieces having some close properties into large pieces. As it can be depicted in Fig. 1.6 we can merge a great number of small jobs into a single large job.

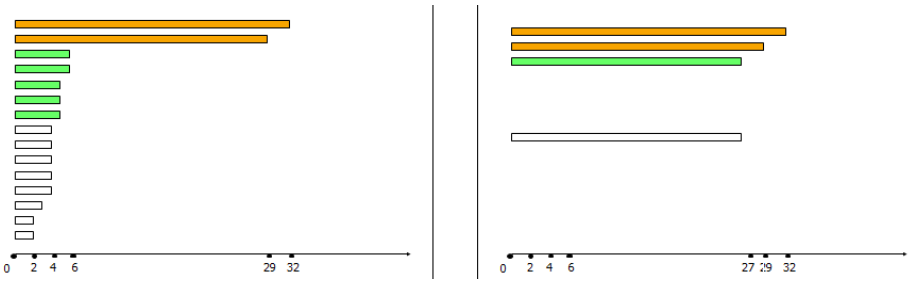



Fig. 1.6 Example of merging

3. **Cutting**: here, we can reduce the instance irregularity by removing a subset of data. As we can observe in Fig. 1.7, we can eliminate a subset of small jobs from the instance.

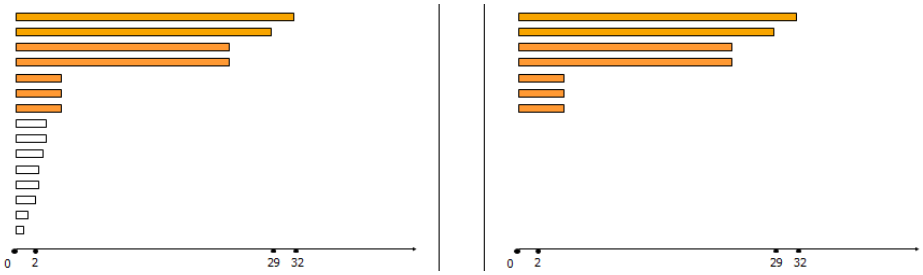



Fig. 1.7 Example of cutting

4. **Aligning**: here, as we can see in Fig. 1.8, we can replace a subset of small jobs by the same copy/job from the instance. Consequently, the instance is simpler.

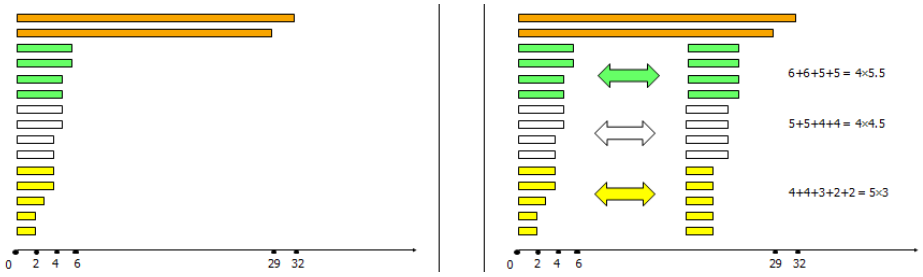



Fig. 1.8 Example of aligning

### Reorganization of the output

Another standard method used to elaborate approximation schemes, consists in dividing the output space (i.e., the space of feasible solutions) into a polynomial number of small areas. Every small area will be represented by one approximate solution. Consequently, we can take the best approximate solution in all these areas to obtain a good global approximate solution. This method can be detailed as follows:

1. **Partition**: the feasible solution space $S$ of instance $I$ into a polynomial number of areas $S^{(1)}, S^{(2)}, ..., S^{(j)}$ such that $\bigcup_{i=1}^{j} S_j = S$.

2. **Determine a representative**: for each district $S^{(i)}$. Such a representative should have a good objective value $APP^{(i)}$ near to the best objective value $OPT^{(i)}$ in area $S^{(i)}$. Determining each representative must be done in polynomial time in the instance size.

3. **Keep the best**: representative from all representatives as an approximate solution with the objective value $APP$ for instance $I$.

### Truncation of the execution of an exact algorithm

The main principle of this technique is to take an exact algorithm and to reduce it in order to derive a truncated polynomial algorithm. Note that during the execution process, the truncated algorithm may need some additional data. As a consequence, we must remove some of these additional data in order to make it more effective.

## 1.4 Some related existing works

Given the aim of the thesis, this section will be dedicated to review some representative and related works: makespan and maximum lateness minimization, parallel-machine scheduling, scheduling under non-availability constraints. In particular, we will focus on approximation results. It is worth-mentioning that over the past decades the approximability of many scheduling problems has been widely studied in the literature. For instance, one of the well-known PTASs has been proposed in 1989 by [27] for the problem of two-machine flow shop with release dates to minimize the maximum completion time. Furthermore, they have studied the problem of scheduling single and parallel machine with release times to minimize the maximum lateness. They also discussed two scheduling problems with precedence constraints and introduced new approximate results. Leung and Ng [44] proposed a fast approximation algorithm for a non-preemptively scheduling problem on uniform machines with processing set restrictions. They proposed a polynomial-time algorithm for two cases (inclusive processing set and tree-hierarchical processing set), with a running time $O(\log P^* m^* n)$ and worst-case bound of $4/3$. Moreover, they showed that the bounds are achievable. A polynomial approximation scheme on uniform parallel processors has been presented in [19] to minimize the makespan. They used a dual approximation approach to convert the infeasible solutions of the related problem into the desired feasible solution.

Noteworthy, the problems of scheduling a single machine have been widely studied with and without non-availability constraint. For instance, Kacem and Levner [39] considered a single machine problem to minimize the total completion time with non-resumable jobs under maintenance constraint. They introduced a DP method and an FPTAS. Kacem et al. [28] considered a single machine problem, with aim of minimizing the maximum lateness with a non-availability interval. They established that $\frac{\sqrt{37}+1}{6}$ is a lower bound for the competitive ratio of any online algorithm associated to the breakdown model; They also established that $\frac{9}{\sqrt{31}+2}$ is a valid lower bound for the competitive ratio of every online rule for solving the emergent job model. They obtained a 2-approximation for the two models and proved that these ratios are tight. In [34], Kacem considered the problem of scheduling a single non-resumable machine with a fixed non-availability interval to minimize the maximum lateness. He proposed a polynomial approximation algorithm with a tight worst-case performance ratio of 3/2. He also proposed a dynamic programming algorithm and showed that the problem has an FPTAS with strongly polynomial complexity.

In [36], Kacem and Haouari proposed approximation algorithms for the single machine scheduling problem with release dates and tails, under one non-availability constraint. They addressed the maximum lateness criterion. They showed that the problem has an FPTAS

when the tails are equal, a $(2+\varepsilon)$-approximation has been deduced for the general case. Moreover, showed that the worst-case performance of Schrage's algorithm is not affected by a single non-availability interval and established that its tight bound is 2. Later, Kacem and Kellerer showed that the general case admits a PTAS.

Kacem and Chu [35] considered a single machine scheduling problem, with aim of minimizing the weighted sum of the completion times with a non-availability period. They proved that the WSPT and MWSPT heuristics, have the same worst-case performance ratios (3 for the two heuristics under some conditions), and they showed that these worst-case performance ratios are tight. Kacem and Mahjoub [40] proposed an efficient FPTAS with $O(n^2/\varepsilon^2)$ time complexity.

Cui and Lu [16] considered a single machine scheduling problem, to minimize the makespan with flexible periodic preventive maintenance and release dates. They presented two algorithms, Branch and Bound algorithm (BB) to solve small and medium-sized problems and Earliest Release Date (ERD-LPT) which have a high degree of accuracy and effectiveness. Furthermore, they proved that the improvement of the integration between production scheduling and Preventive Maintenance (PM) is significant compared to the First-in-First-out (FIFO) rule.

Kacem et al. [29] considered a single machine scheduling problem, with aim of minimizing the maximum lateness with a fixed operator, where jobs have positive tails. Noteworthy, two cases were considered, the non-availability of the machine and the non-availability of the operator. They established two FPTAS and showed that the two algorithms are strongly polynomial.

An FPTAS to minimize the makespan has been proposed in [41]. They showed that each algorithm of the scheme runs in $O(n^5L^4/\varepsilon^3)$ time. In [61], Wan proved that the dynamic programming proposed in [41] is incorrect. Therefore, he presented a correct DP, based on the new derived FPTAS.

Fang et al. [33] studied the single machine scheduling problem, where the objective is to minimize the total electricity cost under time-of-use electricity tariffs. They demonstrated that the non-preemptive on a uniform speed of the problem is strongly NP-hard, unless $P = NP$. They also gave exact polynomial-time algorithms for preemptive on a uniform speed. Moreover, they proposed different approximation algorithms.

Kacem and Kellerer [38] proposed an approximation algorithms for the $1,h_1|r_j|\sum p_jC_j$ problem. They showed that the trivial FIFO (First In First Out) sequence can lead to an arbitrary large worst-case performance bound, and the MFIFO has a performance ratio of 2. Moreover, they proved that this performance ratio is tight. Finally, they showed the existence

of an FPTAS for which the time complexity is strongly polynomial.

On the other hand, scheduling problems on the parallel machine have attracted several researchers and have been widely studied in the literature. Kacem et al. [31] studied the two-parallel machines where one of the machines has a non-available interval, with the aim of minimizing the weighted completion time. They proposed a dynamic programming algorithm and they showed that the problem has an FPTAS with a strongly polynomial time complexity.

Kacem and Hifi [37] considered a two-parallel machines scheduling problem with release times to minimize the makespan. They proposed two approximation heuristics with performance ratios of 2 and $3/2$, and proved that these performance ratios are tight. Moreover, they established that the problem admits an FPTAS, that runs in a strongly polynomial time. In [45], Lin and Jeng proposed approximation algorithms for the parallel machine scheduling problem to minimize the maximum lateness and the number of tardy jobs. They introduced a DP for the problem. They have also developed heuristic algorithms and conduct computational experiments to study relative performance. Hebrard et al. [20] studied the parallel machines with additional unit resources, with the aim of minimizing the makespan. They proved the NP-hardness of the problem on two machines and its strongly NP-hardness in the general case. They also proposed approximation heuristic, which is a $(2-\frac{2}{m+1})$-approximation algorithm bounded by a ratio of $\rho$, and showed that this bound is tight by a factor of $(1+\rho\frac{m-1}{n})$. Moreover, they showed that the existence of $(2-\frac{1}{m})$-approximation of such batch sequences. Furthermore, the approximation ratio between maximum and minimum processing times is bounded by $\lfloor\frac{s-1}{m-1}\rfloor$, and the proposed algorithm approximates the optimal schedule within a factor of $(1+\frac{s-1}{n})$.

Agnetis et al. [2] addressed the problem of non-preemptive scheduling jobs on parallel machines subject to exponential unrecoverable interruptions, with the aim of maximizing the expected amount of completed work. They demonstrated its NP-hardness even for two identical machines. Furthermore, they proposed an exact algorithm which has a pseudo-polynomial complexity and a heuristic algorithm. Györgyi and Kis [26] studied the parallel machines with non-renewable resources, with the aim of minimizing the makespan. They proposed a PTAS and showed that the problem is APX-complete even if we have only two resources. Kacem et al. [30] studied the two-parallel capacitated machines scheduling problem to minimize the total weighted completion time, where one of the machines has non-available interval. They introduced a DP method and showed that the problem admits an FPTAS that runs in a strongly polynomial time. Sen et al. [58] proposed an effective algorithm to minimize the maximum job lateness and flowtime in the two-machine system.

They proposed a Branch-and-Bound solution to reach the optimal solution. Also, Cheng and Gen [15] proposed an efficient algorithm to minimize the maximum weighted absolute lateness.

Oron et al. [17] studied the workload partition problem, with the aim of minimizing the total cost of the set of $m$ identical machines. They showed that the problem is strongly NP-hard when the number of machines is not fixed. Furthermore, they proposed two approximation algorithms. The problem of two-agent on a single parallel-batching machine with equal processing time and non-identical job sizes have been studied in [32] to minimize the makespan of one agent subject to an upper bound on the makespan of the other agent. They showed that there is no polynomial-time approximation algorithm for the studied problem unless $P = NP$. Moreover, they proposed an algorithm to obtain a lower bound of the optimal solution. Furthermore, they proposed two heuristics: reserved-space heuristic and dynamic mix heuristic. Rudek [51] studied the parallel processor scheduling problem with varying processing times to minimize the maximum job completion times. He proposed a pseudo-polynomial dynamic programming algorithm, and an FPTAS.

Given the aim of the thesis, in this section, we summarized the main results in the literature on the single-machine and two-parallel machines problems with and without non-availability constraints, to achieve different objectives. In Chapter 3, we will address our contribution to the design of approximation algorithms, for solving the problem of two-parallel machine scheduling with a non-availability interval to minimize the maximum lateness. It is worth-mentioning that, to the best of our knowledge, this problem has not been studied before in the literature. Thus, the design of efficient methods for this problem is a new attempt.

# Chapter 2

# Multi-objective Optimization

*Multiple-objective Optimization Problem (MOP) is one of the most challenging areas facing researchers since the 1950s. This problem aims in general at finding a special subset of "good" solutions. These solutions are called "Pareto-optimal" and they are defined as those not dominated by other solutions. This chapter tries to cover the main aspects/properties related to multi-objective combinatorial optimization. For instance, we will illustrate the concept of dominated and non-dominated solutions, Pareto optimality, and the approximation of Pareto-optimal solutions. The rest of this chapter tries to describe the main techniques that can be used to solve such problems. In addition, we will present the main design issues of multi-objective metaheuristics. Finally, this chapter concludes with some concrete problems related to multi-objective combinatorial optimization.*

## 2.1 Introduction

Generally, Multi-criteria Decision Making (MCDM) refers to the decision-making process in multiple/conflicting criteria. Indeed, there are two types of MCDM problems: the first type is Multi-objective Decision Making that contains an infinite number of alternative solutions. Thus, the non-dominated Frontier has an infinite number of non-dominated points. The second type of MCDM problems is Multi-attribute Decision Making that contains a finite number of alternative solutions. Thus, the non-dominated frontier has a finite number of non-dominated points. Noteworthy, MCDM problems may not always have a definitive or unique solution. Therefore, different names are given to different solutions depending on the nature of the solutions.

## 2.2 Multiple-objective Optimization

Multi-objective Optimization (also known as, multi-criteria optimization, multi-attribute optimization, or Pareto optimization) was highlighted as an area of MCDM that takes into account optimization problems involving more than one objective function.

Multiple-objective Optimization Problem (MOP) is one of the challenging areas faced by researchers in decision sciences since the 1950s, with many papers and books (see for example [60, 56, 6, 11, 53, 23, 7]).

The importance of multiple-objective optimization derives from the proliferation of multi-criteria problems in almost all deciding-making areas such as manufacturing, engineering, transportation, biology, economics, business, healthcare systems, and supplies. For instance, the development of a new component may involve reducing the weight to a minimum. While increasing the strength or choosing a portfolio may include maximizing the expected return while minimizing the opportunity. Noteworthy, there are significant differences between the multi-objective optimization problems that allow the preemption, which are called preemptive (often easier in mathematical terms and have an infinite number of schedules), and those that are non-preemptive (having a finite number of schedules).

It should be noted that many multi-objective scheduling problems are NP-hard. Thus, it is necessary to find schedules with better performance. The good news is that there are various modifications of traditional approaches that can be exploited to solve multiple objective scheduling problems. These approaches include the following [48]:

- Procedures based on dispatching rules;
- Branch-and-bound and filtered beam search;
- Perturbation analysis;
- Local search technique;
- Approximation algorithms.

Multi-objective optimization involves in the most of times a set of conflicting objectives. We can use one objective function by connecting the two objectives with a common cost advantage. Typically, some optimization problems cannot achieve some ways to reach competitive objectives. Therefore, the relative importance of those objectives cannot be measured numerically [4].

In general, a Multi-objective Optimization Problem (MOP) can be defined by a vector function as follows:

$$MOP = \begin{cases} \min & f(x) = (f_1(x), f_2(x), ..., f_n(x)), \\ s.t, & x \in X. \end{cases}$$

We assume that: for each decision vector $x \in X$ assigns an objective vector $z \in Z$, where $X$ is the feasible set in the decision space and $Z$ is the feasible set in the objective space. Here, the possible values of the objective function $f(x) = f_1(x), f_2(x), ..., f_n(x)$ constitute the objective space $Z$, where $n \geq 2$ (see Fig. 2.1).

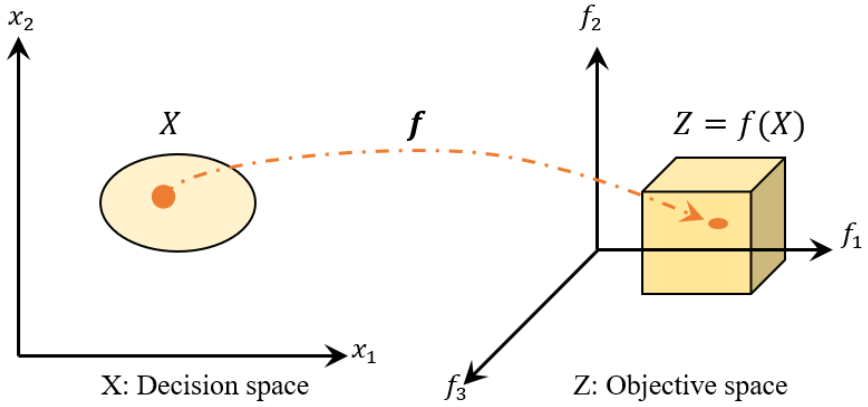



Fig. 2.1 Decision space and objective space

Worth mentioning, for the problem of single-objective, we have only one objective to optimize it in the objective space, while we have at least two objectives in the multi-objective optimization problem. Hence, the Pareto-optimal set plays an important role in this type of problems. In the next section, we will discuss the concept of Pareto dominance [3].

## 2.3 Dominated and Non-Dominated Solutions

**Dominant solution**: the solution $x$ dominates another solution $x^*$, if and only if $x$ is better than $x^*$ at least for one objective and it is equivalent to $x^*$ for all the other objectives.
**Non-Dominant solution**: the solution $x$ is non-dominated in the set $S$, if and only if there is no solution $x^*$ in $S$, which dominates $x$.

Clearly, suppose we aim to minimize two objectives $(f_1, f_2)$, we can say that the objective vector $z \in Z$ dominates an objective vector $z^* \in Z$, if both of the following conditions are true:

- $\forall_i \in \{1,2,...,n\}, z_i \leq Z_i^*$
- $\exists_j \in \{1,2,...,n\}, z_j < Z_j^*$

In other words, the objective vector $z \in Z$ dominates an objective vector $z^* \in Z$, if and only if, for all $i$ in $\{1,2,...,n\}$ so that $z_i \leq Z_i^*$, as well as there are some of $j$ in $\{1,2,...,n\}$ so that $z_j < Z_j^*$.

For example, as we can see in Fig. 2.2, solution $A$ dominates solution $B$, because $A$ is better than $B$ for all objectives $(f_1, f_2)$. On the other side, if we compare solution $A$ and solution $C$, it is clear that these two solutions cannot dominate each other. Indeed, $A$ is better than $C$ for $f_1$ objective and $C$ is better than $A$ for the $f_2$ objective.
It is worth mentioning that if we are maximizing one or both of these objectives, the direction of the inequalities will change.

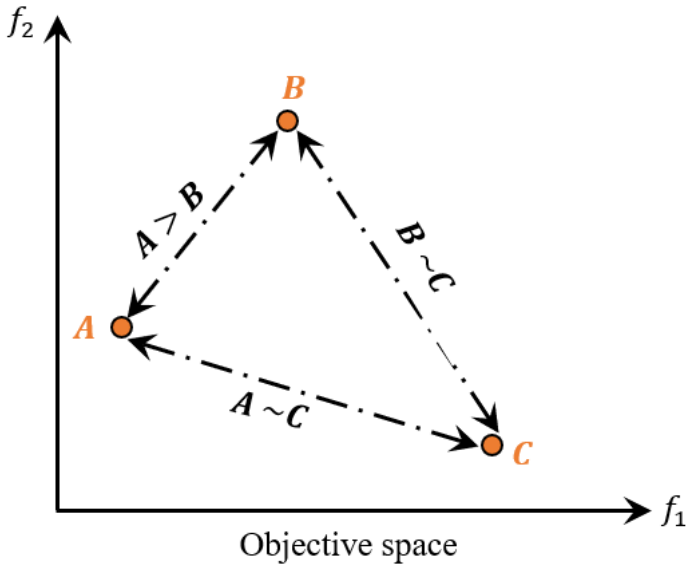



Fig. 2.2 Solution $A$ dominates solution $B$

Furthermore, we can say that the objective vector $z \in Z$ is non-dominated (or eligible, efficient, non-inferior, Pareto-optimal), if and only if there is no objective vector $z^* \in Z$ such that $z^*$ dominates $z$ .

Finally, the decision vector $x \in X$ dominates a decision vector $x^* \in X$, if $f(x)$ dominates $f(x^*)$. On the other hand, a decision vector $x \in X$ is non-dominated (or eligible, efficient, non-inferior, Pareto-optimal), if $f(x)$ maps to a non-dominated point. The set of all non-dominated solutions is called a Pareto-optimal set (or Pareto-optimal solutions) and its mapping in the objective space is called Pareto front (See Fig. 2.3).

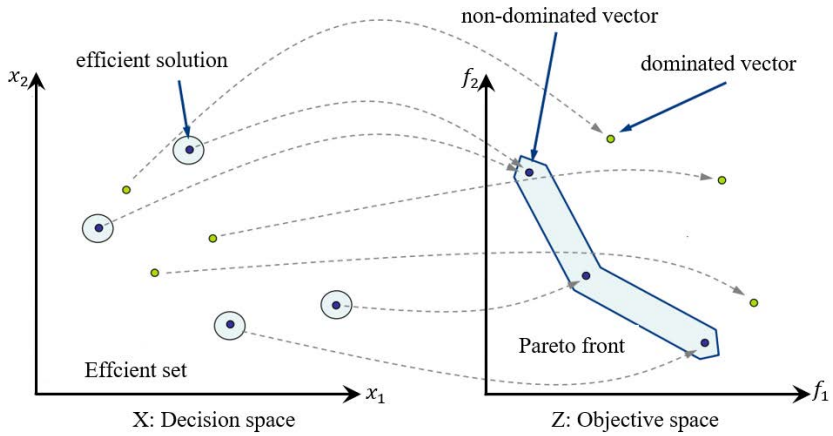



Fig. 2.3 Efficient sets in decision space and Pareto front in objective space

## 2.4 Primary and Secondary Objectives

In practice, scheduling is often concerned with more than one objective. For example, machine scheduling problems with two objectives can be described by $\alpha|\beta|\gamma_1,\gamma_2$ which describes the machine environment, the job characteristics, the primary objective $\gamma_1$ and the secondary objective $\gamma_2$. Typically, scheduling problems can be solved with respect to the primary objective. On the other hand, usually, there is no uniform solution for all objectives. Hence, if we have more than one optimal solution we can choose the best solution according to the secondary objective. For example, a schedule that minimizes a combination of makespan $C_{max}$ and maximum lateness $L_{max}$ on two parallel machines without constraints can be described by $2||C_{max},L_{max}$.

## 2.5 Pareto Optimality

Normally, there is no single global solution to all scheduling problems. Therefore, it is often necessary to define a set of satisfactory dominant points. It should be noted that the dominant concept of optimal point determination is the Pareto concept, which is defined as follows:

***Definition*** *Pareto solutions are solutions that are not dominated by other solutions. Therefore, we can call the solution a Pareto-optimal solution if this solution is better than all other solutions for all objectives.*

***Definition*** *Pareto Frontier is a Pareto set that is placed in an objective space (with drawing the objectives along each axis).*

Clearly, Pareto solutions represent a range of optimal and reasonable solutions for all possible functions based on different objectives.

Suppose our objectives are $\gamma_1$ and $\gamma_2$. If the overall objective is $\Theta_1\gamma_1 + \Theta_2\gamma_2$ where $\Theta_1$ and $\Theta_2$ are the weights of the two objectives, then we can refer to the scheduling problem by $1|\beta|\Theta_1\gamma_1 + \Theta_2\gamma_2$. It should be noted that all Pareto solutions can be represented by a set of points. This set of points shows the trade-offs between the two objectives $(\gamma_1, \gamma_2)$. Consider the two following objectives: the total completion time $\sum C_j$ and the maximum lateness $L_{max}$. Two of the extreme points of the trade-off curve can be considered as follows [48, 18]:

$$\text{If } \Theta_1 \to 0 \text{ and } \Theta_2 \to 1, \text{ then } 1|\beta|\Theta_1\gamma_1 + \Theta_2\gamma_2 \to 1|\beta|\gamma_2(opt)$$
$$\text{If } \Theta_1 \to 1 \text{ and } \Theta_2 \to 0 \text{ then, } 1|\beta|\Theta_1\gamma_1 + \Theta_2\gamma_2 \to 1|\beta|\gamma_1(opt)$$

In addition, the trade-offs between total completion time and maximum lateness are described in Fig. 2.4.

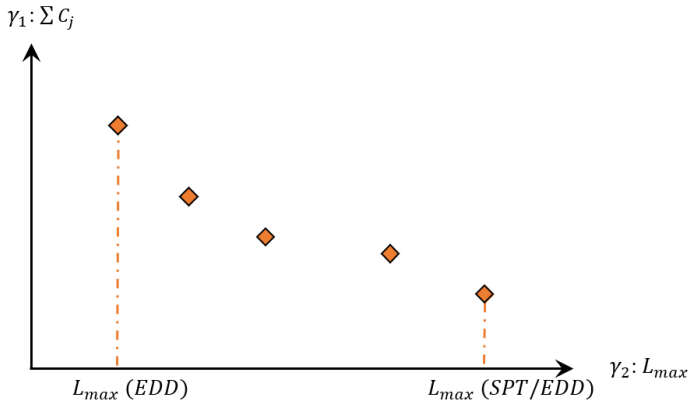



Fig. 2.4 Trade-offs between total completion time and maximum lateness

## 2.6 Approximation of Pareto-optimal Solutions

The most common concept of approximation for the Pareto set is based on the concept of the $\varepsilon$-Pareto set. It can be defined as follows: for all $\varepsilon > 0$, an $\varepsilon$-Pareto set is a set $P_\varepsilon$ of solutions that approximately dominate all other solutions. That is, for every solution $s$, the set contains a solution $s^*$ that is good approximately within a factor $(1+\varepsilon)$ in all objectives. The study of the existence of the $\varepsilon$-Pareto set (some of these sets can be small or large) has been investigated for many multi-objective optimization problems, as we shall see in the literature review.

## 2.7 Techniques for Solving Multi-objective Problems

It is worth mentioning that multi-objective optimization (MOP) as a part of the decision-making process can be classified into three classes. These classes have an interaction between the algorithm and the decision maker. This interaction can be expressed during the resolution process as follows. The decision maker preferences may be expressed:

- **A priori**: before the results of the optimization process are known (i.e. before the resolution process).
- **A posteriori**: which has no preferences before (i.e. after the resolution process).
- **Interactive**: in this case, the decision maker interaction will be during the resolution process [14].

A classification of multi-objective optimization algorithms is described in Fig. 2.5. The most popular scalarization techniques for solving multi-objective optimization problems are scalarization approach, $\varepsilon$-constraints method, goal programming, multi-level programming, weighted metrics, and aggregation.

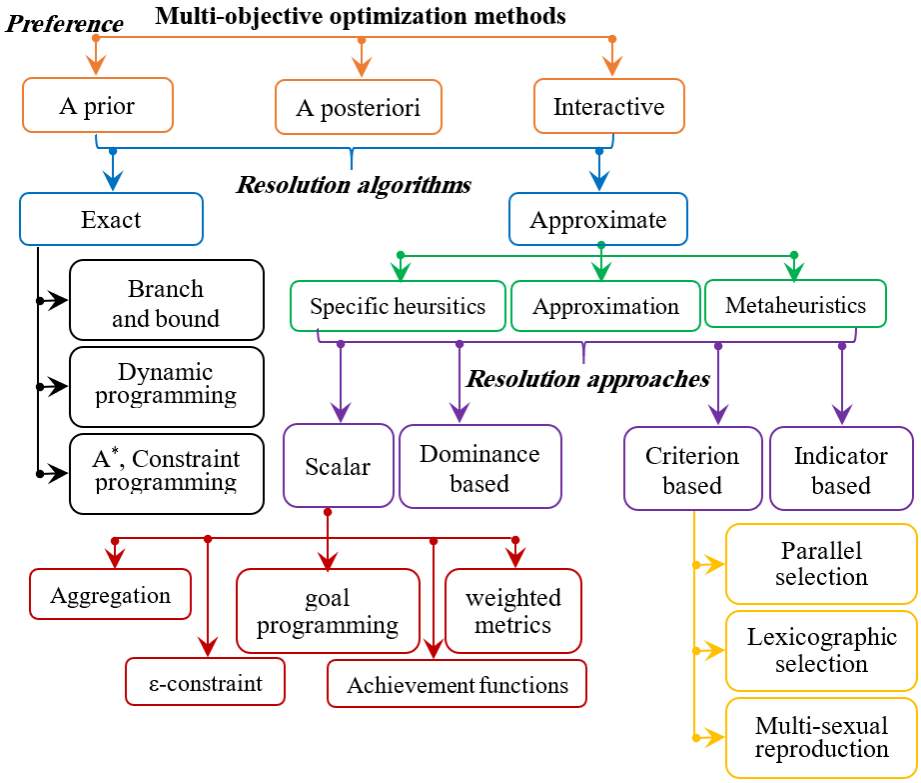



Fig. 2.5 Classification of multi-objective optimization algorithms

## 2.8 Main Design Issues of Multi-objective Heuristics

Optimization algorithms for solving multi-objective optimization algorithms can be classified into exact and approximate algorithms. Noteworthy, during the last decades, the bi-criteria optimization problems have attracted many researchers from all over the world and have been widely studied in the literature using exact methods such as Branch and Bound and Dynamic Programming algorithms. Indeed, exact search methods are effective for small-size problems. For problems with more than two criteria, there are no accurate and effective measures due to simultaneous difficulties of the NP-hard complexity and the framework of multi-criteria optimization problems. However, there are some new techniques proposed in the literature for multi-objective problems.

Heuristic methods can be used to solve large-scale or multi-criteria problems to find the Pareto approximation set. Hence, approximate methods can be divided into two classes as follows:

- The algorithms for a particular problem using some knowledge about the problem.
- The metaheuristics (general purpose algorithms) that can be used for a large variety of multi-objective problems.

It is noteworthy that the application of metaheuristics to solve multi-objective problems have attracted numerous researchers from all the world and has become an active development area to obtain an approximate set of Pareto optimal solutions.

Suppose our goal is to find a good approximation for two-objective of our study $(f_1, f_2)$. Indeed, approximating an efficient set is itself a bi-objective problem. As we can see in Fig. 2.6, the minimum distance to the Pareto front gives us a very good convergence (efficient set approximation) and very bad diversity (See Fig. 2.6, A). On the other hand, the maximum diversity in the objective space (and/or decision space) gives us a very good diversity (efficient set approximation) and very bad convergence (See Fig. 2.6, B). Therefore, when designing an efficient multi-objective algorithm, the approximation must meet the good properties of both convergence and diversity (See Fig. 2.6, C) [59]. Hence, the main difference between designing a heuristic for a single-objective and multi-objective relates to these two criteria.

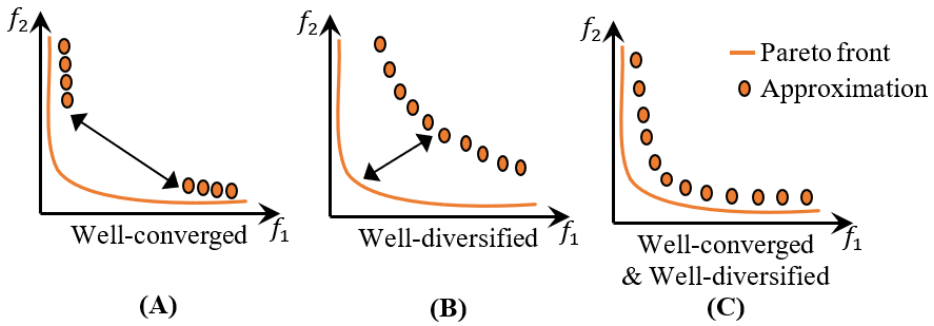



Fig. 2.6 Examples of Pareto fronts with properties of convergence and diversity

Indeed, the multi-objective heuristics, including especially metaheuristics, will have a contribution based on the three main search components (fitness assignment, diversity preserving and elitism) that will be described in the next paragraphs.

### 2.8.1 Fitness assignment

This procedure is to guide the search algorithm toward Pareto optimal solutions for better convergence. It is worth mentioning that in the case of single-objective, the fitness value procedure maps a fitness vector to a single value. However, in the case of multi-objective, the fitness value procedure aims to guide the search toward Pareto optimal solutions for better convergence. According to the fitness assignment strategy, multi-objective heuristics may be classified into four different approaches as follows (See Fig. 2.5):

- **Scalar Approaches:** these approaches include the methods that turn a multi-objective problem into a single-objective or a combination of these problems. Moreover, among these approaches, one can find many methods such as the aggregation (or weighted) methods, the weighted metric methods, the goal programming methods, the achievement functions methods, the goal attainment methods, and the $\varepsilon$-constraint methods.
- **Criterion-Based Approaches:** in these approaches, each objective is treated separately. Moreover, among these approaches, we can mention the parallel approach (e.g., parallel selection in evolutionary algorithms, parallel pheromone update in the ant colony optimization) and lexicographic (sequential) approach.
- **Dominance-Based Approaches:** in these approaches, we do not need to convert the multi-objective problem into a single-objective. Therefore, we use the concept of dominance in the fitness assignment and Pareto optimality to guide the search process.
- **Indicator-Based Approaches:** these approaches use some performance indicator to guide the search for better convergence.

### 2.8.2 Diversity Preserving

Although the previously mentioned methods of fitness assignment tend to guide the search algorithm toward Pareto optimal solutions for better convergence, these methods are unable to generate a diverse subset of non-dominated solutions (Pareto optimal solutions) in the objective space (and/or decision space). Therefore, diversity preservation strategies can generate a diverse subset of non-dominated solutions (Pareto optimal solutions) in the objective space (and/or decision space). In addition, the most common diversity preservation methods are classified into three main categories used to estimate statistical density, namely, kernel methods, nearest-neighbor methods, and histograms (See Fig. 2.7).

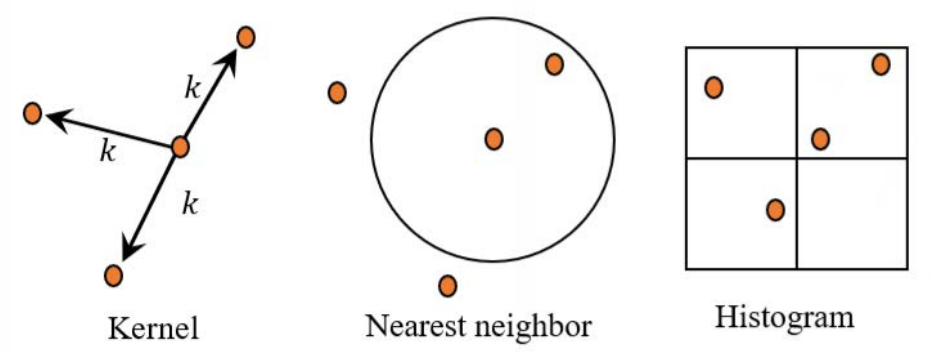



Fig. 2.7 Diversity maintaining strategies

### 2.8.3 Elitism

The aim of elitism is to preserve and use elite solutions (e.g., Pareto optimal solutions that the algorithm has found). This will allow us to ensure strong, fast and improved performances of the heuristic.

## 2.9 Some existing related works

Given the aim of this work, we recall some representative and related existing works to the investigated subject: multiobjective scheduling, makespan and maximum lateness minimization, exact and approximation search.

For example, Geng and Yuan [24] considered an unbounded p-batch machine with family jobs to find all Pareto optimal points, with the aim of minimizing the makespan and maximum lateness. They proposed a dynamic programming algorithm to solve the study problem. He et al. [9] studied the single bounded serial-batching machine problem, where the objective is to minimize jointly the makespan and maximum lateness. They showed that the problem

could be solved in $O(n^6)$. Moreover, they presented an $O(n^3)$- time algorithm to find all Pareto optimal solutions where processing times and deadlines are agreeable.

Sabouni et al. [52] proposed an optimal method for the problem of scheduling jobs on a single batch processing machine, with the aim of minimizing the makespan and maximum lateness. The researchers showed that the proposed method is optimal when the group with the maximum lateness has the same processing times. Wan et al. [43] studied the problem of Pareto optimization on a single machine with two competing agents and linear non-increasing deterioration to find all Pareto optimal points. They showed that the problem can be solved in polynomial time and all the optimal points of Pareto are polynomial.

In [42], a polynomial-time algorithm has been proposed by Wan et al., to find all Pareto optimal solutions for scheduling two-agent on a single machine, with the aim of minimizing the number of tardy jobs and the maximum cost. Bazgan et al. [8] studied the Pareto sets for a minimal size, which approximates the accuracy of $\varepsilon$. They proposed a three-approximation algorithm for two-objective. Moreover, they proposed a study of the performance of the greedy algorithm for three objective cases when the points were explicitly given in the input. They also showed that the three objective cases are NP-hard.

The method of multi-objective mathematical programming (which is suitable for general multi-objective integer programming problems) have been used by Florios and Mavrotas [22], to produce all Pareto optimal solutions for the problems of a multi-objective traveling salesman and set covering. The researchers showed that the performance of the algorithm is slightly better than those of the literature. Moreover, they showed that their results can be helpful for other multi-objective mathematical programming methods or even multi-objective metaheuristics. Feng et al. [50] considered a two-agent scheduling problem on an unbounded parallel-batching machine. They proposed a polynomial-time algorithm to obtain all Pareto optimal solutions. Lazarev et al. [1] considered a single machine scheduling problem with equal processing times, with the aim to minimize the maximum lateness and makespan. They proposed a polynomial algorithm to find all Pareto solutions.

For the bicriteria scheduling problem, He et al. [12] showed that the problem of minimizing maximum cost and makespan is solvable in $O(n^5)$ time. Noteworthy, they proposed a polynomial-time algorithm to find all Pareto optimal solutions. Moreover, He et al. [13] showed that the bicriteria batching problem of minimizing maximum cost and makespan can be solvable in $O(n^3)$ time. The scheduling problem on a series-batching machine has been considered in [11], with the aim of minimizing the makespan and the total completion time. They proposed a dynamic programming algorithm and showed that all Pareto optimal solutions could be found in $O(n^2)$.

The multi-objective (bi-criteria) scheduling problem on a parallel-batching machine has been studied by He et al. [10], with the aim of minimizing the maximum lateness and the makespan. They presented a polynomial-time algorithm to find all Pareto optimal solutions. Allahverdi and Aldowaisan [6] considered the no-wait flowshop scheduling problem with bicriteria of makespan or maximum lateness. They proposed a dominance relation and a branch-and-bound algorithm. Moreover, they showed that these algorithms are very effective.

From the presented state-of-the-art, we can conclude that the multi-objective problems have attracted numerous researchers from around the world to achieve various objectives. In Chapter 4 and 5, we will introduce our contribution to the design of approximation methods to solve a multi-objective problem to minimize the maximum lateness and makespan. It should be noted that, to the best of our knowledge, the multi-objective problem we will consider in these chapters has not been studied before in the literature. For this reason, the design of efficient methods for this problem is a new attempt.

# Chapter 3

# Maximum Lateness Minimization on Two-Parallel Machine with a Non-availability Interval[1]

*In this chapter, we deal with the two-parallel machine scheduling problem with a non-availability interval. We aim to minimize the maximum lateness when every job has a positive tail. We show that the problem has a constant polynomial approximation algorithm. We present a dynamic programming algorithm and we show that the problem has an FPTAS (Fully Polynomial Time Approximation Algorithm). The proposed FPTAS has a strongly polynomial running time. Finally, we present some numerical experiments and we analyze the obtained results.*

## 3.1 Introduction

This chapter addresses the two-parallel machine scheduling problem with a non-availability interval. The objective is to minimize the maximum lateness. The problem can be formulated as follows. A set $J$ of $n$ jobs should be performed on two identical parallel machines. Each job $j \in J$ has a processing time $p_j$, and delivery time $q_j$. Each machine cannot carry out more than one job at a given time. The machines are available at time $t = 0$, which can process at most one job at a time (note that the first machine is available until time $T_1$ after that machine cannot process any job). The problem is to identify a sequence of jobs, that

[1]This work was published in IEEE / CoDIT18, with the best paper award at this conference.
Gais ALHADI, Imed KACEM, Pierre LAROCHE, Izzeldin M. OSMAN: Maximum Lateness Minimization on Two-Parallel Machine with a Non-availability Interval. CoDIT 2018: International Conference on Control, Decision and Information Technologies, April 10-13, 2018, pp. 757-762, DOI: 10.1109/CoDIT.2018.8394831.

minimizes the maximum lateness criterion: $L_{max} = max_{(1\leq j\leq n)}\{C_j(s) + q_j\}$ where $C_j(s)$ the completion time of job $j$ in sequence $S$. We assume that all data are integers and that jobs are sorted in non-increasing order of their delivery times $q_1 \geq q_2 \geq \ldots \geq q_n$ (i.e., the same order as Jackson's order). In the remainder of this chapter, the studied problem is denoted by $\pi$.

Numerous scheduling problems with non-availability constraints have recently attracted several researchers and have been widely studied in the literature. For instance, Kacem and Kellerer [5] studied the problem of the maximum lateness minimization on a single machine with an unexpected machine non-availability interval. They proposed a $(1+\sqrt{2}/2)$-approximation algorithm capable to solve the problem with delivery times but no release dates. This ratio is tight for the proposed algorithm and allows us to establish a precise window for the best possible ratio, which belongs to $[3/2, 1+\sqrt{2}/2]$. Kacem et al. [2] studied the two-parallel machines where one of the machines has one non-availability interval, with the aim of minimizing the weighted completion time. They formulated a DP algorithm and they showed that the problem admits an FPTAS that runs in strongly polynomial time. The problem of minimizing the weighted flow-time on a single machine with a fixed non-availability interval has been studied in [11] by Kacem who showed that there is a $2-$approximation algorithm, which can be implemented in $O(n^2)$.

Furthermore, the problems without a non-availability constraints to minimize the maximum lateness have attracted many researchers and have studied extensively in literature. For instance, Chiang et al. [10] proposed a local search-based heuristic to minimize maximum lateness on identical parallel batch machine scheduling problem. They compared the proposed heuristic with benchmark approach, and showed that their approach is better than the benchmark approach to most of the problem instances. An efficient adaptive repulsive particle swarm optimization algorithm has been proposed by Qiu et al. [9] to solve the permutation flow shop scheduling problem with the aim of minimizing the makespan and maximum lateness. They proposed a heuristic rule to present the discrete job permutation for the permutation flow shop scheduling problem.

Also, the problems of minimizing the makespan have been widely studied in the literature. For instance, Nguyen and Bao [6] proposed a modification Genetic Algorithm (GA) for the mixed shop scheduling problem to minimize the makespan. In the study problem they used a sample instances generated under the constraints of shop scheduling problems, and shown that the proposed GA provides an efficient solution. Cui and Lu [3] studied the problem of minimizing the makespan on a single machine with flexible periodic preventive maintenance and release dates. They developed two algorithms Branch and Bound algorithm (BB) to solve small and medium-sized problems, and Earliest Release Date (ERD-LPT) which have a high

degree of accurately and efficiently. Furthermore, they proved that the improvement of the integration between production scheduling and Preventive Maintenance (PM) is significant compared with the First-in-First-out (FIFO) rule.

In addition, a number of different approximation algorithms have been proposed, for example, Ma et al. [7] designed a new approximation algorithm for aircraft arrival sequencing and scheduling problem. They showed that the new algorithm have a much better performance than ant colony (AC) algorithm and CPLEX, especially when the number of aircraft types is not too many. Mitavskiy and He [8] designed a polynomial time approximation scheme for a single machine scheduling problem without precedence constraints using a hybrid evolutionary algorithm .They presented mathematical ideas and techniques that can be used to analyze many other hybrid evolutionary algorithms. The single machine scheduling problem and the parallel machine scheduling problem has been addressed by Rajkanth et al. [4] to minimize the completion time. They proposed two heuristics to solve the two problems. Furthermore, they proposed a genetic algorithm which have a good upper bound to evaluate the performance of the two heuristics. They also presented an extensive computation evaluation of the proposed heuristics for the two studied problems.

The organization of this chapter is as follows: the next Section 3.2 presents the constant approximation heuristics. Section 3.3 provides the description and the analysis of the dynamic programming algorithm and the FPTAS. In section 3.4, we demonstrate how to implement our Dynamic Programming and FPTAS algorithms on a small example. Section 3.5 presents the experimental results and the last Section 3.6 concludes the chapter.

## 3.2 Constant Approximation Heuristics

In this section, we consider two constructive heuristics in the worst-case.

### 3.2.1 The first heuristic

The first heuristic $H'$ is to put all the available jobs as soon as possible on the second machine (not necessary according to the Jackson's order).

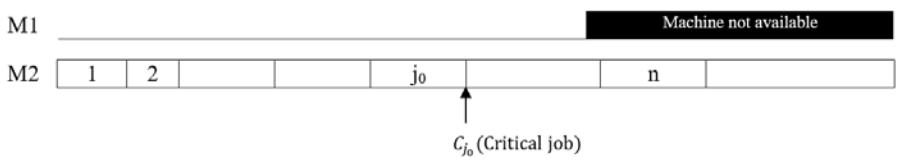



Fig. 3.1 Illustration of the critical job

**Lemma 3.2.1.1.** *Any assignment of jobs to the machines is a constant approximation with a standard ratio no more than* 3.

*Proof.* We define $P = \sum_{j=1}^{n} p_j$, and $OPT$ the optimal value of the maximum lateness.
Now, let $j_0$ be the critical job in the sequence given by $H'$ such that $C_{j_0} + q_{j_0} = L_{max}(H')$ as it is shown in Fig. 3.1. Clearly, we have:

$$L_{max}(H') \leq P + q_{j_0} \tag{3.2.1.1}$$

Obviously, we have:

$$OPT \geq P/2 \tag{3.2.1.2}$$

and

$$OPT \geq q_{j_0} \tag{3.2.1.3}$$

Therefore, it can be deduced that $L_{max}(H') \leq 2.(P/2) + q_{j_0} \leq 2.OPT + OPT = 3.OPT$ □

### 3.2.2 The second heuristic

The second heuristic $H$ puts all the jobs as soon as possible on the second machine according to the Jackson's order.
We use again Fig. 3.1 to refer to the sequence given by $H$. Here, we define again the critical job such that $C_{j_0} + q_{j_0} = L_{max}(H)$.

**Lemma 3.2.2.1.** *Any assignment of jobs to the machines according to the Jackson's order is a constant approximation with a standard ratio no more than* 2.

*Proof.* Clearly, we have:

$$L_{max}(H) = \sum_{i=1}^{j_0} p_i + q_{j_0} \tag{3.2.2.1}$$

Moreover, it can be established by considering the relaxed problem of scheduling the jobs in $\{1, 2, ..., j_0\}$ on two identical machines that the following relation holds:

$$OPT \geq (\sum_{i=1}^{j_0} p_i)/2 + q_{j_0} \tag{3.2.2.2}$$

Indeed, there is at least one job from the subset $\{1,2,...,j_0\}$ for which the completion time will be greater than $(\sum_{i=1}^{j_0} p_i)/2$. Consequently, we deduce the following relation:

$$L_{max}(H) = \sum_{i=1}^{j_0} p_i + q_{j_0} \leq OPT + (\sum_{i=1}^{j_0} p_i)/2 \leq 2.OPT \tag{3.2.2.3}$$

□

## 3.3 Fully Polynomial-Time Approximation Scheme

### 3.3.1 Dynamic programming algorithm

The following dynamic programming algorithm $A$, can be applied for solving this problem optimally. This algorithm $A$ generates iteratively some sets of states. At every iteration $j$, a set $Q_j$ of states is generated $(0 \leq j \leq n)$. Each state $[t,f]$ in $Q_j$ can be associated to a feasible schedule for the $j$ first jobs. Here, variable $t$ denotes the completion time of the last job scheduled on the first machine before $T_1$, and variable $f$ is the maximum lateness of the corresponding schedule.

**Algorithm** $A$

1. Set $Q_0 = \{[0,0]\}$.
2. For $j \in \{1,2,3,...,n\}$,
   - $Q_j = \{\}$
   - For each $[t,f]$ in $Q_{j-1}$
     (a) Put $[t, max\{f, \sum_{i=1}^{j} p_i - t + q_j\}]$ in $Q_j$
     (b) Put $[t+p_j, max\{f, t+p_j+q_j\}]$ in $Q_j$ (if $t+p_j \leq T_1$)
   - Remove $Q_{j-1}$
3. Return $L^*_{max} = min_{[t,f]\in Q_n}\{f\}$

The standard version of this dynamic programming has an exponential complexity since it needs $O(2^n)$ time. Nevertheless, this complexity can be reduced to $O(n.T_1)$ by keeping only one state having the smallest value of $f$ for every value $t$ in $Q_j$ (for every iteration $j \in \{1,2,...,n\}$). Therefore, $A$ can be reduced to a pseudo-polynomial algorithm, which proves at the same time that the problem is NP-hard only in the ordinary sense.

### 3.3.2 FPTAS

Here, we are interested in the existence of an FPTAS for the studied problem. In the proposed FPTAS we remove a special part of the states generated by Algorithm $A$. This process is done in order to create an approximate solution enough close to the optimum. The new algorithm is called $A^{\varepsilon}$ ($\varepsilon > 0$) It will be the proposed FPTAS, which uses similar ideas as in Kacem et al. (see for instance [1]-[2]).
By comparing the steps of algorithms $A^{\varepsilon}$ and $A$, we produce the analysis at the worst-case for the proposed FPTAS.
Given an arbitrary $\varepsilon > 0$, we define:

$$\gamma_1 = \frac{2}{\varepsilon}, \tag{3.3.2.1}$$

$$\gamma_2 = \frac{4.n}{\varepsilon} \tag{3.3.2.2}$$

We assume that $\gamma_1$ and $\gamma_2$ are integers (otherwise, we round up to the next integer values). Now, we define the following additional parameters, which we will use in the FPTAS:

$$\delta_1 = \frac{T_1}{\gamma_1}, \tag{3.3.2.3}$$

$$\delta_2 = \frac{L_{max}(H)}{\gamma_2} \tag{3.3.2.4}$$

We divide the interval $[0, T_1]$ (respectively, the interval $[0, L_{max}(H)]$) into $\gamma_1$ equal sub intervals $X_h = [(h-1)\delta_1, h\delta_1]_{1 \leq h \leq \gamma_1}$ of length $\delta_1$ (respectively, into $\gamma_2$ equal sub intervals $Y_k = [(k-1)\delta_2, k\delta_2]_{1 \leq k \leq \gamma_2}$ of length $\delta_2$ ). The algorithm $A^{\varepsilon}$ described below produces the reduced sets $Q_j^{\#}$ instead of sets $Q_j$.

**Algorithm** $A^{\varepsilon}$

1. Set $Q_0^{\#} = \{[0,0]\}$.
2. For $j \in \{1,2,3,...,n\}$,
   - $Q_j^{\#} = \{\}$
   - For each $[t,f]$ in $Q_{j-1}^{\#}$
     (a) Put $[t, max\{f, \sum_{i=1}^{j} p_i - t + q_j\}]$ in $Q_j^{\#}$
     (b) Put $[t + p_j, max\{f, t + p_j + q_j\}]$ in $Q_j^{\#}$ (if $t + p_j \leq T_1$)

- Remove $Q^{\#}_{j-1}$
- Let $[t,f]_{h,k}$ and $[s,g]_{h,k}$ be the states in $Q^{\#}_j$ such that $t,s \in X_h$ and $f,g \in Y_k$ where $t$ (respectively, $s$) is the smallest value (respectively, the greatest value) in subinterval $X_h$. Set $Q^{\#}_j = \{[t,f]_{h,k},[s,g]_{h,k} | 1 \leq h \leq \gamma_1, 1 \leq k \leq \gamma_2\}$.

3. Return $L^{A^{\varepsilon}}_{max} = min_{[t,f] \in Q^{\#}_n}\{f\}$

**Lemma 3.3.2.1.** *For every state $[t,f] \in Q_j$ there exists at least one approximate state $[t^{\#},f^{\#}] \in Q^{\#}_j$ such that:*

$$t - \delta_1 \leq t^{\#} \leq t \tag{3.3.2.5}$$

*and*

$$f^{\#} \leq f + \delta_1 + j\delta_2 \tag{3.3.2.6}$$

*Proof.* By induction on $j$.
First, for $j = 0$ we have $Q^{\#}_j = Q_j$ and the statement is trivial. Then, assume that the lemma is valid up to level $j-1$.
Now, let us consider an arbitrary state $[t,f] \in Q_j$. Algorithm $A$ introduces this state into $Q_j$ when job $j$ is considered to complete a certain partial solution for the $j-1$ first jobs. Let $[t^{'},f^{'}]$ be the above feasible state. Two scenarios can be considered:
Either $[t,f] = [t^{'}, max\{f^{'}, \sum_{i=1}^{j} p_i - t^{'} + q_j\}]$ or $[t,f] = [t^{'} + p_j, max\{f^{'}, t^{'} + p_j + q_j\}]$ must hold. We will demonstrate the result for level $j$ in the two cases.

Case 1 :$[t,f] = [t^{'}, max\{f^{'}, \sum_{i=1}^{j} p_i - t^{'} + q_j\}]$

Since $[t^{'},f^{'}] \in Q_{j-1}$, there exists $[t^{\#\prime},f^{\#\prime}] \in Q^{\#}_{j-1}$, such that $t^{'} - \delta_1 \leq t^{\#\prime} \leq t^{'}$ and $f^{\#\prime} \leq f^{'} + \delta_1 + (j-1)\delta_2$. Consequently, the state $e = [t^{\#\prime}, max\{f^{\#\prime}, \sum_{i=1}^{j} p_i - t^{\#\prime} + q_j\}]$ is created at iteration $j$ by algorithm $A^{\varepsilon}$ and it is feasible. But, it may be eliminated during the reduction of the state subset. Let $[\lambda,\mu]$ and $[\alpha,\beta]$ be the states replacing state $e$ in $Q^{\#}_j$ ($\lambda \leq \alpha$). This means that the states $[\lambda,\mu]$ and $[\alpha,\beta]$ belong to the same "box" as the state $e$. Therefore, we have:

$$\lambda \leq t^{\#\prime} \leq \alpha, \tag{3.3.2.7}$$

Now we will consider two sub-cases to prove the existence of a "close" state in $Q^{\#}_j$ for approximating $[t,f]$. These sub-cases are: $t' \leq \alpha$ or $t' > \alpha$.

In the first sub-case ($t' \leq \alpha$), we will take $[\lambda, \mu]$ as a close state for $[t, f]$. Indeed, we can observe that $\lambda \leq t^{\#'} \leq t' = t$ and that $\lambda \geq \alpha - \delta_1 \geq t' - \delta_1 = t - \delta_1$. Moreover,

$$\begin{aligned}
\mu &\leq max\{f^{\#'}, \sum_{i=1}^{j} p_i - t^{\#'} + q_j\} + \delta_2 \\
&\leq max\{f^{'} + \delta_1 + (j-1)\delta_2, \sum_{i=1}^{j} p_i - t^{\#'} + q_j\} + \delta_2 \\
&\leq max\{f^{'} + \delta_1 + (j-1)\delta_2, \sum_{i=1}^{j} p_i - t^{'} + \delta_1 + q_j\} + \delta_2 \\
&\leq max\{f^{'}, \sum_{i=1}^{j} p_i - t^{'} + q_j\} + \delta_1 + j\delta_2 \\
&= f + \delta_1 + j\delta_2 \qquad (3.3.2.8)
\end{aligned}$$

Consequently, $[\lambda, \mu]$ is an approximate state in $Q_j^{\#}$ verifying the two conditions.

In the second sub-case ($t' > \alpha$), we will take $[\alpha, \beta]$ as a close state for $[t, f]$. Indeed, we can remark that $\alpha < t' = t$ and that $\alpha \geq t^{\#'} \geq t' - \delta_1 = t - \delta_1$. Moreover,

$$\beta \leq max\{f^{\#'}, \sum_{i=1}^{j} p_i - t^{\#'} + q_j\} + \delta_2 \qquad (3.3.2.9)$$

By a similar reasoning as in the first sub-case for bounding $\mu$, we can deduce the following relation for $\beta$:

$$\beta \leq f + \delta_1 + j\delta_2$$

Therefore, we can verify that $[\alpha, \beta]$ respects all the required conditions.

Case 2 :$[t, f] = [t^{'} + p_j, max\{f^{'}, t^{'} + p_j + q_j\}]$

Since $[t^{'}, f^{'}] \in Q_{j-1}$, there exists $[t^{\#\prime}, f^{\#\prime}] \in Q_{j-1}^{\#}$, such that $t^{'} - \delta_1 \leq t^{\#'} \leq t^{'}$ and $f^{\#\prime} \leq f^{'} + \delta_1 + (j-1)\delta_2$. Consequently, the state $e' = [t^{\#'} + p_j, max\{f^{\#'}, t^{\#'} + p_j + q_j\}]$ is created by algorithm $A^{\varepsilon}$ at iteration $j$. But, it may be eliminated when we reduce the state subset. Again, we have to consider two states $[\lambda', \mu']$ and $[\alpha', \beta']$, which are used to replace $e'$ at iteration $j$. Similarly to the proof in Case 1, we have to distinguish two sub-cases ($\alpha' \leq t' + p_j$

or $\alpha' > t' + p_j$ ). For these two sub-cases, we can demonstrate that one of the states $[\lambda', \mu']$ and $[\alpha', \beta']$ can verify the required conditions. The detail is left to the reader.
In conclusion, the statement is also verified for level $j$ in the two considered cases, and this completes the proof. □

We are now ready to establish the existence of an FPTAS with a strongly polynomial time for the studied problem.

**Theorem 3.3.2.2.** *Algorithm $A^{\varepsilon}$ is an FPTAS for the problem $\pi$ and it can be implemented in $O(\frac{n^2}{\varepsilon^2})$.*

*Proof.* Let $[t^*, f^*] \in Q_n$ such that $OPT = f^*$. From the previous lemma, it can be deduced that there exists a feasible solution $S^{\#}$ of the problem $\pi$ associated to a state $[t^{\#}, f^{\#}] \in Q_n^{\#}$ such that:

$$f^{\#} \leq f^* + \delta_1 + n.\delta_2 \tag{3.3.2.10}$$

$$= OPT + \varepsilon.T_1/2 + n.\varepsilon.L_{max}(H)/4.n \tag{3.3.2.11}$$

$$\leq (1+\varepsilon).OPT \tag{3.3.2.12}$$

Regarding the time complexity, the algorithm generates at each iteration a number of states in $O(\gamma_1\gamma_2)$. Indeed, we keep at most $2\gamma_1\gamma_2$ states at every iteration $j$, leading to a maximum of possible candidates equal to $4\gamma_1\gamma_2$ for the next iteration $j+1$, at which only $2\gamma_1\gamma_2$ states can stand again. In sum, this leads to a global time complexity in $O(\frac{n^2}{\varepsilon^2})$. □

## 3.4 Illustrative Example

In this section, we demonstrate how to implement our Dynamic Programming and FPTAS algorithms on a small example. The instance we use is described in the Table 3.1: We have six jobs; each job has a processing time $p_j$ and its delivery time $q_j$. We will take $T_1 = P/2 = 118$ (where $P = \sum_{j=1}^{6} p_j$) and $\varepsilon = 0.9$. Notes that the jobs have already been sorted according to the non-increasing order of their delivery times.

Table 3.1 An instance with six jobs

| Jobs($J_j$) | Processing time ($p_j$) | Delivery time ($q_j$) |
|---|---|---|
| 1 | 38 | 9 |
| 2 | 32 | 8 |
| 3 | 48 | 6 |
| 4 | 41 | 5 |
| 5 | 35 | 3 |
| 6 | 42 | 2 |

### 3.4.1 Dynamic programming

In this algorithm (as described in Section 3.3.1), a solution is given by building partial solutions, stored in $Q_j$, consisting of scheduling the six jobs $(0 \leq j \leq 6)$. The final solution $L^*_{max}$ is given by keeping only one state having the smallest value of $f$ for every value $[t, f]$ in $Q_6$. Here, variable $t$ denotes the completion time of the last job scheduled on the first machine before $T_1$, and variable $f$ is the maximum lateness value obtained for the associated partial solution.

**First Schedule.**

The first job $(38,9)$ is assigned to machines $M_1$ and $M_2$, leading to two partial solutions in $Q_1 = \{(0,47),(38,47)\}$.

with $M_2$: (0,47)

with $M_1$: (38,47)

**Second Step.**

The second job $(32,8)$ is scheduled alternatively on machines $M_1$ and $M_2$, completing each partial scheduling of $Q_1$, as follows:

**From (38,47):**

with $M_2$: (38,47)

with $M_1$: (70,78)

**From (0,47):**

with $M_2$: (0,78)

with $M_1$: (32,47)

Hence, this gives new partial solutions in $Q_2 = \{(0,78),(32,47),(70,78),(38,47)\}$.

**Third Step.**

The third job $(48,6)$ is scheduled alternatively on machines $M_1$ and $M_2$, completing each partial scheduling of $Q_2$, as follows:

**From (0,78):**

with $M_2$ : (0,124)

with $M_1$ : (48,78)

**From (32,47):**

with $M_2$ : (32,92)

with $M_1$ : (80,86)

**From (70,78):**

with $M_2$ : (70,78)

with $M_1$ : (118,124)

**From (38,47):**

with $M_2$ : (38,86)

with $M_1$ : (86,92)

Hence, this gives new partial solutions in $Q_3 = \{(80,86),(118,124),(32,92),(0,124),(48,78),(86,92),(70,78),(38,86)\}$.

**Fourth Step.**

In this step, the fourth job $(41,5)$ is scheduled alternatively on machines $M_1$ and $M_2$, completing each partial scheduling of $Q_3$, as follows:

**From (80,86):**

with $M_2$ : (80,86)

with $M_1$ : no possibility

**From (118,124):**

with $M_2$ : (118,124)

with $M_1$ : no possibility

**From (32,92):**
with $M_2$ : (32,132)
with $M_1$ : (73,92)

**From (0,124):**
with $M_2$ : (0,164)
with $M_1$ : (41,124)

**From (48,78):**
with $M_2$ : (48,116)
with $M_1$ : (89,94)

**From (86,92):**
with $M_2$ : (86,92)
with $M_1$ : no possibility

**From (70,78):**
with $M_2$ : (70,94)
with $M_1$ : (111,116)

**From (38,86):**
with $M_2$ : (38,126)
with $M_1$ : (79,86)

Thus, the new partial solutions in $Q_4 = \{(86,92),(38,126),(89,94),(48,116),(80,86),(32,132),(41,124),(70,94),(0,164),(79,86),(111,116),(118,124),(73,92)\}$.

**Fifth Step.**

The fifth job $(35,3)$ is scheduled alternatively on machines $M_1$ and $M_2$, completing each partial scheduling of $Q_4$, as follows:

**From (86,92):**
with $M_2$ : (86,111)
with $M_1$ : no possibility

**From (38,126):**
with $M_2$ : (38,159)

with $M_1$ : (73,126)

**From (89,94):**
with $M_2$ : (89,108)
with $M_1$ : no possibility

**From (48,116):**
with $M_2$ : (48,149)
with $M_1$ : (83,116)

**From (80,86):**
with $M_2$ : (80,117)
with $M_1$ : (115,118)

**From (32,132):**
with $M_2$ : (32,165)
with $M_1$ : (67,132)

**From (41,124):**
with $M_2$ : (41,156)
with $M_1$ : (76,124)

**From (70,94):**
with $M_2$ : (70,127)
with $M_1$ : (105,108)

**From (0,164):**
with $M_2$ : (0,197)
with $M_1$ : (35,164)

**From (79,86):**
with $M_2$ : (79,118)
with $M_1$ : (114,117)

**From (111,116):**
with $M_2$ : (111,116)

with $M_1$ : no possibility

**From (118,124):**
with $M_2$ : (118,124)
with $M_1$ : no possibility

**From (73,92):**
with $M_2$ : (73,124)
with $M_1$ : (108,111)

Hence, the partial solutions in $Q_5 = \{(80,117),(114,117),(108,111),(115,118),(118,124),(111,116),(73,124),(48,149),(105,108),(41,156),(83,116),(76,124),(70,127),(67,132),(38,159),(79,118),(0,197),(35,164),(86,111),(89,108),(32,165)\}$.

**Sixth Step.**

In this step, the last job $(42,2)$ is scheduled alternatively on machines $M_1$ and $M_2$, completing each partial scheduling of $Q_5$, as follows:

**From (80,117):**
with $M_2$ : (80,158)
with $M_1$ : no possibility

**From (114,117):**
with $M_2$ : (114,124)
with $M_1$ : no possibility

**From (108,111):**
with $M_2$ : (108,130)
with $M_1$ : no possibility

**From (115,118):**
with $M_2$ : (115,123)
with $M_1$ : no possibility

**From (118,124):**
with $M_2$ : (118,124)
with $M_1$ : no possibility

**From (111,116):**
with $M_2$ : (111,127)
with $M_1$ : no possibility

**From (73,124):**
with $M_2$ : (73,165)
with $M_1$ : (115,124)

**From (48,149):**
with $M_2$ : (48,190)
with $M_1$ : (90,149)

**From (105,108):**
with $M_2$ : (105,133)
with $M_1$ : no possibility

**From (41,156):**
with $M_2$ : (41,197)
with $M_1$ : (83,156)

**From (83,116):**
with $M_2$ : (83,155)
with $M_1$ : no possibility

**From (76,124):**
with $M_2$ : (76,162)
with $M_1$ : (118,124)

**From (70,127):**
with $M_2$ : (70,168)
with $M_1$ : (112,127)

**From (67,132):**
with $M_2$ : (67,171)
with $M_1$ : (109,132)

**From (38,159):**
with $M_2$ : (38,200)
with $M_1$ : (80,159)

**From (79,118):**
with $M_2$ : (79,159)
with $M_1$ : no possibility

**From (0,197):**
with $M_2$ : (0,238)
with $M_1$ : (42,197)

**From (35,164):**
with $M_2$ : (35,203)
with $M_1$ : (77,164)

**From (86,111):**
with $M_2$ : (86,152)
with $M_1$ : no possibility

**From (89,108):**
with $M_2$ : (89,149)
with $M_1$ : no possibility

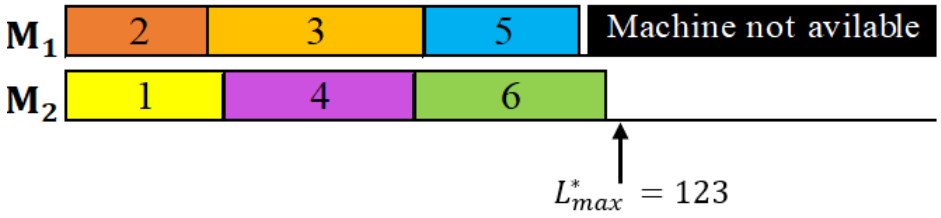
**From (32,165):**
with $M_2$ : (32,206)
with $M_1$ : (74,165)

Thus, the new partial solutions in $Q_6 = \{(77,164),(118,124),(38,200),(42,197),(48,190),(109,132),(73,165),(67,171),(32,206),(70,168),(74,165),(89,149),(0,238),(111,127),(112,127),(86,152),(115,123),(80,158),(83,155),(35,203),(90,149),(76,162),(105,133),(79,159),(114,124),(41,197),(108,130)\}$

**Seventh (Final) Step.**

After having scheduled the last job (job 6), the last set of partial solutions is composed of 27 states. To obtain the final solution $L^*_{max}$, for each value in $Q_6$ we only keep one state

which has the smallest value of $f$ that is $(115,123)$. Therefore, $L^*_{max} = 123$, as shown in the following figure.




### 3.4.2 FPTAS

In the FPTAS algorithm (as described in Section 3.3.2), we remove a special part of the states generated by DP algorithm, in order to produce an approximation solution instead of the optimal solution.
In this example, the search space is divided into (intervals) $\gamma 1 = 3$ of length $\delta 1 = 53.1$ and $\gamma 2 = 27$ of length $\delta 2 = 8.9$.
Once again, the jobs have to be sorted according to the non-increasing order of their delivery times, and the partial solutions $Q_j$ are composed of states $[t,f]$. Thus, after each step, for each interval we keep only two states which have the smallest and greatest value of $t$. Finally, the final solution is given by keeping only one state having the smallest value of $f$ for every value $[t,f]$ in $Q_6$.

**First Schedule. (Step.) to the Third Step.**

In these steps, the results are the same as DP. Clearly, the partial solutions sets will be as follows:
$Q_1 = \{(0,47),(38,47)\}$.
$Q_2 = \{(0,78),(32,47),(70,78),(38,47)\}$.
$Q_3 = \{(80,86),(118,124),(32,92),(0,124),(48,78),(86,92),(70,78),(38,86)\}$.

**Fourth Step.**

In this step, the fourth job $(41,5)$ is scheduled alternatively on machines $M_1$ and $M_2$ , completing each partial scheduling of $Q_3$, as follows:

**From (48,78):**

with $M_2$ : (48,116)

with $M_1$ : (89,94)

**From (32,92):**
with $M_2$ : (32,132)
with $M_1$ : (73,92)

**From (86,92):**
with $M_2$ : (86,92)
with $M_1$ : no possibility

**From (0,124):**
with $M_2$ : (0,164)
with $M_1$ : (41,124)

**From (70,78):**
with $M_2$ : (70,94)
with $M_1$ : (111,116)

**From (80,86):**
with $M_2$ : (80,86)
with $M_1$ : no possibility

**From (118,124):**
with $M_2$ : (118,124)
with $M_1$ : no possibility

**From (38,86):**
with $M_2$ : (38,126)
with $M_1$ : (79,86)

It is worth-mentioning that the states (70,94), (73,92), (86,92) and (89,94) in the same box, so we will keep only the two states (70,94) and (89,94), which have the smallest and greatest value of $t$. Hence, the partial solutions in $Q_4 = \{(41,124),(111,116),(0,164),(48,116),(118,124),(38,126),(80,86),(79,86),(89,94),(32,132),(70,94)\}$

**Fifth Step.**

The fifth job $(35,3)$ is scheduled alternatively on machines $M_1$ and $M_2$ , completing each partial scheduling of $Q_4$, as follows:

**From (41,124):**
with $M_2$ : (41,156)
with $M_1$ : (76,124)

**From (111,116):**
with $M_2$ : (111,116)
with $M_1$ : no possibility

**From (0,164):**
with $M_2$ : (0,197)
with $M_1$ : (35,164)

**From (48,116):**
with $M_2$ : (48,149)
with $M_1$ : (83,116)

**From (118,124):**
with $M_2$ : (118,124)
with $M_1$ : no possibility

**From (38,126):**
with $M_2$ : (38,159)
with $M_1$ : (73,126)

**From (80,86):**
with $M_2$ : (80,117)
with $M_1$ : (115,118)

**From (79,86):**
with $M_2$ : (79,118)
with $M_1$ : (114,117)

**From (89,94):**
with $M_2$ : (89,108)
with $M_1$ : no possibility

**From (32,132):**
with $M_2$ : (32,165)
with $M_1$ : (67,132)

**From (70,94):**
with $M_2$ : (70,127)
with $M_1$ : (105,108)

At this step, there are 4 states $\{(70,127),(79,118),(89,108),(115,118)\}$ have been removed, because,
(70,127) in the same box as (67,132) and (73,126),
(79,118) in the same box as (76,124) and (80,117),
(89,108) in the same box as (83,116) and (105,108),
(115,118) in the same box as (114,117) and (118,124).

Thus, the partial solutions in $Q_5 = \{(114,117),(76,124),(35,164),(105,108),(118,124),(0,197),(83,116),(32,165),(111,116),(67,132),(38,159),(41,156),(48,149),(80,117),(73,126)\}$.

**Sixth Step.**

In this step, the sixth job $(42,2)$ is scheduled alternatively on machines $M_1$ and $M_2$ , completing each partial scheduling of $Q_5$, as follows:

**From (114,117):**
with $M_2$ : (114,124)
with $M_1$ : no possibility

**From (76,124):**
with $M_2$ : (76,162)
with $M_1$ : (118,124)

**From (35,164):**
with $M_2$ : (35,203)
with $M_1$ : (77,164)

**From (105,108):**
with $M_2$ : (105,133)
with $M_1$ : no possibility

**From (118,124):**
with $M_2$ : (118,124)
with $M_1$ : no possibility

**From (0,197):**
with $M_2$ : (0,238)
with $M_1$ : (42,197)

**From (83,116):**
with $M_2$ : (83,155)
with $M_1$ : no possibility

**From (32,165):**
with $M_2$ : (32,206)
with $M_1$ : (74,165)

**From (111,116):**
with $M_2$ : (111,127)
with $M_1$ : no possibility

**From (67,132):**
with $M_2$ : (67,171)
with $M_1$ : (109,132)

**From (38,159):**
with $M_2$ : (38,200)
with $M_1$ : (80,159)

**From (41,156):**
with $M_2$ : (41,197)
with $M_1$ : (83,156)

**From (48,149):**
with $M_2$ : (48,190)
with $M_1$ : (90,149)

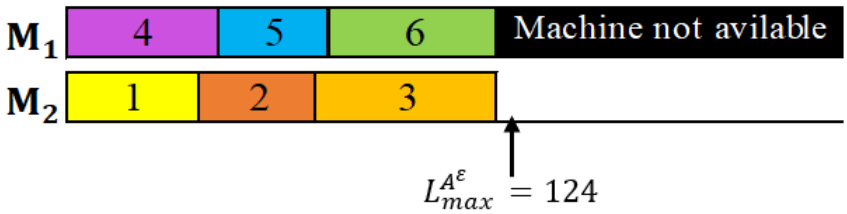
**From (80,117):**
with $M_2$ : (80,158)
with $M_1$ : no possibility

**From (73,126):**
with $M_2$ : (73,165)
with $M_1$ : (115,126)

At this step, the partial solutions in $Q_6 = \{(118,124),(90,149),(83,155),(105,133),(67,171),(73,165),(35,203),(109,132),(114,124),(0,238),(80,158),(42,197),(48,190),(115,126),(77,164),(32,206)\}$. Thanks to our FPTAS algorithm, the states (118,124),(76,162), (74,165), (111,127), (38,200), (80,159), (41,197) and (83,156) have not been added in $Q_6$.

**Seventh (Final) Step.**

After having scheduled the last job, the last set of partial solutions is composed of 16 states. To obtain the final solution, for each value in $Q_6$ we only keep one state which has the smallest value of $f$ that is (118,124). Therefore, $L_{max}^{A^\varepsilon} = 124$, as shown in the following figure.




## 3.5 Results

The following results have been obtained after testing the performance of the proposed algorithms. The code has been done in Java and the experiments were performed on an Intel(R) Core(TM)-i7 with 8GB RAM. The results have been tested with two values of $\varepsilon$: 0.9 and 0.3, and three values for non-availability intervals: $T_1 = P/2, P/3$ and $P/7$. To ensure the consistency of running times, each test has been run three times.

The remainder of this section is organized as follows. In Subsection 3.5.1, we will present the results of the First (small) instances. The results of the big instances are presented in Subsection 3.5.2.

### 3.5.1 First instances

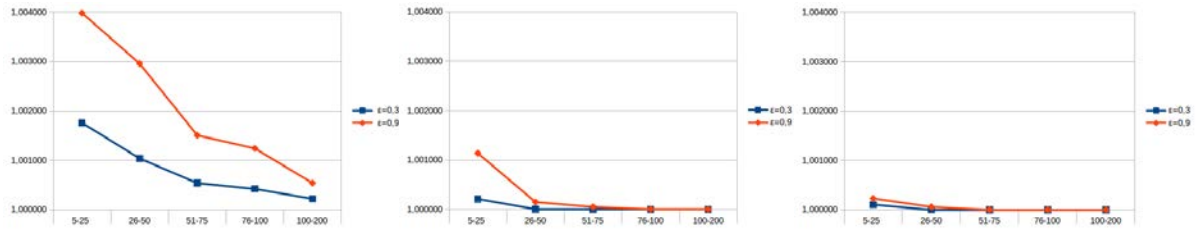
We have randomly generated five sets of instances, with different numbers of jobs and various processing and delivery times:

- number of jobs: from 5 to 25, 26 to 50, 51 to 75, 76 to 100 and 100 to 200;
- processing times : from 1 to 20, 1 to 100 and 1 to 500;
- delivery times : from 1 to 20, 1 to 100 and 1 to 500;

For each set of parameters, we have generated three different instances. That gave us 135 instances in each set of instances.
In this subsection, we present the results for our five sets of instances, from small instances (5-25 jobs) to bigger ones (100-200 jobs).

Fig.3.2 presents a comparison of FPTAS and Dynamic Programming. For three values of non-availability interval and two $\varepsilon$ values, each sub-figure shows the ratio $L_{max}^{A^\varepsilon}/L_{max}^{*}$ for our five sets of instances. It is no surprise that it is easier to obtain close to optimal solutions when the non-availability interval is bigger. Indeed, in these cases, the solutions space is limited: as soon as machine $M_1$ has reached non-availability, the only choice left is to assign the jobs to machine $M_2$.

This figure also shows that, regardless of the non-availability interval, FPTAS gives better results when the number of jobs is important. The average gap between FPTAS solutions and the optimal solutions is never more than 0.4%. Considering all the tests we have made, the biggest gap with the optimal solution was 3.2%. The good quality of FPTAS when dealing with big instances can be explained by the way the space search is decomposed, the number of cells given by $\gamma 2$ (see 3.3.2.2) being a function of $n$.




$T_1 = P/2$ $T_1 = P/3$ $T_1 = P/7$

Fig. 3.2 Quality of solutions for our instances with various $T_1$ and $\varepsilon$ values

We have also studied the influence of processing and delivery times. Our experiments showed that delivery time ranges have no influence, neither on the quality of FPTAS, nor on the computing times of our algorithms. At the opposite, Table 3.2 shows that instances composed of jobs with various processing times are more difficult to solve. Results are presented for $T_1 = P/3$ and $\varepsilon = 0.9$. It should be noted that the analysis would be the same whatever the considered parameters. This table shows the number of instances solved to the optimal by FPTAS and the ratio between the FPTAS solution and the optimal solution ($L^{A^\varepsilon}_{max}/L^*_{max}$). The results are presented for two sets of instances, short ones (from 5 to 25 jobs) and bigger ones (from 100 to 200 jobs).

Table 3.2 Quality of FPTAS solutions as a function of processing time ranges (for $T_1 = P/3$ and $\varepsilon = 0.9$)

| $p_i$ range | % of optimal solutions | $L^{A^\varepsilon}_{max}/L^*_{max}$ | DP time | FPTAS time |
|---|---|---|---|---|
| #jobs 5 to 25 | | | | |
| 1-20 | 100 | 1 | 0.3 | 0.15 |
| 1-100 | 80 | 1.0008 | 3 | 0.15 |
| 1-500 | 40 | 1.0013 | 37 | 0.15 |
| #jobs 100 to 200 | | | | |
| 1-20 | 100 | 1 | 145 | 21 |
| 1-100 | 100 | 1 | 3068 | 20 |
| 1-500 | 90.48 | 1.000006 | 177449 | 20 |

We can see that instances with jobs having a wide range of processing times are more difficult to solve to optimality. For processing times between 1 and 100, the FPTAS gives

the optimal solution for 80% (resp. 100%) of the small (resp. big) instances, whereas the optimality is not reached for 40% (resp. 90.48%) of the small (resp. big) instances. The ratios between FPTAS solutions and optimal solutions show that, even when the optimality is rarely reached, the gap remains very small.

This table also mentions the computing times, in milliseconds. We can see that the range of processing times has no influence on the FPTAS, whereas it is an important parameter for the Dynamic Programming algorithm. This is consistent w.r.t. the complexity results given in previous section.

More computing times are also given in Tables 3.3 and 3.4. They compare our Dynamic Programming algorithm and our FPTAS, considering two different values for $T_1$. As earlier, all values are in milliseconds. The Dynamic Programming algorithm is slower when the non-availability interval is decreasing and the number of states is growing, as expected with a complexity $O(n.T_1)$ (see Section 3.3.1). We can remark that computing times are not in a $2/7$ ratio, as expected. This is due to computational effort made to keep only one state for each $t$ value, which is not considered by the theoretical study.

As expected, FPTAS outperforms Dynamic Programming, especially with big values of $\varepsilon$ and small values of $T_1$. The computing times are consistent with the complexity $O(\frac{n^2}{\varepsilon^2})$.

Table 3.3 Average computing times (ms) for $T_1 = P/2$

| #jobs | DP | FPTAS | |
|---|---|---|---|
| | | $\varepsilon = 0.3$ | $\varepsilon = 0.9$ |
| 5-25 | 67 | 0.9 | 0.3 |
| 26-50 | 881 | 3 | 1 |
| 51-75 | 5903 | 10 | 3 |
| 76-100 | 21963 | 22 | 7 |
| 100-200 | 138431 | 74 | 23 |

Table 3.4 Average computing times (ms) for $T_1 = P/7$

| #jobs | DP | FPTAS | |
|---|---|---|---|
| | | $\varepsilon = 0.3$ | $\varepsilon = 0.9$ |
| 5-25 | 1.5 | 0.2 | 0.1 |
| 26-50 | 37 | 1.5 | 0.5 |
| 51-75 | 290 | 4 | 1.2 |
| 76-100 | 975 | 8 | 2.5 |
| 100-200 | 8118 | 28 | 10 |

### 3.5.2 Big instances

We also tested bigger instances, with 1000 jobs. We generated 12 instances, with 1-20 and 1-100 processing times. The results are consistent with the ones of smaller instances. Table 3.5 presents the quality of FPTAS and computing times for both Dynamic Programming and FPTAS. For these instances, as the computing times are quite big, they have been expressed in seconds.

Table 3.5 Results for big instances (times in seconds)

| $T_1$ | DP time | FPTAS | | | |
|---|---|---|---|---|---|
| | | $\varepsilon = 0.3$ | | $\varepsilon = 0.9$ | |
| | | time | $L_{max}^{A^\varepsilon}/L_{max}^*$ | time | $L_{max}^{A^\varepsilon}/L_{max}^*$ |
| $P/2$ | 1883 | 3.2 | 1.000007 | 1.6 | 1.00003 |
| $P/3$ | 980 | 3.5 | 1 | 1.2 | 1 |
| $P/7$ | 137 | 1.1 | 1 | 0.6 | 1 |

We can see that the FPTAS still managed to find very good solutions very quickly, as we expected after previous experiments.

## 3.6 Conclusion

In this chapter, we considered the two-parallel machine scheduling problem without release dates assumption, with the aim of minimizing the maximum lateness. In this problem, instead of allowing both machines to be continuously available we consider that the first machine is available for a specified period of time (available until time $T_1$). After that, this machine cannot process any job. We proposed an approximation heuristic which has the performance ratio of the worst case not more than $3/2$. We also have proved the existence of an FPTAS for the problem. The results of the numerical experiments showed that the proposed algorithms for the considered problem are very efficient, especially for big instances composed of a lot of jobs. Moreover, the proposed FPTAS which has a strongly polynomial running time has been shown to be very efficient. We have also shown the importance of the values of processing times, non-availability interval, and $\varepsilon$ value.
In our future works, we look forward to extend these results to other variants of this problem, considering for instances a fixed number of machines (more than two). We will also investigate some multi-objective versions of this problem, trying to optimize not only the maximum lateness, but also other criteria, e.g. the maximum completion time.

# Bibliography


[1] I. Kacem, H. Kellerer (2018). Improved Fully Polynomial Approximation Schemes for the Maximum Lateness Minimization with a Single Machine with a Fixed Operator or Non-Availability Interval. *International Conference on Computational Logistics*, 417-427.

[2] I. Kacem, M. Sahnoune, G. Schmidt (2017). Strongly Fully Polynomial Time Approximation Scheme for the weighted completion time minimization problem on two-parallel capacitated machines. *RAIRO - Operations Research*, 51(4):1177-1188.

[3] Cui and Z. Lu (2017). Minimizing the makespan on a single machine with flexible maintenances and jobs release dates. *Computers & Operations Research*, 80:11–22.

[4] R. Rajkanth, C. Rajendran, and H. Ziegler (2017). Heuristics to minimize the completion time variance of jobs on a single machine and on identical parallel machines. *The International Journal of Advanced Manufacturing Technology*, 88(5–8):1923-1936.

[5] I. Kacem, H. Kellerer (2016). Semi-online scheduling on a single machine with unexpected breakdown. *Theoretical Computer Science*, 646:40-48.

[6] V. Nguyen and H. P. Bao (2016). An Efficient Solution to the Mixed Shop Scheduling Problem Using a Modified Genetic Algorithm. *Procedia Computer Science*, 95:475–482.

[7] W. Ma, B. Xu, M. Liu, and H. Huang (2014). An Efficient Approximation Algorithm for Aircraft Arrival Sequencing and Scheduling Problem. *Mathematical Problems in Engineering*.

[8] B. Mitavskiy and J. He (2012). A Polynomial Time Approximation Scheme for a Single Machine Scheduling Problem Using a Hybrid Evolutionary Algorithm. *arXiv:1202.1708v1*, DOI: 10.1109/CEC.2012.6256166.

[9] J. Qiu and J. Yin (2009). An Adaptive Repulsive Particle Swarm Optimization for Makespan and Maximum Lateness Minimization in the Permutation Flowshop Schedul-

ing Problem. *In Intelligent Systems and Applications, ISA, International Workshop*, 1–4.

[10] T. Chiang, H. Cheng, and L. Fu (2008). An Efficient Heuristic for Minimizing Maximum Lateness on Parallel Batch Machines. *In Intelligent Systems Design and Applications, ISDA'08. Eighth International Conference*, 2:621–627.

[11] I. Kacem (2008). Approximation algorithm for the weighted flow-time minimization on a single machine with a fixed non-availability interval. *Computers & Industrial Engineering*, 54(3):401-410.

# Chapter 4

# Approximation Algorithms for Minimizing the Maximum Lateness and Makespan on Two-Parallel Machine[1]

*In this chapter, we consider a multiobjective scheduling problem, where the maximum lateness and the makespan have to be minimized on two identical machines. In this problem, we are given a set J of n jobs to be scheduled on two identical machines. Each job* $j \in J$ *has a processing time* $p_j$ *and a delivery time* $q_j$*. The machines are available at time* $t = 0$ *and each of them can process at most one job at a given time. This chapter proposes an exact algorithm (based on a dynamic programming algorithm) to generate the complete Pareto Frontier in a pseudo-polynomial time. Then, we present a PTAS (Polynomial Time Approximation Scheme) to generate an approximate Pareto Frontier. In this scheme, we use a simplification technique based on the merging of jobs. Furthermore, we present two FPTAS (Fully Polynomial Time Approximation Scheme) to generate an approximate Pareto Frontier, the first one is based on the conversion of the dynamic programming, the second one*

[1]These results have been the subject of several communications and a publication in a top international journal: Annals of Operations Research (ANOR).
Gais ALHADI, Imed KACEM, Pierre LAROCHE, Izzeldin M. OSMAN: An approximate Pareto set for minimizing the maximum lateness and makespan on parallel machines. MOPGP 2017: International Conference on Multiple Objective Programming and Goal Programming, October 30-31, 2017.

Gais ALHADI, Imed KACEM, Pierre LAROCHE, Izzeldin M. OSMAN: A PTAS for Minimizing the Maximum Lateness and Makespan on Two-Parallel Machine. AFROS 2018: The 2018 International Conference of the African Federation of Operational Research Societies, July 2-4, 2018.

Gais ALHADI, Imed KACEM, Pierre LAROCHE, Izzeldin M. OSMAN: Approximation Algorithms for Minimizing the Maximum Lateness and Makespan on Parallel Machines. Annals of Operations Research (SPRINGER), DOI: 10.1007/s10479-019-03250-x.

*is applied to the simplified instances given by the PTAS. The proposed FPTAS algorithms are strongly polynomial. Finally, some numerical experiments are provided in order to compare the four proposed approaches.*

## 4.1 Introduction

In this chapter we deal with a multiobjective scheduling problem on two parallel machines. The criteria to be minimized are the maximum lateness and makespan. The problem is formulated as follows. A set $J$ of $n$ jobs must be performed on two identical machines. These machines are available at time $t = 0$ and each of them cannot perform more than one job at a given time. Each job $j \in J$ has a processing time $p_j$ and a delivery time $q_j$. The problem is to find a sequence of jobs, with the objective of minimizing the maximum lateness $L_{max}$ and the makespan $C_{max}$. We assume that all data are integers and that jobs are sorted in non-increasing order of their delivery times $q_1 \geq q_2 \geq \ldots \geq q_n$.

It is noteworthy that during the last decade the multi-objective scheduling problems have attracted numerous researchers from all the world and have been widely studied in the literature. For example, Geng et al. (2018) [2] studied scheduling problem with or without precedence relations, where the objective is to minimize the makespan and the maximum cost. They have provided highly efficient polynomial-time algorithms to generate all Pareto optimal points. Chen and Zou (2014) [7] proposed a runtime analysis of a $(\mu + 1)$ multi-objective evolutionary algorithm for three multi-objective optimization problems with unknown attributes. They showed that when the size of the population is less than the total number of Pareto-vector, the $(\mu + 1)$ multi-objective evolutionary algorithm cannot obtain the expected polynomial runtime for the exact discrete multi-objective optimization problems. Thus, we must determine the size of the population equal to the total number of leading ones, trailing zeros. Furthermore, the expected polynomial runtime for the exponential discrete multi-objective optimization problem can be obtained by the ratio of $n/2$ to $\mu - 1$ over an appropriate period of time. They also showed that the $(\mu + 1)$ multi-objective evolutionary algorithm can be solved efficiently in polynomial runtime by obtaining an $\varepsilon-$ adaptive Pareto front.

The reader is invited to consult the papers by Paschos (2018) [1], Kumar (2008) [9], Amor et al. (2017) [4], Amor and Martel (2014) [5] and Demange and Paschos (2005) [10] for more state-of-the-art.

From the presented state-of-the-art, we can conclude that the multi-objective problem we consider in this chapter has not been studied before in the literature. For this reason, the design of efficient methods for this problem is a new attempt. Moreover, the study of

polynomial approximation algorithms for this parallel machine problem seems to be very challenging at the scientific level, as well as the practical level (since it allows us to guarantee the performance of the solutions in reasonable computation time).

The remainder of this chapter is organized as follows. In Section 4.2, we describe the proposed dynamic programming algorithm. Section 4.3 provides the description and the analysis of the PTAS based on the merging technique. Section 4.4 presents the steps and the analysis of the FPTAS. In Section 4.5 is described an improved FPTAS based on the merging technique. Section 4.6, presents some results obtained by using our algorithms. Finally, Section 4.7 concludes the chapter.

## 4.2 Dynamic Programming Algorithm

The following dynamic programming algorithm $A$ can be applied to solve exactly this problem. This algorithm $A$ iteratively generates some sets of states. At each iteration $j$, a set $\chi_j$ composed of states is generated $(0 \leq j \leq n)$. Each state $[k, L_{max}, C_{max}]$ in $\chi_j$ can be associated to a feasible partial schedule for the first $j$ jobs. Let variable $k \in \{0,1\}$ denote the most loaded machine, $L_{max}$ denote the maximum lateness and $C_{max}$ denote the maximum completion time of the corresponding schedule. The dynamic programming algorithm is described by Algorithm $A$.

The standard version of this dynamic programming has an exponential complexity since it needs $O(2^n)$ time. Nevertheless, this complexity can be reduced to $O(n \cdot P)$ by keeping only one state having the smallest value of $L_{max}$, when one has several triplets with the same $C_{max}$ and $k$ in $\chi_j$ (for every iteration $j \in \{2, \ldots, n\}$). Therefore, algorithm $A$ can be reduced to a pseudo-polynomial algorithm, which proves at the same time that the problem is NP-hard only in the ordinary sense.

**Algorithm** $A$

1. Set $\chi_1 = \{[1, p_1 + q_1, p_1]\}$.
2. For $j \in \{2, 3, ..., n\}$ do
   (a) $\chi_j = \emptyset$.
   (b) For each state $[k, L_{max}, C_{max}]$ in $\chi_{j-1}$ do
      - (schedule job $j$ on machine $k \Rightarrow C_{max}$ stays on machine $k$ )
        add $[k, max\{L_{max}, C_{max} + p_j + q_j\}, C_{max} + p_j]$ to $\chi_j$

- (schedule job $j$ on machine $1-k \Rightarrow C_{max}$ stays on machine $k$ or $C_{max}$ goes to machine $1-k$)
  if $(C_{max} \geq \sum_{i=1}^{j} p_i - C_{max})$
  - add $[k, max\{L_{max}, \sum_{i=1}^{j} p_i - C_{max} + q_j\}, C_{max}]$ to $\chi_j$
    else
  - add $[1-k, max\{L_{max}, \sum_{i=1}^{j} p_i - C_{max} + q_j\}, \sum_{i=1}^{j} p_i - C_{max}]$ to $\chi_j$

(c) For every $k$, for every $C_{max}$: keep only one state with the smallest possible $L_{max}$.

(d) Remove $\chi_{j-1}$.

3. Return the Pareto front of $\chi_n$, by only keeping non-dominated states.

***Comment***: To destroy the symmetry, we start by $\chi_1 = \{[1, p_1 + q_1, p_1]\}$ (i.e., we perform job 1 on the first machine).

## 4.3 New Simplifications and PTAS

In this section, we describe our PTAS (Polynomial Time Approximation Scheme). It uses a simplification method consisting in merging some similar jobs (small and having equivalent tails), inspired from Kacem and Kellerer (2014) [6]. This PTAS is described by the following Algorithm.

**PTAS Algorithm**

1. Let $\varepsilon > 0$ (assume that $\frac{2}{\varepsilon}$ is an integer) and $J = \{1, 2, ..., n\}$.
2. Adjust every tail $q_j (j \in J)$ as follows: $q_j := \lfloor \lceil \frac{q_j}{\varepsilon q_{max}/2} \rceil \cdot \frac{\varepsilon q_{max}}{2} \rfloor$ and define the $\frac{2}{\varepsilon}$ classes $C_k (1 \leq k \leq 2/\varepsilon)$ such that the tail of every job in $C_k$ is equal to $\lfloor k.\varepsilon q_{max} \rfloor$.
3. For every class $C_k (1 \leq k \leq 2/\varepsilon)$:
   (a) Divide the set of all available jobs into two subsets as follows:
      i. The small jobs in $S$, where $S = \{j \in J | p_j < \varepsilon^2 P/8\}$ .
      ii. The large jobs in $G$, where $G = \{j \in J | p_j \geq \varepsilon^2 P/8)\}$ .
   (b) Merge jobs in $S$ until the processing time $p$ of the obtained job will satisfy $\varepsilon^2 P/8 \leq p < \varepsilon^2 P/4$ or remain single small job in the class.
4. After merging the small jobs, the new instance $I''$ is optimally solved by using the dynamic programming algorithm.

5. This schedule of $I^{''}$ is used to schedule the jobs of instance $I'$: when a job $j$ of instance $I^{''}$ is scheduled on a machine, then all the corresponding jobs of instance $I'$ are also scheduled on this same machine.

6. Return the Pareto front from the solutions obtained in Step (5), by only keeping non-dominated states.

In the following, the two main steps of the algorithm are described and the corresponding proofs are given.

**STEP 1:**

Let $L^*_{max}(I)$ denote the minimal maximum lateness for instance $I$.
First, we simplify the difficult instance $I$ into a more primitive instance $I^{'}$ which is easier to deal with, depending on the desired precision $\varepsilon$ of approximation. The simplification will be as follows:
Given an arbitrary $\varepsilon > 0$. We assume that $\frac{2}{\varepsilon}$ is an integer (to have an integer number of job classes, and to have the same delivery time value for each class) and we split the interval $[0, max_{j\in J}\{q_j\}]$ in $\frac{2}{\varepsilon}$ equal length classes. In class $C_k (1 \leq k \leq 2/\varepsilon)$ we adjust each tail $q_j$ to $q_j := \lfloor \lceil \frac{q_j}{\varepsilon q_{max}/2} \rceil \cdot \frac{\varepsilon q_{max}}{2} \rfloor$ where $q_{max} = max_{j\in J}\{q_j\}$. The obtained instance is called $I^{'}$.

**Proposition 4.3.0.1.** *The modified instance $I^{'}$ can be computed in $O(n)$ time and this will not lead to more than $(1+\varepsilon/2)-$loss.*

*Proof.* The modification can be done by setting $q_j := \lfloor \lceil \frac{q_j}{\varepsilon q_{max}/2} \rceil \cdot \frac{\varepsilon q_{max}}{2} \rfloor$ for every $j \in J$. Then, it can be done in $O(n)$ time.
Moreover, since $\lfloor \lceil \frac{q_j}{\varepsilon q_{max}/2} \rceil \cdot \frac{\varepsilon q_{max}}{2} \rfloor \leq q_j + \varepsilon q_{max}/2$ and $q_j \leq L^*_{max}(I)$.
Therefore, it can be deduced that: $L^*_{max}(I^{'}) \leq L^*_{max}(I) + \varepsilon q_{max}/2$ then,
$L^*_{max}(I^{'}) \leq L^*_{max}(I) + \varepsilon L^*_{max}(I)/2 \leq (1+\varepsilon/2) L^*_{max}(I)$ □

**STEP 2:**

In this step, $J$ is divided into at most $2/\varepsilon$ subsets $C_k$ $(1 \leq k \leq 2/\varepsilon)$, with a unique tail equal to $\lfloor k.\varepsilon q_{max} \rfloor$ for every job in $C_k$. Then, we reduce the number of small jobs in every subset $C_k$. Indeed, for each class we distinguish the large and small jobs. The small jobs have processing time less than $\varepsilon^2 P/8$ and the large jobs have processing time greater or equal to $\varepsilon^2 P/8$, where $P = \sum_{j=1}^{n} p_j$.

The simplification consists in proceeding by merging some similar small jobs, in an iterative way, in each class $C_k$. In every class, we merge jobs until having a new greater job that has a processing time between $\varepsilon^2 P/8$ and $\varepsilon^2 P/4$. The small jobs are processed

according to Jackson's order. At the end, it remains at most a single small job for each class $C_k$. At most, in the new instance, $(\frac{8}{\varepsilon^2}+\frac{2}{\varepsilon})$ jobs can remain. Clearly, $\frac{2}{\varepsilon}$ for the small jobs of every subset $C_k$ and $\frac{8}{\varepsilon^2}$ for the large jobs.

After merging the small jobs, a new instance $I''$is obtained. Thus, a matching array is created to identify for each job of instance $I''$ the corresponding jobs in the instance $I'$. Moreover, the instance $I''$ will be solved by using the dynamic programming algorithm $A$.

Finally, the obtained schedule of $I''$ is used to schedule the jobs of instance $I'$. When a job $j'$of instance $I''$ is scheduled on a machine, then all the corresponding jobs of instance $I'$ are also scheduled on the same machine. Obviously, in instance $I'$ jobs are scheduled on every machine by using Jackson's order.

**Theorem 4.3.0.1.** *The studied problem has a Polynomial Time Approximation Scheme (PTAS) with a time complexity of* $O(n.2^{(\frac{8}{\varepsilon^2}+\frac{2}{\varepsilon})})$.

*Proof.* The proof is based on the analysis of the previous steps. By Proposition 1, we need to prove that the second step will not cost more than $(1+\varepsilon/2)-$loss. The instance $I''$ obtained after the second step, has a limited number of jobs. All their possible assignments on the two machines can be determined in $O(2^{(\frac{8}{\varepsilon^2}+\frac{2}{\varepsilon})})$. Then, we can derive all the associated feasible solutions for $I'$ in $O(n\cdot 2^{(\frac{8}{\varepsilon^2}+\frac{2}{\varepsilon})})$ and select the best schedule. Let $\Delta_k$ be the length of jobs to be assigned from class $C_k$ on the first machine in an optimal schedule/assignment of instance $I'$. We will show that the PTAS generates a feasible assignment $AS$ with a length of jobs assigned from class $C_k$ on the first machine, (this length is denoted as $\Delta'_k$ ), which is "close" enough to the value $\Delta_k$. In this assignment $AS$, the large jobs, not merged, to be optimally assigned from class $C_k$ on the first machine are assumed to be known (by guessing). We have to approximate the optimal remaining length of the other (small) jobs on the first machine. W.L.O.G., we consider that this remaining length is equal to $\Delta_k$ (since the large jobs, not merged, to be optimally assigned on the first machine can be correctly guessed). Take now the merged jobs (and possibly the only remaining small job after the merging step) from class $C_k$ and add their processing times one by one in any order until we get a total processing time $g$ minimally greater or equal to $\Delta_k$. If $g-\Delta_k \leq \varepsilon^2 P/8$, then take $\Delta'_k = g$, otherwise eliminate the processing time of the last added job and take the resulted total processing time $f < g$ (i.e., we take $\Delta'_k = f$). In this latter case, it can be demonstrated that $\Delta_k - f \leq \varepsilon^2 P/8$, otherwise the last eliminated merged job should be longer than $\varepsilon^2 P/4$, which is a contradiction with the definition of merged jobs in Step 2. By symmetry, the loss on the second machine will be limited to $\varepsilon^2 P/8$ for class $C_k$. Since there are at most $\frac{2}{\varepsilon}$ classes, the total loss is bounded by $\varepsilon P/4$, which is less or equal to $\varepsilon L^*_{max}(I)/2$. This loss is limited for the two criteria. □

## 4.4 FPTAS

The main idea of our FPTAS consists in eliminating a specific subset of the states created by the DP algorithm. Therefore, the modified algorithm $A'$ produces an approximate solution close to the optimum (see the detail in FPTAS Algorithm $A'$).
Given an arbitrary $\varepsilon > 0$, we define the following parameters:

$$\delta_1 = \frac{\varepsilon P/2}{n},$$

and

$$\delta_2 = \frac{\varepsilon(P + q_{max})/3}{n}.$$

Let $L^*_{max}$ and $C^*_{max}$ be the minimal values and let $LMAX$ and $CMAX$ be the upper bounds for the two considered objectives, such that,

$$0 \leq LMAX = P + q_{max} \leq 3L^*_{max}$$

$$0 \leq CMAX = P \leq 2C^*_{max}$$

We divide the intervals $[0, CMAX]$ and $[0, LMAX]$ into equal sub-intervals respectively of lengths $\delta_1$ and $\delta_2$. Then, an FPTAS is defined by following the same procedure as in the dynamic programming, except the fact that it will keep at every iteration $j$ only one representative state for every couple of the defined subintervals produced from $[0, CMAX]$ and $[0, LMAX]$. Thus, our FPTAS will generate approximate sets $\chi^{\#}_j$ of states instead of $\chi_j$.

**FPTAS Algorithm** $A'$

1. Let $\varepsilon > 0$ and $j = \{1, 2, ..., n\}$.
2. Split the interval $[0, CMAX]$ and $[0, LMAX]$ into sub-intervals respectively of sizes $\delta_1$ and $\delta_2$.
3. Set $\chi^{\#}_1 = \{[1, p_1 + q_1, p_1]\}$.
4. For $j \in \{2, 3, ..., n\}$ do
   (a) $\chi^{\#}_j = \phi$.
   (b) For each state $[k, L_{max}, C_{max}]$ in $\chi^{\#}_{j-1}$ do

- add $[k,max\{L_{max},C_{max}+p_j+q_j\},C_{max}+p_j]$ to $\chi_j^{\#}$ (i.e., schedule job $j$ on machine $k$)
- if $(C_{max} \geq \sum_{i=1}^{j} p_i - C_{max})$
  - add $[k,max\{L_{max},\sum_{i=1}^{j} p_i - C_{max}+q_j\},C_{max}]$ to $\chi_j^{\#}$ (i.e., schedule job $j$ on machine $1-k$)
    else
  - add $[1-k,max\{L_{max},\sum_{i=1}^{j} p_i - C_{max}+q_j\},\sum_{i=1}^{j} p_i - C_{max}]$ to $\chi_j^{\#}$ (i.e., schedule job $j$ on machine $1-k$)

(c) Keep only one representative state for every couple of the defined sub-intervals produced from $[0,CMAX]$ and $[0,LMAX]$.

(d) Remove $\chi_{j-1}^{\#}$.

5. Return the Pareto front from the solutions obtained in previous step, by only keeping non-dominated states.

The following lemma shows the closeness of the result generated by the FPTAS compared to the dynamic programming.

**Lemma 4.4.0.1.** *For each state $[k,L_{max},C_{max}] \in \chi_i$ there exists at least one approximate state $[m,L_{max}^{\#},C_{max}^{\#}] \in \chi_i^{\#}$ such that:*

$$L_{max}^{\#} \leq L_{max} + j.\max\{\delta_1,\delta_2\},$$

*and*

$$C_{max} - j.\delta_1 \leq C_{max}^{\#} \leq C_{max} + j.\delta_1.$$

*Proof.* By induction on $j$.
First, for $j=0$ we have $\chi_j^{\#} = \chi_1$. Therefore, the statement is trivial. Now, assume that the lemma is valid until level $j-1$. Consider an arbitrary state $[k,L_{max},C_{max}] \in \chi_j$. Algorithm $A$ introduces this state into $\chi_j$ when job $j$ is considered to complete a certain partial solution for the first $j-1$ jobs. Let $[k',L'_{max},C'_{max}]$ be the above feasible state. Three cases can be distinguished:

1. $[k,L_{max},C_{max}] = [k', \max\{L'_{max}, C'_{max}+p_j+q_j\}, C'_{max}+p_j]$
2. $[k,L_{max},C_{max}] = [k', \max\{L'_{max}, \sum_{i=1}^{j} p_i - C'_{max}+q_j\}, C'_{max}]$
3. $[k,L_{max},C_{max}] = [1-k', \max\{L'_{max}, \sum_{i=1}^{j} p_i - C'_{max}+q_j\}, \sum_{i=1}^{j} p_i - C'_{max}]$

We demonstrate the property for level $j$ in the three cases.

- $1^{st}$ **Case:** $[k, L_{max}, C_{max}] = [k^{'}, \max\{L^{'}_{max}, C^{'}_{max} + p_j + q_j\}, C^{'}_{max} + p_j]$

Since $[k^{'}, L^{'}_{max}, C^{'}_{max}] \in \chi_{j-1}$, there exists $[k^{'\#}, L^{'\#}_{max}, C^{'\#}_{max}] \in \chi^{\#}_{j-1}$, such that:
$L^{'\#}_{max} \leq L^{'}_{max} + (j-1)\max\{\delta_1, \delta_2\}$ and $C^{'}_{max} - (j-1)\delta_1 \leq C^{'\#}_{max} \leq C^{'}_{max} + (j-1)\delta_1$.
Consequently, the state $[k^{'\#}, \max\{L^{'\#}_{max}, C^{'\#}_{max} + p_j + q_j\}, C^{'\#}_{max} + p_j]$ is created by algorithm $A^{'}$at iteration $j$. But, it may be eliminated when we reduce the state space. Let $[\alpha, \lambda, \mu]$ be the state in $\chi^{\#}_j$ that is in the same box as the sate $[k^{'\#}, \max\{L^{'\#}_{max}, C^{'\#}_{max} + p_j + q_j\}, C^{'\#}_{max} + p_j]$.
Hence, we have:

$$\begin{aligned}
\lambda &\leq \max\{L^{'\#}_{max}, C^{'\#}_{max} + p_j + q_j\} + \delta_2 \\
&\leq \max\{L^{'}_{max} + (j-1)\max\{\delta_1, \delta_2\}, C^{'}_{max} + (j-1)\delta_1 + p_j + q_j\} + \delta_2 \\
&\leq \max\{L^{'}_{max}, C^{'}_{max} + p_j + q_j\} + (j-1)\max\{\delta_1, \delta_2\} + \delta_2 \\
&\leq L_{max} + j\max\{\delta_1, \delta_2\} \qquad (4.4.1)
\end{aligned}$$

In addition,

$$\mu \leq C^{'\#}_{max} + p_j + \delta_1 \leq \quad C^{'}_{max} + (j-1)\delta_1 + p_j + \delta_1 = C_{max} + j\delta_1. \qquad (4.4.2)$$

and,

$$\mu \geq C^{'\#}_{max} + p_j - \delta_1 \geq C^{'}_{max} - (j-1)\delta_1 + p_j - \delta_1 \geq C_{max} - j\delta_1. \qquad (4.4.3)$$

Consequently, $[\alpha, \lambda, \mu]$ is an approximate state verifying the two conditions.

- $2^{nd}$ **Case:** $[k, L_{max}, C_{max}] = [k^{'}, max\{L^{'}_{max}, \sum_{i=1}^{j} p_i - C^{'}_{max} + q_j\}, C^{'}_{max}]$

Since $[k^{'}, L^{'}_{max}, C^{'}_{max}] \in \chi_{j-1}$, there exists $[k^{'\#}, L^{'\#}_{max}, C^{'\#}_{max}] \in \chi^{\#}_{j-1}$, such that:
$L^{'\#}_{max} \leq L^{'}_{max} + (j-1)max\{\delta_1, \delta_2\}$ and $C^{'}_{max} - (j-1)\delta_1 \leq C^{'\#}_{max} \leq C^{'}_{max} + (j-1)\delta_1$.

Consequently, two sub-cases can occur:

- Sub-case 2.1:$\sum_{i=1}^{j} p_i - C^{'\#}_{max} \leq C^{'\#}_{max}$

Here, the state $[k^{'\#}, \max\{L^{'\#}_{max}, \sum_{i=1}^{j} p_i - C^{'\#}_{max} + q_j\}, C^{'\#}_{max}]$ is created by algorithm $A^{'}$at iteration $j$. But, it may be eliminated when we reduce the state space. Let $[\alpha, \lambda, \mu]$ be the state in $\chi^{\#}_j$ that is in the same box as $[k^{'\#}, \max\{L^{'\#}_{max}, \sum_{i=1}^{j} p_i - C^{'\#}_{max} + q_j\}, C^{'\#}_{max}]$. Hence, we have:

$$\begin{aligned}
\lambda &\leq \max\{L_{max}^{'\#}, \sum_{i=1}^{j} p_i - C_{max}^{'\#} + q_j\} + \delta_2 \\
&\leq \max\{L_{max}^{'} + (j-1)\max\{\delta_1,\delta_2\}, \sum_{i=1}^{j} p_i - (C_{max}^{'} - (j-1)\delta_1) + q_j\} + \delta_2 \\
&\leq \max\{L_{max}^{'}, \sum_{i=1}^{j} p_i - C_{max}^{'} + q_j\} + (j-1)\max\{\delta_1,\delta_2\} + \delta_2 \\
&\leq L_{max} + (j-1)\max\{\delta_1,\delta_2\} + \delta_2 \\
&< L_{max} + j\max\{\delta_1,\delta_2\} \qquad (4.4.4)
\end{aligned}$$

Moreover,

$$\mu \leq C_{max}^{'\#} + \delta_1 \leq C_{max}^{'} + (j-1)\delta_1 + \delta_1 = C_{max} + j\delta_1. \qquad (4.4.5)$$

And,

$$\mu \geq C_{max}^{'\#} - \delta_1 \geq C_{max}^{'} - (j-1)\delta_1 - \delta_1 = C_{max} - j\delta_1. \qquad (4.4.6)$$

Consequently, $[\alpha, \lambda, \mu]$ is an approximate state verifying the two conditions.

- Sub-case 2.2:$\sum_{i=1}^{j} p_i - C_{max}^{'\#} > C_{max}^{'\#}$

Here, the state $[1-k^{'\#}, \max\{L_{max}^{'\#}, \sum_{i=1}^{j} p_i - C_{max}^{'\#} + q_j\}, \sum_{i=1}^{j} p_i - C_{max}^{'\#}]$ is created by algorithm $A^{'}$at iteration $j$. But, it may be eliminated when we reduce the state space. Let $[\alpha, \lambda, \mu]$ be the state in $\chi_j^{\#}$ that is in the same box as $[1-k^{'\#}, \max\{L_{max}^{'\#}, \sum_{i=1}^{j} p_i - C_{max}^{'\#} + q_j\}$, $\sum_{i=1}^{j} p_i - C_{max}^{'\#}]$. Hence, we have:

$$\begin{aligned}
\lambda &\leq \max\{L_{max}^{'\#}, \sum_{i=1}^{j} p_i - C_{max}^{'\#} + q_j\} + \delta_2 \\
&\leq \max\{L_{max}^{'} + (j-1)\max\{\delta_1,\delta_2\}, \sum_{i=1}^{j} p_i - (C_{max}^{'} - (j-1)\delta_1) + q_j\} + \delta_2 \\
&\leq \max\{L_{max}^{'}, \sum_{i=1}^{j} p_i - C_{max}^{'} + q_j\} + (j-1)\max\{\delta_1,\delta_2\} + \delta_2 \\
&\leq L_{max} + (j-1)\max\{\delta_1,\delta_2\} + \delta_2 \\
&< L_{max} + j\max\{\delta_1,\delta_2\} \qquad (4.4.7)
\end{aligned}$$

Moreover,

$$\mu \leq \sum_{i=1}^{j} p_i - C'^{\#}_{max} + \delta_1 \tag{4.4.8}$$

Since $C'^{\#}_{max} \geq C'_{max} - (j-1)\delta_1$, then the following relation holds

$$\mu \leq \sum_{i=1}^{j} p_i - C'_{max} + j\delta_1 \leq C_{max} + j\delta_1 (\text{since} \sum_{i=1}^{j} p_i - C'_{max} \leq C'_{max}). \tag{4.4.9}$$

And,

$$\mu \geq \sum_{i=1}^{j} p_i - C'^{\#}_{max} - \delta_1 \geq C'^{\#}_{max} - \delta_1 \geq C_{max} - j\delta_1. \tag{4.4.10}$$

Thus, $[\alpha, \lambda, \mu]$ verifies the necessary conditions.

- $3^{rd}$ **Case:** $[k, L_{max}, C_{max}] = [1-k', \max\{L'_{max}, \sum_{i=1}^{j} p_i - C'_{max} + q_j\}, \sum_{i=1}^{j} p_i - C'_{max}]$

Since $[k', L'_{max}, C'_{max}] \in \chi_{j-1}$, there exists $[k'^{\#}, L'^{\#}_{max}, C'^{\#}_{max}] \in \chi^{\#}_{j-1}$, such that:
$L'^{\#}_{max} \leq L'_{max} + (j-1)\max\{\delta_1, \delta_2\}$ and $C'_{max} - (j-1)\delta_1 \leq C'^{\#}_{max} \leq C'_{max} + (j-1)\delta_1$.
Consequently, two sub-cases can occur:

- Sub-case 3.1:$\sum_{i=1}^{j} p_i - C'^{\#}_{max} \geq C'^{\#}_{max}$

Here, the state $[1-k'^{\#}, \max\{L'^{\#}_{max}, \sum_{i=1}^{j} p_i - C'^{\#}_{max} + q_j\}, \sum_{i=1}^{j} p_i - C'^{\#}_{max}]$ is created by algorithm $A'$ at iteration $j$. But, it may be eliminated when we reduce the state space. Let $[\alpha, \lambda, \mu]$ be the state in $\chi^{\#}_j$ that is in the same box as $[1-k'^{\#}, \max\{L'^{\#}_{max}, \sum_{i=1}^{j} p_i - C'^{\#}_{max} + q_j\}, \sum_{i=1}^{j} p_i - C'^{\#}_{max}]$. Hence, we have:

$$\begin{aligned}
\lambda &\leq \max\{L'^{\#}_{max}, \sum_{i=1}^{j} p_i - C'^{\#}_{max} + q_j\} + \delta_2 \\
&\leq \max\{L'_{max} + (j-1)\max\{\delta_1, \delta_2\}, \sum_{i=1}^{j} p_i - (C'_{max} - (j-1)\delta_1) + q_j\} + \delta_2 \\
&\leq \max\{L'_{max}, \sum_{i=1}^{j} p_i - C'_{max} + q_j\} + (j-1)\max\{\delta_1, \delta_2\} + \delta_2 \\
&\leq L_{max} + (j-1)\max\{\delta_1, \delta_2\} + \delta_2 \\
&\leq L_{max} + j\max\{\delta_1, \delta_2\}
\end{aligned} \tag{4.4.11}$$

and

$$\mu \leq \sum_{i=1}^{j} p_i - C'^{\#}_{max} + \delta_1 \leq \sum_{i=1}^{j} p_i - (C'_{max} - (j-1)\delta_1) + \delta_1 \leq C_{max} + j\delta_1. \quad (4.4.12)$$

In the other hand, we have

$$\mu \geq \sum_{i=1}^{j} p_i - C'^{\#}_{max} - \delta_1 \geq \sum_{i=1}^{j} p_i - (C'_{max} + (j-1)\delta_1) - \delta_1 \geq C_{max} - j\delta_1. \quad (4.4.13)$$

Thus, $[\alpha, \lambda, \mu]$ fulfills the conditions.

- Sub-case 3.2:$\sum_{i=1}^{j} p_i - C'^{\#}_{max} < C'^{\#}_{max}$

Here, the state $[k'^{\#}, \max\{L'^{\#}_{max}, \sum_{i=1}^{j} p_i - C'^{\#}_{max} + q_j\}, C'^{\#}_{max}]$ is created by algorithm $A'$ at iteration $j$. But, it may be eliminated when we reduce the state space. Let $[\alpha, \lambda, \mu]$ be the state in $\chi_j^{\#}$ that is in the same box as $[k'^{\#}, \max\{L'^{\#}_{max}, \sum_{i=1}^{j} p_i - C'^{\#}_{max} + q_j\}, C'^{\#}_{max}]$. Hence, we have:

$$\begin{aligned}
\lambda &\leq \max\{L'^{\#}_{max}, \sum_{i=1}^{j} p_i - C'^{\#}_{max} + q_j\} + \delta_2 \\
&\leq \max\{L'_{max} + (j-1)\max\{\delta_1, \delta_2\}, \sum_{i=1}^{j} p_i - (C'_{max} - (j-1)\delta_1) + q_j\} + \delta_2 \\
&\leq \max\{L'_{max}, \sum_{i=1}^{j} p_i - C'_{max} + q_j\} + (j-1)\max\{\delta_1, \delta_2\} + \delta_2 \\
&\leq L_{max} + (j-1)\max\{\delta_1, \delta_2\} + \delta_2 \\
&\leq L_{max} + j\max\{\delta_1, \delta_2\} \qquad (4.4.14)
\end{aligned}$$

and

$$\mu \leq C'^{\#}_{max} + \delta_1 \leq C'_{max} + (j-1)\delta_1 + \delta_1 \leq C_{max} + j\delta_1. \quad (4.4.15)$$

In the other hand, we have

$$\mu \geq C'^{\#}_{max} - \delta_1 \geq \sum_{j=1}^{i} p_j - C'^{\#}_{max} - \delta_1 \geq \sum_{i=1}^{j} p_i - C'_{max} - j\delta_1 = C_{max} - j\delta_1. \quad (4.4.16)$$

Therefore, $[\alpha, \lambda, \mu]$ fulfills the conditions.

Consequently, the property is also valid for level $i$ in the third case, and this completes the proof. □

Based on the lemma, we deduce easily that for every non-dominated state $[k, L_{max}, C_{max}] \in \chi_n$, it must remain a close state $[m, L^{\#}_{max}, C^{\#}_{max}] \in \chi^{\#}_n$ such that:

$$L^{\#}_{max} \leq L_{max} + n\max\{\delta_1, \delta_2\} \leq (1+\varepsilon)L_{max}$$

and

$$C^{\#}_{max} \leq C_{max} + n\delta_1 \leq (1+\varepsilon)C_{max}.$$

Moreover, it is clear that the FPTAS runs polynomially in $n$ and $1/\varepsilon$.

**Theorem 4.4.0.2.** *The studied problem has a Fully Polynomial Time Approximation Scheme (FPTAS) with a time complexity of $O(n^3/\varepsilon^2)$.*

## 4.5 Improved FPTAS

In this section, we describe an improved FPTAS algorithm. Noteworthy, the improvements used in the PTAS algorithm are exploited to improve the previous FPTAS (i.e., we exploit the modification of the input $(I - I'')$). Noteworthy, the two FPTAS are strongly polynomial. The modified algorithm becomes faster. In Section 4.6, we will present some numerical experiments in order to compare the proposed algorithms.

### Improved FPTAS Algorithm

1. Let $\varepsilon > 0$ (assume that $\frac{2}{\varepsilon}$ is an integer).
2. Let $J = \{1, 2, ..., n\}$.
3. Split the interval $[0, max_{j\in J}\{q_j\}]$ in $\frac{2}{\varepsilon}$ equal-length sub-intervals.
4. For every $j \in C_k (1 \leq k \leq 2/\varepsilon)$ do
   (a) Round up every tail $q_j$ as follows: $q_j := \lfloor \lceil \frac{q_j}{\varepsilon q_{max}/2} \rceil \cdot \frac{\varepsilon q_{max}}{2} \rfloor$.
   (b) Divide the set of all available jobs into two subsets as follows:
      i. The small jobs $S$, where $S = \{j \in J | p_j < \varepsilon^2 P/8\}$ .
      ii. The large jobs $G$, where $G = \{j \in J | p_j \geq \varepsilon^2 P/8\}$ .
   (c) Merge jobs in $S$ until the processing time $p$ of the obtained job will satisfy $\varepsilon^2 P/8 \leq p < \varepsilon^2 P/4$ or remain single small job in the class.

5. After merging the small jobs, the new instance $I''$ will be solved using the FPTAS algorithm $A'$ (presented in Section 4.4).

## 4.6 Results

The following results have been obtained after testing the efficiency of every method. The code has been implemented in Java and the experiments have been performed on an Intel(R) Core(TM)-i7 with 8GB RAM. The results have been tested with different values of $\varepsilon$: 0.8, 0.4, 0.2 and 0.1. To ensure the consistency of running times, each test has been run three times.

The hypervolume indicator (HV), introduced by Bradstreet (2011) [8] has recently received more attention as a measure of the quality of the Pareto front. Therefore, we will use this indicator to measure the performance of our algorithms. Noteworthy, the hypervolume is the most famous indicator that can reflect the dominance of Pareto fronts. In this chapter, we use a program by Fonseca et al. (2018) [3] that implements a repetitive algorithm, scanning the dimension for calculating the hypervolume indicator for the quality of a set of non-dominated points in a multi-dimensional space. The hypervolume of a set of points is measured relatively to a reference point, usually the bounding point in space. It can be seen as the $R$-dimensional space contained in a set of non-dominated points, where $R$ is the number of criteria to optimize in our problem (2 in our study) (see Bradstreet (2011) [8]). The reference point is determined by taking the worst value (worst solution + 1) in each dimension, among the solutions found by our algorithms.

To compare the quality of our approximation algorithms, the HV ratio (denoted by $HV_r$) is used to compute the distance between the approximate solutions and the optimal Pareto front provided by the Dynamic Programming algorithm. $HV_r$ has been obtained using the following formula:

$$HV_r = (1 - \frac{HV_{DP} - HV_{APPROX}}{HV_{DP}})$$

Fig. 4.1, presents an example to illustrate the notion of the hypervolume indicator, for the two objectives of our study: $L_{max}$ and $C_{max}$. On the left part of the figure are presented the solutions of the algorithms, the Hypervolume of each Pareto front, and the quality based on Hypervolume. On the right part, these solutions are presented in an orthonormal coordinate system, allowing to give a graphical representation of the hypervolumes.

| Algorithm | Pareto fronts | HV | $HV_r$ |
|---|---|---|---|
| DP | {(37, 30), (36, 32)} | 70 | N/A |
| PTAS | {(41, 31), (40, 33), (38, 35)} | 38 | 54,3% |
| FPTAS | {(42, 38)} | 2 | 2,7% |
| Impr. FPTAS | {(41, 34), (43, 30)} | 19 | 27,1% |

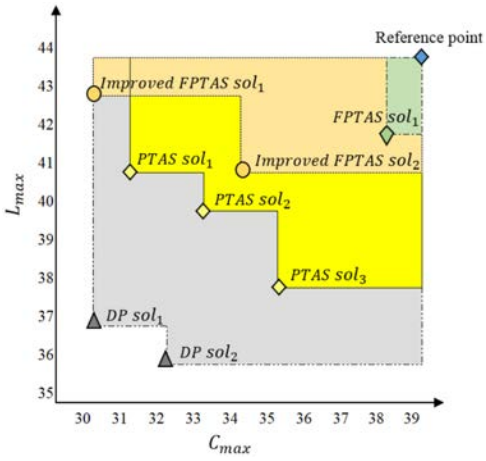



Fig. 4.1 Comparison of two dimensions Pareto sets using Hypervolume

Furthermore, to compare the two objectives of our PTAS and two FPTAS algorithms, we will use the average quality of the two objectives: $L^P_{max}/L^*_{max}$ and $C^P_{max}/C^*_{max}$ for our PTAS (respectively, $L^F_{max}/L^*_{max}$ and $C^F_{max}/C^*_{max}$ for our two FPTAS).

The remainder of this section is organized as follows. In Subsection 4.6.1, we will present the results of the small instances. The results of the big instances are presented in Subsection 4.6.2. Finally, Subsection 4.6.3 compares the results of our algorithms.

### 4.6.1 First instances

We have randomly generated five sets of instances, with different numbers of jobs and various processing and delivery times:

- number of jobs: from 5 to 25, 26 to 50, 51 to 75, 76 to 100 and 100 to 200;
- processing times : from 1 to 20, 1 to 100 and 1 to 500;
- delivery times : from 1 to 20, 1 to 100 and 1 to 500;

For each set of parameters, we have generated three different instances. That gave us 135 instances in each set of instances.
In this subsection, we present the results for our five sets of instances, from small instances (5-25 jobs) to bigger ones (100-200 jobs).

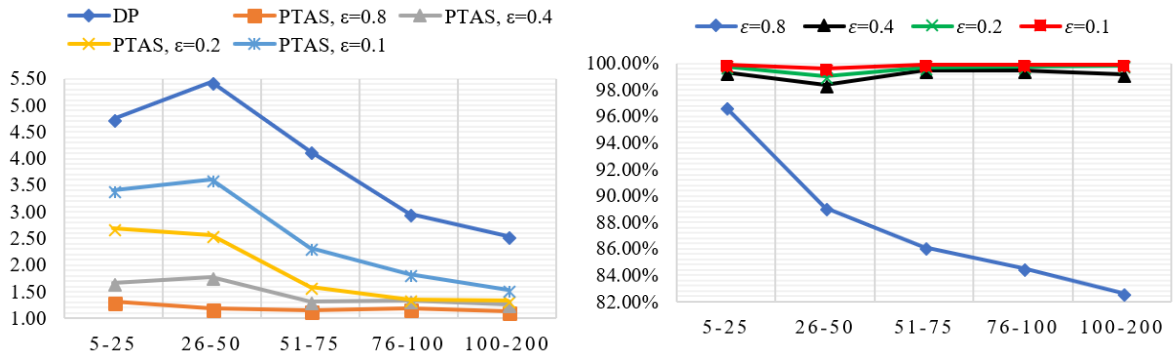
It is worth mentioning that we will present the results of the PTAS in Subsection 4.6.1.1. Subsection 4.6.1.2 provides the results of the FPTAS. Finally, the improved FPTAS results will be presented in Subsection 4.6.1.3.

#### 4.6.1.1. Results of our PTAS algorithm vs. DP algorithm

**Quality of our PTAS algorithm**

In this section, we will see the quality of our PTAS algorithm using HV. We will also present the results for the average values of $L_{max}$ and $C_{max}$.

Fig.4.2 presents a comparison of our PTAS and Dynamic Programming (DP). The results are given for our five sets of instances, from small instances (5-25 jobs) to bigger ones (100-200 jobs). On these two figures (A and B), we can observe that: with a small $\varepsilon$ value, the size of the Pareto front is near the size of the DP, and with big $\varepsilon$ value, we lose a lot of solutions (see Fig. 4.2. A). We can see that with good $\varepsilon$ (small $\varepsilon$), we have good solutions (see Fig. 4.2. B), which is consistent with the theory. We can see that it is true for each set of instances: the best solutions are obtained with $\varepsilon = 0.1$.




A. Size of Pareto Front

B. Hypervolume Indicator ($HV_r$)

Fig. 4.2 Quality of our PTAS algorithm with $\varepsilon = 0.8, 0.4, 0.2$ and $0.1$

Another remark can be done from Fig. 4.2: the results of $\varepsilon = 0.8$ are very bad. First, (Fig. 4.2. A), the size of the Pareto front is very small. When the size of the front for the DP is between 2.5 and 5.5, the size of the front for PTAS using $\varepsilon = 0.8$ is always less than 1.5. Secondly (Fig. 4.2. B), the HV ratio is very bad and strongly decreases with the size of instances. This can be explained as follows: we can see in the second column in Table 4.1 ($\varepsilon$=0.8), whatever the size of instances, we approximately have the same number of jobs in the simplified instance $I'$. Indeed, when $\varepsilon$ is big, we will have big classes (i.e., a few numbers of classes, each containing a lot of jobs). Thus, it is easy to find small jobs to merge (which means that the results are bad as the number of jobs is too big). On the other hand, as the value of $\varepsilon$ becomes small, we get small classes (i.e., a big number of classes, each containing few jobs). Therefore, the number of small jobs to merge is very limited, so, when using a small value for $\varepsilon$, instance $I'$ becomes very close to the original instance $I$ (see columns 4, 5, and 6 in Table 4.1). Thus, in the following, we will not present the results for $\varepsilon = 0.8$.

Table 4.1 Average number of jobs in the simplified instance $I'$ with $\varepsilon$ = 0.8, 0.4, 0.2 and 0.1

| #jobs | Average number of jobs in $I'$ | | | | I |
|---|---|---|---|---|---|
| | $\varepsilon = 0.8$ | $\varepsilon = 0.4$ | $\varepsilon = 0.2$ | $\varepsilon = 0.1$ | |
| 5-25 | 9.06 | 14.61 | 14.99 | 15.00 | 15.00 |
| 26-50 | 8.34 | 31.71 | 35.76 | 35.99 | 36.00 |
| 51-75 | 8.09 | 41.67 | 59.81 | 60.97 | 61.00 |
| 76-100 | 7.90 | 38.07 | 81.92 | 85.70 | 86.00 |
| 100-200 | 7.98 | 30.25 | 125.60 | 148.06 | 150.00 |

Tables 4.2, 4.3, and 4.4 present the results of our PTAS as a function of the number of jobs, for three different values of $\varepsilon$ (0.2, 0.4 and 0.1). Column 1 shows our five sets of instances. In columns 2 and 3, we present the size of Pareto front for the two algorithms (DP and PTAS). Column 4 presents the average number of jobs in the simplified instance $I'$. In column 5 and 6, we will provide the average of our PTAS results for the two objectives of our study: $L^P_{max}/L^*_{max}$ and $C^P_{max}/C^*_{max}$. Finally, in Column 7 we present the results of the HV ratios.

We can see that the small set of instances (5-25) have good quality because we do not lose a lot of solutions, and the instance $I'$ is very close to the original instance $I$. When $\varepsilon$ is small (see the tables 4.3 and 4.4), the PTAS algorithm finds solutions closer to the optimal ones when the number of jobs increases (which is not true for $\varepsilon = 0.4$).

It is worth-mentioning that the difference between the average approximate makespan and the average optimal value of this criterion is growing when the number of jobs increases. The other objective ($L_{max}$) has the opposite behavior. Indeed, with a lot of jobs, it is more likely to obtain very similar solutions, a lot of them being dominated by others. $C_{max}$ values are closer to the optimum than $L_{max}$ values, which is not a surprising, as $L_{max}$ depends on $C_{max}$.

***Remark***: We can see that $C^P_{max}/C^*_{max}$ can be smaller than one. This does not mean that PTAS is better than DP for the makespan objective, because this is only average value and the size of the Pareto front is smaller.

Table 4.2 Quality of our PTAS algorithm vs. number of jobs with $\varepsilon$= 0.4

| #jobs | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 1.67 | 14.61 | 9.99016E-01 | 1.00301E+00 | 99.34% |
| 26-50 | 5.44 | 1.77 | 31.67 | 9.99678E-01 | 1.00254E+00 | 98.35% |
| 51-75 | 4.13 | 1.31 | 41.67 | 9.99860E-01 | 1.00199E+00 | 99.46% |
| 76-100 | 2.96 | 1.34 | 38.07 | 9.99965E-01 | 1.00174E+00 | 99.44% |
| 100-200 | 2.55 | 1.27 | 30.25 | 9.99991E-01 | 1.00123E+00 | 99.17% |

Table 4.3 Quality of our PTAS algorithm vs. number of jobs with $\varepsilon$= 0.2

| #jobs | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 2.70 | 14.99 | 9.99675E-01 | 1.00083E+00 | 99.72% |
| 26-50 | 5.44 | 2.56 | 35.76 | 9.99782E-01 | 1.00105E+00 | 99.03% |
| 51-75 | 4.13 | 1.59 | 59.81 | 9.99881E-01 | 1.00089E+00 | 99.72% |
| 76-100 | 2.96 | 1.36 | 81.92 | 9.99965E-01 | 1.00075E+00 | 99.78% |
| 100-200 | 2.55 | 1.34 | 125.60 | 9.99991E-01 | 1.00044E+00 | 99.85% |

Table 4.4 Quality of our PTAS algorithm vs. number of jobs with $\varepsilon$= 0.1

| #jobs | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 3.41 | 15.00 | 9.99861E-01 | 1.00052E+00 | 99.92% |
| 26-50 | 5.44 | 3.61 | 35.99 | 9.99893E-01 | 1.00043E+00 | 99.62% |
| 51-75 | 4.13 | 2.33 | 60.97 | 9.99935E-01 | 1.00034E+00 | 99.88% |
| 76-100 | 2.96 | 1.84 | 85.70 | 9.99977E-01 | 1.00035E+00 | 99.88% |
| 100-200 | 2.55 | 1.53 | 148.06 | 9.99991E-01 | 1.00021E+00 | 99.93% |

In the following of this section, we will present the results of $\varepsilon = 0.2$, the analysis of the results are the same with $\varepsilon$ = 0.8, 0.4 and 0.1 (See Appendix A.1).

We have also studied the influence of processing time and delivery time in Table 4.5 and 4.6. The results are also given for our five sets of instances, from small instances (5-25 jobs) to bigger ones (100-200 jobs). The results showed that the instances having a wide range of job processing times are more difficult to solve to optimality. Indeed, with a wide range of processing times, it is easy to find small jobs to merge, so we lose some solutions (see column 4 in Table 4.5). Nevertheless, as we can see in columns 5 and 6, the average of $C_{max}$ is not really affected by processing times, but, $L_{max}$ is growing when the jobs have a small range of processing times. Furthermore, the delivery times have no real influence on the quality obtained by our PTAS algorithm.
In addition, the size of the Pareto front found by our algorithms is increasing with increasing ranges of processing times. But the delivery times have no real influence on the size of the Pareto fronts found by our PTAS.

Table 4.5 Quality of our PTAS algorithm vs. processing time ranges with $\varepsilon$= 0.2

| $p_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.34 | 68.34 | 9.99941E-01 | 1.00605E+00 | 99.94% |
| 1-100 | 3.99 | 1.90 | 68.24 | 9.99878E-01 | 1.00121E+00 | 99.31% |
| 1-500 | 5.25 | 2.36 | 68.04 | 9.99951E-01 | 1.00025E+00 | 96.96% |

* The size of the original instance $I = 75.5$.

Table 4.6 Quality of our PTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.2

| $q_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 1.65 | 65.75 | 9.99998E-01 | 1.00002E+00 | 99.21% |
| 1-100 | 2.97 | 1.60 | 69.23 | 9.99983E-01 | 1.00032E+00 | 98.44% |
| 1-500 | 6.59 | 2.34 | 69.65 | 9.99835E-01 | 1.00157E+00 | 98.56% |

* The size of the original instance $I = 75.5$.

**Average running times of our PTAS vs. DP**

In this subsection, the average running times are given in the figures 4.3 and 4.4. They compare the DP and PTAS algorithms, considering different values of $\varepsilon$ = 0.8, 0.4, 0.2 and 0.1. All values are in seconds.

Fig. 4.3 shows that all algorithms are slower when the number of jobs is growing. Moreover, the running time in PTAS decreases when the value of $\varepsilon$ grows, which is consistent with theory. Hence, as we have seen before, we can get a very good solution when $\varepsilon$ is small, but, as it can be seen in Fig. 4.3, this will take a lot of times. Therefore, we advise using a value that can achieve a good solution at a reasonable time, depending on the importance given to the quality required.

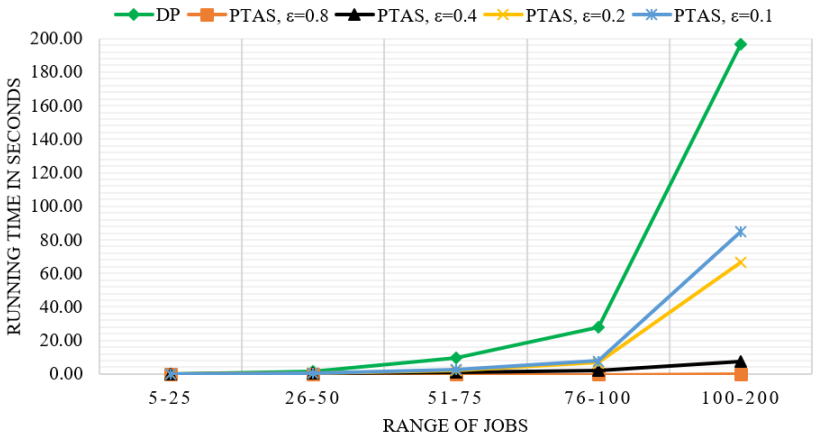
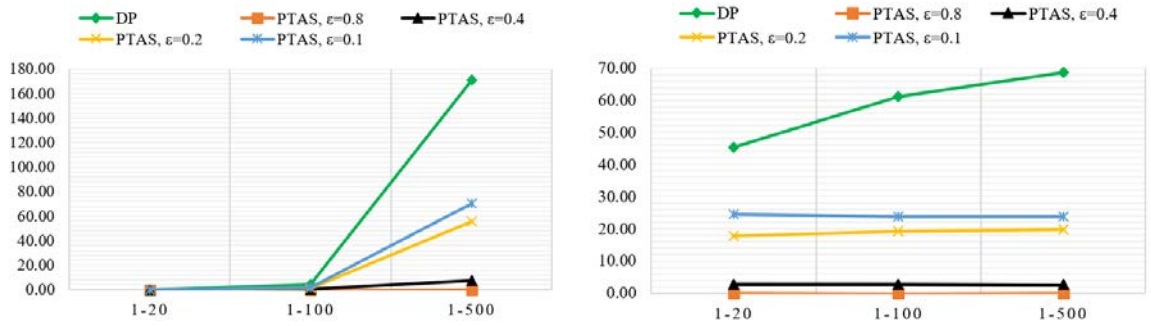



Fig. 4.3 PTAS: Average running times (s) vs. size of instances

We have also studied the average running times depending on processing and delivery times, in Fig. 4.4. The left part of this figure shows the average running times (s) vs. processing times found by the DP algorithm and our PTAS. We can see that all algorithms are slower when the processing time is growing.

On the right part of the same figure are given the average running times (s) vs. delivery times. As we can see, the delivery times have no real influence on the running times obtained by our algorithms: running times are growing, but slower than the delivery times ranges. This is not surprising, as the delivery times are not part of the complexity.

We can conclude that, the running time in PTAS decreases when the value of $\varepsilon$ grows, which is consistent with theory. As expected, our PTAS is faster than DP algorithm, especially with big values of $\varepsilon$. The running times are consistent with the complexity $O(n \cdot 2^{(\frac{8}{\varepsilon^2}+\frac{2}{\varepsilon})})$.




Running times vs. processing times

Running times vs. delivery times

Fig. 4.4 Average running times (s) vs. processing and delivery times ranges

#### 4.6.1.2 Results of our FPTAS algorithm vs. DP algorithm

**Quality of our FPTAS algorithm**

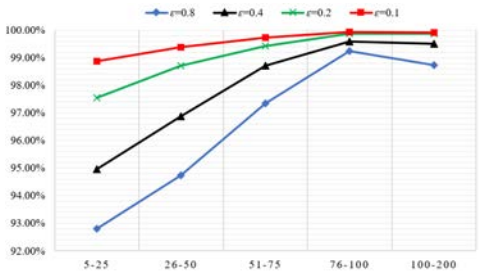
In this section, we will see the quality of our FPTAS using Hypervolume ratios. We will also present the results for the average values of $L_{max}$ and $C_{max}$.

Fig. 4.5 presents a comparison of our FPTAS and Dynamic Programming. As earlier, the results are given for our five sets of instances, from small instances (5-25 jobs) to bigger ones (100-200 jobs).

In Fig. 4.5(a), we can see that, with a small $\varepsilon$ value, the size of Pareto front obtained by our FPTAS algorithm decreases as the number of jobs increases, with big $\varepsilon$ value, we do not have the same behavior. Indeed, when $\varepsilon$ is big, the length of FPTAS intervals will be very big, so we remove some solutions. Hence, we can see that with good $\varepsilon$ (small $\varepsilon$), we have a good solution (See Fig. 4.5(b)), which is consistent with the theory. We can see that it is true for each set of instances, with $\varepsilon = 0.1$ we have the best solutions.

| #jobs | DP | FPTAS | | | |
|---|---|---|---|---|---|
| | | $\varepsilon = 0.8$ | $\varepsilon = 0.4$ | $\varepsilon = 0.2$ | $\varepsilon = 0.1$ |
| 5-25 | 4.74 | 1.47 | 1.56 | 1.64 | 1.80 |
| 26-50 | 5.44 | 1.56 | 1.59 | 1.50 | 1.72 |
| 51-75 | 4.13 | 1.59 | 1.67 | 1.42 | 1.55 |
| 76-100 | 2.96 | 1.53 | 1.58 | 1.32 | 1.33 |
| 100-200 | 2.55 | 1.54 | 1.58 | 1.30 | 1.24 |

A. Size of Pareto Front




B. Hypervolume Indicator ($HV_r$)

Fig. 4.5 Quality of our FPTAS algorithm with $\varepsilon = 0.8, 0.4, 0.2$ and $0.1$

***Remark***: We can see that $\varepsilon = 0.8$ is not very bad (this is because we are not using the merging technique as we did in PTAS, so we do not lose jobs) as we have seen in the previous subsection 4.6.1.1. However, it still has a less quality than $\varepsilon$ = 0.4, 0.2 and 0.1.

In the remainder of this section, we will present the results of $\varepsilon = 0.2$, the analysis of the results are the same with $\varepsilon$ = 0.8, 0.4 and 0.1 (See Appendix B.1).

As we can see in Table 4.7, as a function of a number of jobs, we have the same previous behavior of the PTAS in the section 4.6.1.1. The FPTAS can find solutions closer to the optimal ones when the number of jobs increases, while $C_{max}$ and $L_{max}$ decreases.

Table 4.7 Quality of our FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.2

| #jobs | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 1.64 | 1.00074E+00 | 1.00366E+00 | 97.55% |
| 26-50 | 5.44 | 1.50 | 1.00020E+00 | 1.00197E+00 | 98.71% |
| 51-75 | 4.13 | 1.42 | 1.00008E+00 | 1.00120E+00 | 99.42% |
| 76-100 | 2.96 | 1.32 | 1.00008E+00 | 1.00159E+00 | 99.88% |
| 100-200 | 2.55 | 1.30 | 1.00002E+00 | 1.00069E+00 | 99.87% |

We have also studied the influence of processing time and delivery time in Tables 4.8 and 4.9. As previously, our results are given for the four sets of instances, from small instances (5-25 jobs) to bigger ones (100-200 jobs). As in PTAS, the instances having a wide range of job processing times are more difficult to solve to optimality. Indeed, with a wide range of processing times, in the FPTAS we can remove too many states. Moreover, delivery times have the same behavior as processing times. In the previous section, we have seen that the delivery times have no influence on the quality obtained by the PTAS. This because in the PTAS, we split the maximum value of delivery times into equal-length classes and round up every $q_j$.
In addition, processing times have no real influence on the size of the Pareto front given by our FPTAS. However, as a function of delivery times, the size of the Pareto front found by our algorithms is increasing with increasing ranges of delivery times.

Table 4.8 Quality of our FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.2

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.27 | 1.00016E+00 | 1.00736E+00 | 99.96% |
| 1-100 | 3.99 | 1.57 | 1.00008E+00 | 1.00152E+00 | 99.71% |
| 1-500 | 5.25 | 1.44 | 1.00008E+00 | 1.00081E+00 | 99.24% |

Table 4.9 Quality of our FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.2

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 1.13 | 1.00008E+00 | 1.00024E+00 | 99.99% |
| 1-100 | 2.97 | 1.35 | 1.00008E+00 | 1.00080E+00 | 99.65% |
| 1-500 | 6.59 | 1.79 | 1.00009E+00 | 1.00251E+00 | 99.27% |

**Average running times of our FPTAS vs. DP**

In this section, the average running times are given in the figures 4.6 and 4.7. They compare our DP and FPTAS, considering different values of $\varepsilon$ = 0.8, 0.4, 0.2 and 0.1. All values are in milliseconds.

As a function of the number of jobs, as shown in Fig. 4.6, our FPTAS is slower when the number of states is growing. Hence, the results are consistent with PTAS.

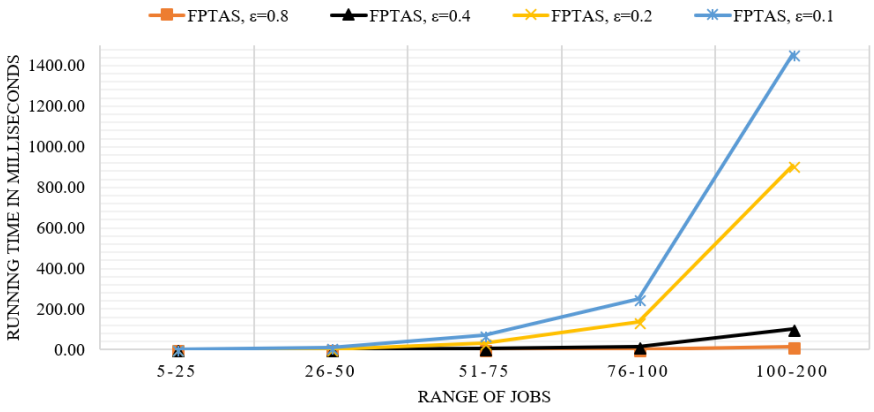
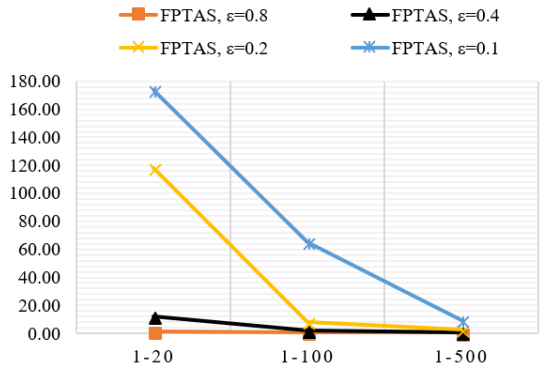



Fig. 4.6 FPTAS: Average running times (ms) vs. size of instances

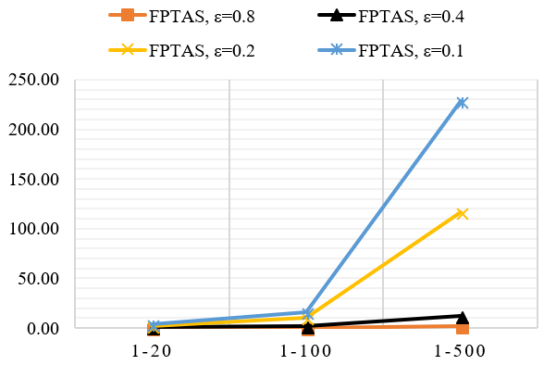
We have also studied the average running times of processing and delivery times, in Fig. 4.7. The left part of this figure shows the average running times (ms) vs. processing times found by our FPTAS. We can see that the FPTAS is going faster when the processing time is growing. Indeed, in the DP algorithm, we generate all possible solutions, while in the FPTAS we remove a special part of the states generated by the DP algorithm. With a wide range of processing times, the FPTAS can remove too many states.
On the right part of the same figure are given the average running times (ms) vs. delivery times. As we can see, the FPTAS is slower when the delivery times are growing.

***Remark***: As DP goes to 196,822 milliseconds, we did not include it here, because the FPTAS running time is too small (the average running times of DP can be seen in Fig. 4.4).

We can conclude that, the running time in FPTAS decreases when the value of $\varepsilon$ grows, which is consistent with theory. As expected, our FPTAS is faster than the DP algorithm, especially with big values of $\varepsilon$. The running times are consistent with the complexity $O(n^3/\varepsilon^2)$.




Running times vs. processing times




Running times vs. delivery times

Fig. 4.7 Average running times (ms) vs. processing and delivery times ranges

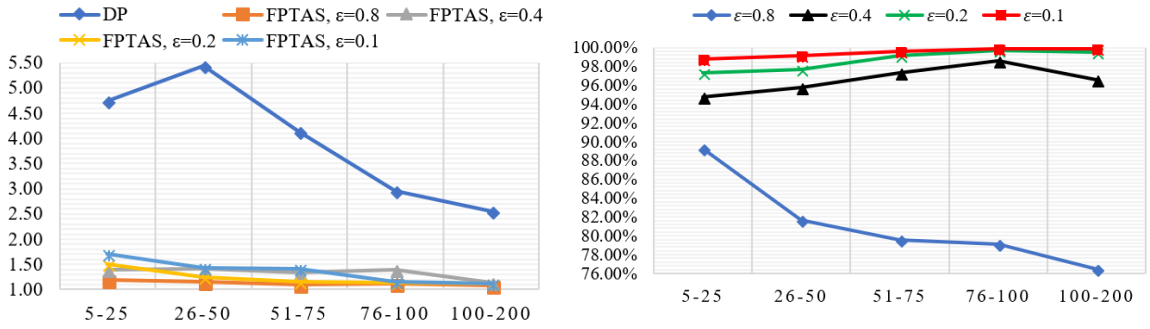
#### 4.6.1.3 Results of the improved FPTAS vs. the DP algorithm

**Quality of the improved FPTAS**

In this section, we will analyze the quality of the improved FPTAS by using Hypervolume ratios. We will also present the results for the average values of $L_{max}$ and $C_{max}$.

Fig. 4.8 presents a comparison of the improved FPTAS and the DP. As earlier, the results are given for our five sets of instances, from small instances (5-25 jobs) to bigger ones (100-200 jobs).

In Fig. 4.8(a), we can see that the number of solutions obtained by our algorithm decreases as the number of jobs increases. With a lot of jobs, it is more likely to obtain very similar solutions, a lot of them being dominated by others. Moreover, as we can see in Fig. 4.8(b), the improved FPTAS algorithm achieves better results with small $\varepsilon$ values than large values, which is consistent with the theory. We can also see that, the results of $\varepsilon = 0.8$ are very bad. Indeed, these results are consistent with the PTAS, which is not surprising, since we use the same simplified instances.




A. Size of Pareto Front

B. Hypervolume Indicator (HV)

Fig. 4.8 Quality of the improved FPTAS algorithm with $\varepsilon = 0.8, 0.4, 0.2$ and $0.1$

In the remainder of this section, we will provide the results of $\varepsilon = 0.2$, the analysis of the results are the same with $\varepsilon$ = 0.8, 0.4 and 0.1 (See Appendix C.1).

As we can see in Table 4.10, as a function of the number of jobs, the improved FPTAS has the same analysis of the PTAS results in section 4.6.1.1.

Table 4.10 Quality of the improved FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.2

| #jobs | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 1.50 | 1.00052E+00 | 1.00468E+00 | 97.32% |
| 26-50 | 5.44 | 1.25 | 9.99960E-01 | 1.00445E+00 | 97.63% |
| 51-75 | 4.13 | 1.15 | 9.99949E-01 | 1.00249E+00 | 99.17% |
| 76-100 | 2.96 | 1.14 | 1.00001E+00 | 1.00198E+00 | 99.70% |
| 100-200 | 2.55 | 1.13 | 9.00882E-01 | 9.02024E-01 | 99.55% |

We have also studied the influence of processing time and delivery time in Tables 4.11 and 4.12. As in previous sections, the results are given for our five sets of instances, from small instances (5-25 jobs) to largest ones (100-200 jobs).

As in PTAS and FPTAS algorithms, the instances having a wide range of job processing times are more difficult to solve to optimality. Delivery times have no influence on the quality obtained by our improved FPTAS (as in PTAS). The average of $C_{max}$ and $L_{max}$ are not affected by processing and delivery times. In addition, as a function of processing times and delivery times, the results of the Pareto front given by our improved FPTAS are consistent with the FPTAS (i.e., processing times have no real influence on the size of the Pareto front given by our improved FPTAS, however, delivery times have an influence: the size of the Pareto front increases with increased ranges of delivery times).

Table 4.11 Quality of the improved FPTAS vs. processing time ranges with $\varepsilon$= 0.2

| $p_i$ ranges | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.13 | 5.32410E+00 | 4.76996E+00 | 99.99% |
| 1-100 | 3.99 | 1.34 | 1.52963E+00 | 1.52877E+00 | 99.67% |
| 1-500 | 5.25 | 1.21 | 7.01523E-01 | 7.03113E-01 | 97.77% |

Table 4.12 Quality of the improved FPTAS vs. delivery time ranges with $\varepsilon$= 0.2

| $q_i$ ranges | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 1.09 | 1.05652E+00 | 1.05728E+00 | 99.33% |
| 1-100 | 2.97 | 1.25 | 8.73247E-01 | 8.75161E-01 | 98.59% |
| 1-500 | 6.59 | 1.33 | 1.07094E+00 | 1.07280E+00 | 99.51% |

**Average running times of our improved FPTAS vs. DP**

In this section, the average running times are given in the figures 4.9 and 4.10. They compare our DP algorithm and improved FPTAS, considering different values of $\varepsilon$ = 0.8, 0.4, 0.2 and 0.1. All values are in milliseconds.

Fig. 4.9 shows that our improved FPTAS algorithm is slower when the number of states is growing. Hence, these results are consistent with the properties of PTAS and FPTAS.

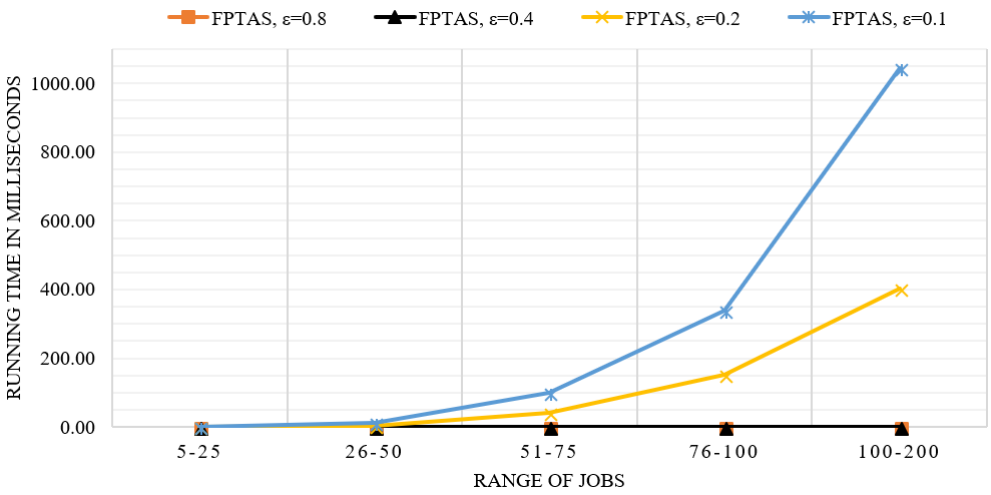
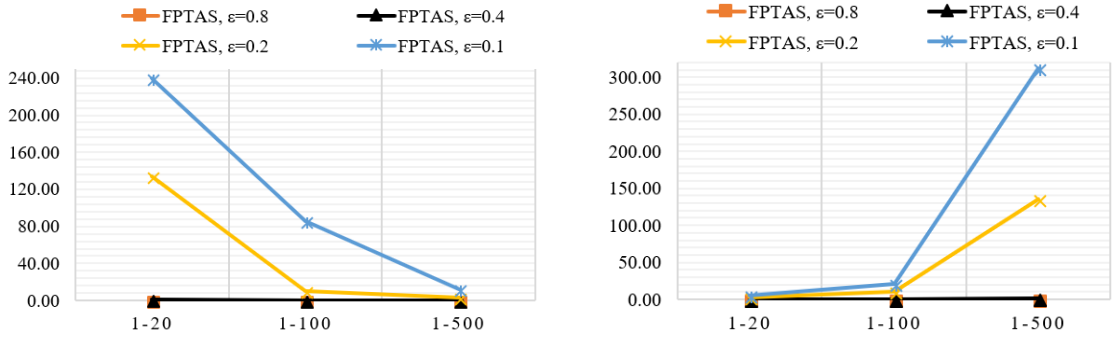



Fig. 4.9 Improved FPTAS: Average running times (ms) vs. size of instances

We have also studied the average running times of processing and delivery times, in Fig.4.10. Indeed, the analysis of the results is the same as in the section 4.6.1.2 (see Fig.4.7).




Running times vs. processing times

Running times vs. delivery times

Fig. 4.10 Average running times (ms) vs. processing and delivery times ranges

We can conclude that the running time of the improved FPTAS decreases when the value of $\varepsilon$ grows, which is consistent with theory. As expected, the improved FPTAS is faster than the DP algorithm, especially with big values of $\varepsilon$.

### 4.6.2 Big instances

We also tested bigger instances. We have randomly generated 36 instances, with different numbers of jobs and various processing and delivery times:

- number of jobs: from 900 to 1000;
- processing times: from 1 to 20 and 1 to 100;
- delivery times: from 1 to 20, 1 to 100 and 1 to 500;

***Remark:*** in small instances, we also had a processing time of 1 to 500, but with this number of jobs it was too slow. Thus, we remove them.

As expected, the results are consistent with the ones of smaller instances. Indeed, as we can see in Tables 4.13, 4.14 and 4.15, our FPTAS still managed to find very good solutions. Moreover, the size of the Pareto front (i.e., the number of solutions) found by our PTAS and improved FPTAS algorithms decreases when $\varepsilon$ value grows. The FPTAS has an opposite behavior.

Table 4.13 Quality of our PTAS with 1000 jobs

| $\varepsilon$ values | Size of Pareto front | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
|---|---|---|---|---|
| 0.8 | 1.056 | 1.03298E+00 | 1.03397E+00 | 71.25% |
| 0.4 | 1.278 | 1.00032E+00 | 1.00108E+00 | 99.33% |
| 0.2 | 1.389 | 1.00000E+00 | 1.00026E+00 | 99.95% |
| 0.1 | 1.667 | 1.00000E+00 | 1.00008E+00 | 99.99% |

***Remark:*** Size of Pareto front for DP is: 2.055.

Table 4.14 Quality of our FPTAS with 1000 jobs

| $\varepsilon$ values | Size of Pareto front | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
|---|---|---|---|---|
| 0.8 | 1.222 | 1.00001E+00 | 1.00028E+00 | 99.98% |
| 0.4 | 1.056 | 1.00001E+00 | 1.00003E+00 | 99.99% |
| 0.2 | 1.056 | 1.00000E+00 | 1.00028E+00 | 99.99% |
| 0.1 | 1.028 | 1.00000E+00 | 1.00005E+00 | 99.99% |

Table 4.15 Quality of our improved FPTAS with 1000 jobs

| $\varepsilon$ values | Size of Pareto front | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
|---|---|---|---|---|
| 0.8 | 1.000 | 1.03570E+00 | 1.03735E+00 | 69.46% |
| 0.4 | 1.039 | 1.00197E+00 | 1.00307E+00 | 97.65% |
| 0.2 | 1.028 | 1.00001E+00 | 1.00056E+00 | 99.77% |
| 0.1 | 1.056 | 1.00000E+00 | 1.00038E+00 | 99.93% |

The average running times are given in Table 4.16. The results of running times are consistent with the results of smaller instances. Noteworthy, for big instances, since the running times are quite big, they have been expressed in seconds. As we can see, the algorithms achieve small running times when the value of $\varepsilon$ is big, which is consistent with the theory. Furthermore, the improved FPTAS has better running times compared with the PTAS and FPTAS algorithms.

Table 4.16 Average running times (seconds) with $\varepsilon$ = 0.8, 0.4, 0.2 and 0.1

| DP time | $\varepsilon$ value | PTAS time | FPTAS Time | Improved FPTAS Time |
|---|---|---|---|---|
| 2345.04 | 0.8 | 0.001 | 4.80 | 0.00 |
| | 0.4 | 16.28 | 27.21 | 0.00 |
| | 0.2 | 222.54 | 127.47 | 0.03 |
| | 0.1 | 1496.76 | 221.15 | 82.90 |

In addition, the results of processing and delivery times are given in the tables 4.17, 4.18, 4.19, 4.20, 4.21 and 4.22 (the analysis of the results are the same with $\varepsilon$=0.8, 0.4 and 0.1, See Appendices A.2, B.2 and C.2). Moreover, the results of running times are consistent with the results of smaller instances.

Table 4.17 Results of our PTAS for big instances as a function of processing time ranges with $\varepsilon = 0.2$

| $p_i$ range | PTAS Size of Pareto front | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ | DP time(s) | PTAS time(s) |
|---|---|---|---|---|---|---|
| 1-20 | 1.500 | 1.00000E+00 | 1.00098E+00 | 99.97% | 144.51 | 12.85 |
| 1-100 | 1.278 | 1.00000E+00 | 1.00010E+00 | 99.57% | 4545.57 | 432.24 |

Table 4.18 Results of our FPTAS for big instances as a function of processing time ranges with $\varepsilon = 0.2$

| $p_i$ range | FPTAS Size of Pareto front | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ | DP time(s) | FPTAS time(s) |
|---|---|---|---|---|---|---|
| 1-20 | 1.056 | 1.00000E+00 | 1.00128E+00 | 99.98% | 144.51 | 223.83 |
| 1-100 | 1.056 | 1.00000E+00 | 1.00007E+00 | 99.92% | 4545.57 | 31.10 |

Table 4.19 Results of our improved FPTAS for big instances as a function of processing time ranges with $\varepsilon = 0.2$

| $p_i$ range | FPTAS Size of Pareto front | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ | DP time(s) | FPTAS time(s) |
|---|---|---|---|---|---|---|
| 1-20 | 1.056 | 4.63684E+00 | 4.63784E+00 | 100.00% | 144.51 | 0.02 |
| 1-100 | 1.000 | 2.15671E-01 | 2.15968E-01 | 99.99% | 4545.57 | 0.04 |

Table 4.20 Results of our PTAS for big instances as a function of delivery time ranges with $\varepsilon = 0.2$

| $q_i$ range | PTAS Size of Pareto front | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ | DP time(s) | PTAS time(s) |
|---|---|---|---|---|---|---|
| 1-20 | 1.583 | 1.00000E+00 | 1.00001E+00 | 99.97% | 2081.30 | 272.81 |
| 1-100 | 1.417 | 1.00000E+00 | 1.00010E+00 | 99.83% | 2145.23 | 131.10 |
| 1-500 | 1.167 | 1.00000E+00 | 1.00066E+00 | 99.52% | 2808.60 | 263.72 |

Table 4.21 Results of our FPTAS for big instances as a function of delivery time ranges with $\varepsilon = 0.2$

| $q_i$ range | FPTAS Size of Pareto front | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ | DP time(s) | FPTAS time(s) |
|---|---|---|---|---|---|---|
| 1-20 | 1.000 | 1.00000E+00 | 1.00000E+00 | 100.00% | 2081.30 | 8.40 |
| 1-100 | 1.083 | 1.00000E+00 | 1.00014E+00 | 99.99% | 2145.23 | 18.35 |
| 1-500 | 1.083 | 1.00000E+00 | 1.00070E+00 | 99.86% | 2808.60 | 355.64 |

Table 4.22 Results of our improved FPTAS for big instances as a function of delivery time ranges with $\varepsilon = 0.2$

| $q_i$ range | FPTAS Size of Pareto front | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ | DP time(s) | FPTAS time(s) |
|---|---|---|---|---|---|---|
| 1-20 | 1.083 | 1.00000E+00 | 1.00000E+00 | 100.00% | 2081.30 | 0.019 |
| 1-100 | 1.000 | 1.00000E+00 | 1.00014E+00 | 100.00% | 2145.23 | 0.012 |
| 1-500 | 1.000 | 1.00000E+00 | 1.00070E+00 | 99.98% | 2808.60 | 0.052 |

### 4.6.3 Comparison of algorithm results

#### Quality of our algorithms

**Dependency on the number of jobs:** as we can see in Fig. 4.11, with big instances, the FPTAS has better quality than the PTAS and the improved FPTAS. But, with small instances,

as mentioned in Section 4.6.1, the PTAS achieves better results as we decrease the number of jobs.

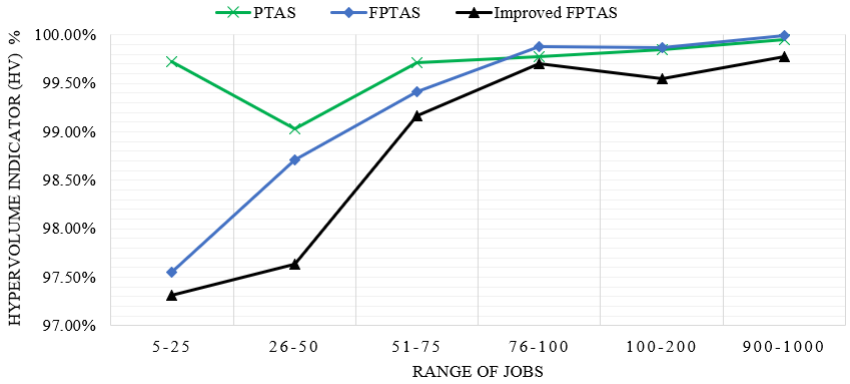



Fig. 4.11 Quality of our algorithms depending on the number of jobs with $\varepsilon$=0.2

**Dependency on processing times:** with big instances, the improved FPTAS always has better quality than PTAS and FPTAS (see Section 4.6.2). However, for small instances, with big processing times, the FPTAS has a good quality. Otherwise, the improved FPTAS is the better (see Fig. 4.12).

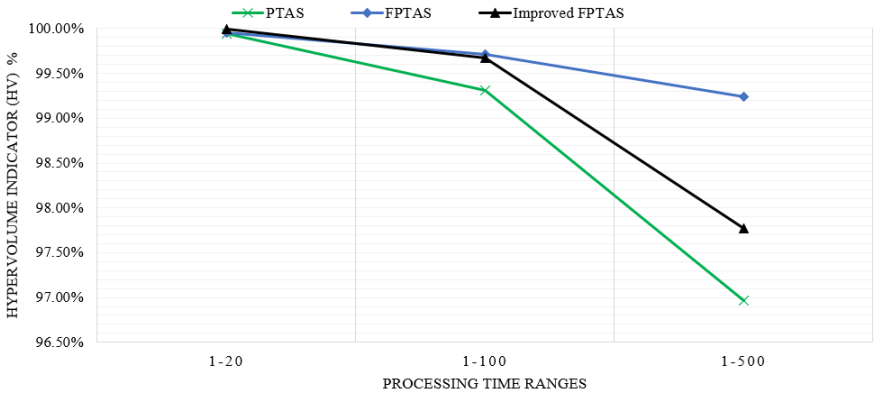



Fig. 4.12 Quality of our algorithms depending on processing times with $\varepsilon$=0.2

**Dependency on delivery times:** with big instances, the improved FPTAS always has better quality than the PTAS and FPTAS. However, for small instances, with small delivery times, the FPTAS has a good quality. Otherwise, the improved FPTAS is the better (see Fig. 4.13).

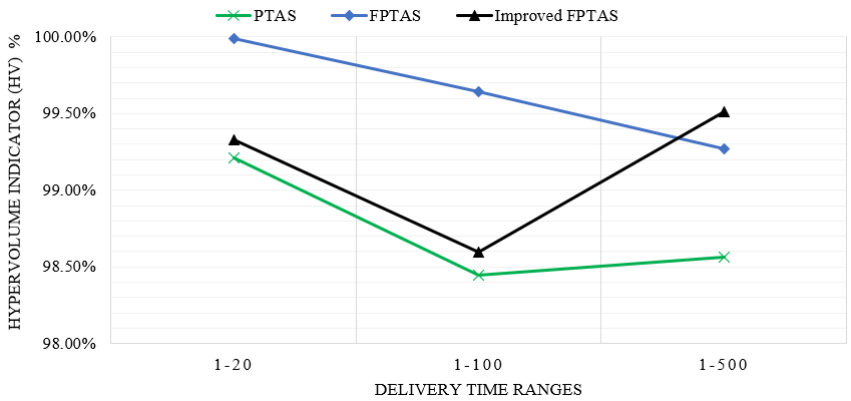



Fig. 4.13 Quality of our algorithms depending on delivery times with $\varepsilon$=0.2

**Average running times of our algorithms**

In general, as we have seen in Section 4.6.2, with big instances, the average running times given by our improved FPTAS is better than DP, PTAS, and FPTAS.

For small instances, as we have seen in the sections 4.6.1.1, 4.6.1.2, and 4.6.1.3, when $\varepsilon$ is too big, the average running times given by the improved FPTAS is smaller than the DP, PTAS, and FPTAS.

Indeed, regarding the dependency on the number of jobs, all algorithms are slower when the number of jobs increases. Hence, when $\varepsilon$ is big, the improved FPTAS is faster than other algorithms. On the other side, when $\varepsilon$ is small, the FPTAS is better, but, the difference is not big.

Moreover, regarding the dependency on processing times, when the PTAS algorithm becomes slower, the FPTAS and improved FPTAS have an opposite behavior and vice versa.

In addition, regarding the dependency on delivery times, when $\varepsilon$ is big, the improved FPTAS is faster than other algorithms. On the other side, when $\varepsilon$ and delivery times are smaller, the FPTAS is better, but, the difference is not too big.

## 4.7 Conclusions and Perspectives

The two-parallel machines scheduling problem has been considered to minimize the maximum lateness and the makespan. We have proposed an exact algorithm (based on dynamic algorithm) to generate the complete Pareto Frontier in a pseudo-polynomial time. Then, we present a PTAS and an FPTAS to solve the problem. Furthermore, the improvement used in the PTAS are exploited to improve the FPTAS. For the proposed algorithms, we randomly

generated several instances with different ranges, and, for each job $j$, its processing time $p_j$ and delivery time $q_j$ are set to be integer numbers. The results of the experiments showed that the proposed algorithms for the considered problem are very efficient, especially for big instances composed of a lot of jobs. It is clear that optimizing the maximum lateness ($L_{max}$) implies to minimize implicitly the makespan ($C_{max}$). Moreover, we demonstrated the importance of the values of processing times, delivery times, and $\varepsilon$ values.

In our future works, the study of the multiple-machine scheduling problems seems to be a challenging perspective of our work.

# Bibliography


[1] V. Th. Paschos (2018). Combinatorial approximation of maximum k-vertex cover in bipartite graphs within ratio 0.7. *RAIRO - Operations Research*, 52:305–314.

[2] Z. Geng and J. Yuan (2018). Scheduling with or without precedence relations on a serial-batch machine to minimize makespan and maximum cost. *Applied Mathematics and Computation*, 332:1-18.

[3] C. M. Fonseca, M. López-Ibáñez, L. Paquete, and A. P. Guerreiro (2018). Computation of the Hypervolume Indicator. http://lopez-ibanez.eu/hypervolume.

[4] S. B Amor, K. Zaras, and E. A. Aguayo (2017). The value of additional information in multicriteria decision making choice problems with information imperfections. *Annals of Operations Research*, 253(1):61–76.

[5] S. B Amor and J. M. Martel (2014). A new distance measure including the weak preference relation: Application to the multiple criteria aggregation procedure for mixed evaluations. *European Journal of Operational Research*, 237(3):1165–1169.

[6] I. Kacem and H. Kellerer (2014). Approximation algorithms for no-idle time scheduling on a single machine with release dates and delivery times. *Discrete Applied Mathematics*, 164:154–160.

[7] Y. Chen and X. Zou (2014). Runtime analysis of a multi-objective evolutionary algorithm for obtaining finite approximations of Pareto fronts. *Information Sciences*, 262:62–77.

[8] L. Bradstreet (2011). The hypervolume indicator for multi-objective optimisation: calculation and use. *University of Western Australia*.

[9] P. Kumar (2008). A Framework for Multi-objective Optimization and Multi-criteria Decision Making for Design of Electrical Drives. *Universiteit Delft*.

[10] M. Demange and V. T. Paschos (2005). Polynomial approximation algorithms with performance guarantees: An introduction-by-example. *European Journal of Operational Research*, 165(3):555–568.

# Chapter 5

# Approximation Schemes for Scheduling Jobs on $m$-Parallel Machine to Minimize the Maximum Lateness and Makespan

*We consider a multiobjective scheduling problem on identical machines, with two criteria to minimize: the maximum lateness and the makespan. We assume the number of machines is a constant. This chapter proposes an exact algorithm (based on a dynamic programming algorithm) to generate the complete Pareto Frontier in a pseudo-polynomial time. Moreover, four heuristics (PH, JOH, RJOH and LPTH) have been proposed in order to optimize our DP algorithm. Then, we present a PTAS (Polynomial Time Approximation Scheme) to generate an approximate Pareto Frontier. In this scheme, we use a simplification technique based on the merging of jobs (exploited in the previous chapter). Furthermore, we present an FPTAS (Fully Polynomial Time Approximation Scheme) to generate an approximate Pareto Frontier, based on the conversion of the dynamic programming. The proposed FPTAS algorithm is strongly polynomial. Finally, some numerical experiments are provided in order to compare the proposed approaches.*

## 5.1 Introduction

In this chapter we deal with a multiobjective scheduling problem, when the number of machines is fixed. We study the join minimization of two criteria: the maximum lateness and the makespan. Formally, the problem is a generalization of the one studied in the previous chapter and it is defined as follows. A set $J$ of $n$ jobs has to be processed on $m \geq 2$ identical machines. Each job $j \in J$ has a processing time $p_j$ and a delivery time $q_j$. The number of

machines is fixed and each of them cannot execute more than one job at a given time. The machines are ready to work at time $t = 0$. The problem is to determine a sequence of jobs, that minimizes the two criteria: the maximum lateness $L_{max}$ and the makespan $C_{max}$. We assume that all the inputs are integers. Moreover, the jobs are sorted in non-increasing order of their delivery times (the same order as the Jackson's order): $q_1 \geq q_2 \geq \ldots \geq q_n$.

Given the objective of this thesis, we recall some representative works for the scheduling problem on $m$ machines. Many objective functions have been considered in the literature when studying this problem. For example, Alon et al. [9] considered the scheduling problem on identical machines to find a schedule that optimizes an objective function that solely depends on the machine completion times. They showed that there are polynomial time approximation schemes of the scheduling problem. In [1], Lin et al. proposed two heuristics and a genetic algorithm for the problem of scheduling jobs on an unrelated parallel machine to minimize three objectives (makespan, total weighted completion time, and total weighted tardiness). They showed that their heuristics are efficient and provide solutions with reasonable quality. Moreover, they showed that their proposed genetic algorithm outperforms other algorithms in terms of the number of non-dominated solutions and the quality of their solutions. Angel et al.[8] studied scheduling problem with a set of unrelated machines to find a schedule, obtaining a trade-off between the makespan and the total cost when the number of machines is a constant. The authors presented an FPTAS with a time complexity of $O(n(n/\varepsilon)^m)$. The scheduling problem with release dates, due dates and sequence-dependent setup times on an identical parallel machine have been considered in [4], to minimize the makespan and the total tardiness. The researchers showed the strong NP-hardness of the problem. Thus, they developed a metaheuristic method and an exact method, to solve the problem of their study. In [5], Chang and Tong proposed a dominance theorem and a lower bound to accelerate the branch-and-bound algorithm, in order to optimize the criterion of the makespan for m-machine permutation flowshop scheduling problem with learning considerations. The researchers adapted four well-known existing heuristic algorithms to solve their studied problem. Furthermore, they showed that the branch-and-bound algorithm can solve the problem within a reasonable period of time, and the heuristic algorithms are extremely accurate with an average error percentage of less than 0.1%. Su [6] presented a heuristic and a branch-and-bound algorithm for a parallel machine scheduling problem with job deadlines and machine eligibility constraints, with the total completion time as a criterion. They showed that the lower bound improves the performance of those in the literature in terms of average CPU time. Moreover, they showed that the heuristic generates a good quality schedule. The parallel machine scheduling problem has been considered in [7],

with the minimum machine completion time as a criterion. The researchers provided lower bounds and ordinal algorithms to solve the problem. Lee et al. [3] presented the state of art in online scheduling problem on parallel machine environments with machine eligibility constraints, in order to minimize the makespan. Huo and Zhao [2] studied two different models for the bi-criteria scheduling problems. They considered the total completion time minimization, subject to the machine non-availability constraints and the makespan is at most a constant T. They showed that there is an optimal polynomial time algorithm for each model studied.

The rest of this chapter is organized as follows. In Section 5.2, we describe the proposed dynamic programming algorithm. The proposed heuristics will be presented in Section 5.3. Section 5.4, presents some of the results obtained using our DP algorithm and heuristics. Section 5.5 provides the description and the analysis of the PTAS based on the merging technique. In Section 5.6 is described our proposed FPTAS. The results of our PTAS and FPTAS will be presented in Section 5.7. Finally, Section 5.8 concludes the chapter.

## 5.2 Dynamic Programming Algorithm

The following dynamic programming algorithm $A$ can be applied to solve exactly this problem. This algorithm $A$ generates iteratively some sets of states. At each iteration $j$, a set $\chi_j$ composed of states is generated $(0 \leq j \leq n)$. Each state $[t_1, t_2, ..., t_m, f]$ in $\chi_j$ can be associated with a feasible partial schedule for the first $j$ jobs. Here, variables $(t_1, t_2, ..., t_m)$ represent the completion times of jobs scheduled in the last positions on the machines, and variable $f$ is the maximum lateness of the corresponding partial schedule. The dynamic programming algorithm can be described as follows.

**Algorithm** $A$

1. Set $\chi_1 = \{[t_1 = p_1, t_2 = 0, ..., t_m = 0, p_1 + q_1]\}$.
2. For $j \in \{2, 3, ..., n\}$,
   (a) $\chi_j = \emptyset$.
   (b) For every state $[t_1, t_2, ..., t_m, f]$ in $\chi_{j-1}$.
      i. Add $[t_1, ..., t_k + p_j, t_{(k+1)}, ..., t_m, max\{f, t_k + p_j + q_j\}]$ to $\chi_j$, avoiding symmetric solutions (for every $k = \{1, 2, ..., m\}$).
   (c) For every $(t_1, t_2, ..., t_m)$ (*i.e., for all equivalent partial solutions*): keep only the state with the smallest possible $f$.

(d) Remove $\chi_{j-1}$.

3. Return the Pareto front of $\chi_n$, by only keeping the non-dominated states.

The standard version of this dynamic programming has an exponential complexity since it needs $O(m^n)$ time. Nevertheless, this complexity can be reduced to $O(n.P^m)$ by keeping only one state having the smallest value of $f$, for each iteration $j \in \{2,...,n\}$. Therefore, algorithm $A$ can be reduced to a pseudo-polynomial algorithm, which proves at the same time that the problem is NP-hard only in the ordinary sense.

### 5.2.1 Illustrative Example

In this section, we provide a description of the Dynamic Programming algorithm applied to a small example. The instance we use is described in Table 5.1: A set $J$ of five jobs must be performed on three machines, each job with its processing time $(1 \leq p_j \leq 100)$ and its delivery time $(1 \leq q_j \leq 20)$.

Table 5.1 An instance with five jobs

| Jobs$(J_j)$ | Processing time $(p_j)$ | Delivery time $(q_j)$ |
|---|---|---|
| 1 | 41 | 18 |
| 2 | 12 | 15 |
| 3 | 95 | 11 |
| 4 | 56 | 6 |
| 5 | 95 | 5 |

Recall that in our DP algorithm, as described in Section 5.2, a solution is given by building partial solutions, stored in $\chi_j$, consisting in scheduling $j$ jobs. The sets $\chi_j$ contain states $[t_1,t_2,...,t_m,f]$, where, $t_1,t_2,...,t_m$ are the completion times of the machines, and $f$ denotes the maximum lateness value of the corresponding schedule. The final solution is given by the Pareto front of $\chi_n$, by only keeping non-dominated states.

**First Schedule.** To break symmetry, the first job is assigned to machine $M_1$, leading to $\chi_1 = \{(41,0,0,59)\}$ (see Fig. 5.1).

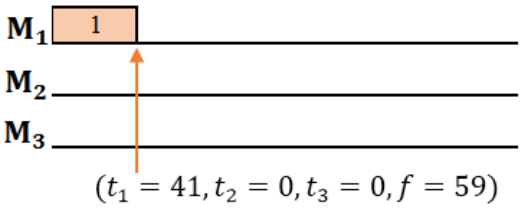



Fig. 5.1 Partial solution after Job 1 has been scheduled

**Second Step.** The second job is scheduled alternatively on machine $M_1$ and $M_2$, not on machine $M_3$ as it would lead to two symmetric solutions $(41,12,0,59)$ and $(41,0,12,59)$. Therefore, this will lead to two partial solutions in $\chi_2 = \{(53,0,0,68),(41,12,0,59)\}$ (see Fig. 5.2).

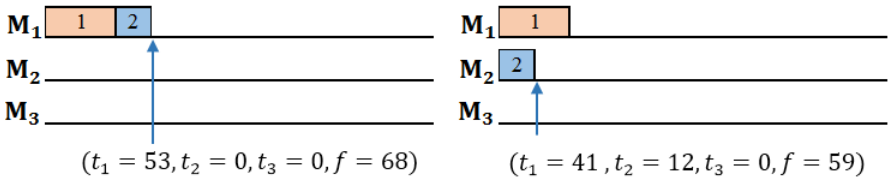



Fig. 5.2 Partial solutions after 2 jobs have been scheduled

**Third Step.** The third job is scheduled alternatively on machines $M_1$, $M_2$ and $M_3$, completing each partial schedule of $\chi_2$. This gives new partial solutions in $\chi_3 = \{(148,0,0,159),$ $(53,95,0,106),(136,12,0,147),(41,107,0,118),(41,12,95,106)\}$ (see Fig. 5.3).

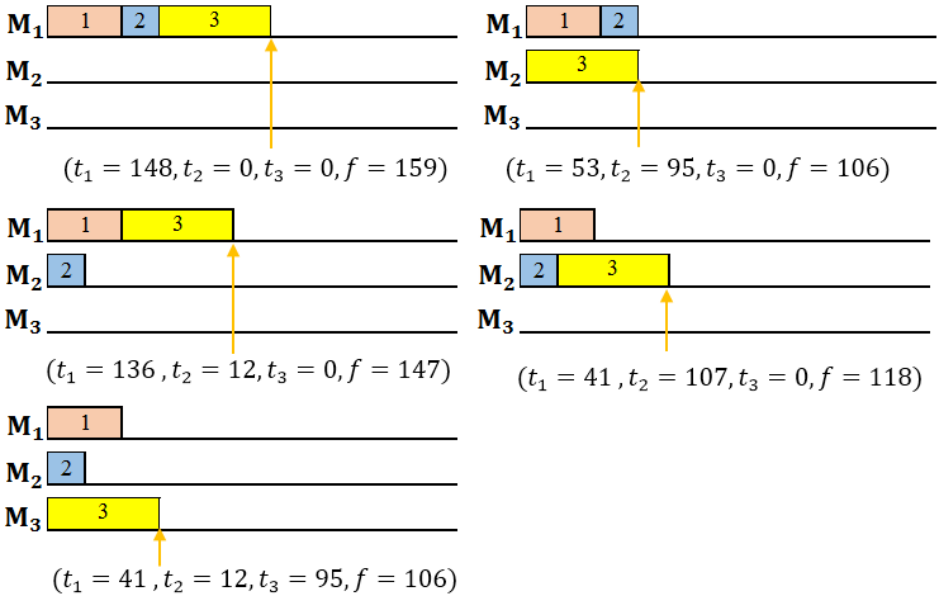



Fig. 5.3 Partial solutions after 3 jobs have been scheduled

**Fourth Step.** The fourth job is scheduled alternatively on machines $M_1$, $M_2$ and $M_3$, completing each partial schedule of $\chi_3$. This gives new partial solutions in $\chi_4 = \{(204,0,0,210), (148,56,0,159), (109,95,0,115), (53,151,0,157), (53,95,56,106), (192,12,0,198), (136,68,0,147), (136,12,56,147), (97,107,0,118), (41,163,0,169), (41,107,56,118), (97,12,95,106), (41,68,95,106), (41,12,151,157)\}$ (see Fig. 5.4).

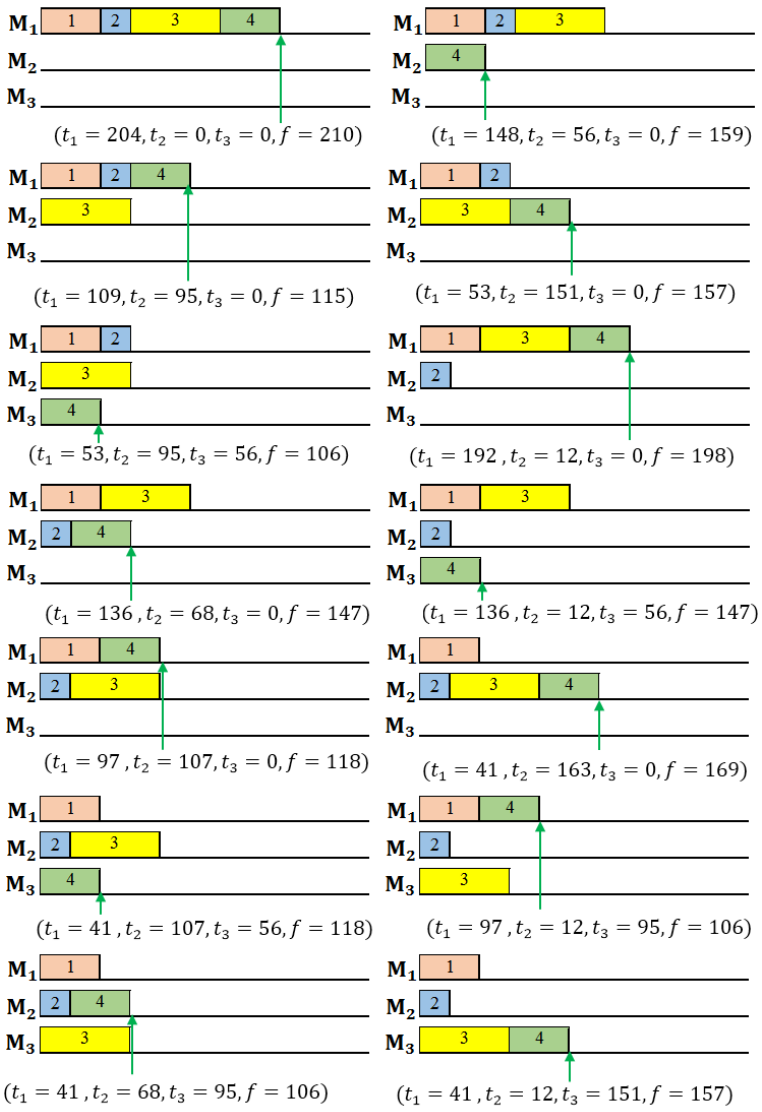



Fig. 5.4 Partial solutions after 4 jobs have been scheduled

**Fifth (Final) Step.** After having scheduled the last job (job 5), we obtained 41 solutions $(\{(136,107,56,141), (41,12,246,251), (287,12,0,292), (136,107,56,147), (136,68,95,147), (243,56,0,248), (53,95,151,156), (204,95,0,210), (109,95,95,115), (204,95,0,209), (136,68,95,141), (53,190,56,195), (231,68,0,236), (136,12,151,156), (53,246,0,251),$

$(136,12,151,157),(97,12,190,195),(136,163,0,168),(136,163,0,169),(41,107,151,157)$, $(41,107,151,156),(299,0,0,304),(97,107,95,112),(192,107,0,198),(41,202,56,207)$, $(148,56,95,159),(41,68,190,195),(192,107,0,197),(41,163,95,169),(41,163,95,168)$, $(41,258,0,263),(97,107,95,118),(109,190,0,195),(148,95,56,153),(192,12,95,197)$, $(97,202,0,207),(231,12,56,236),(148,151,0,157),(53,151,95,157),(148,151,0,159)$, $(192,12,95,198)\}$). Within this set, we have some states that have the same $t$ values, so we can remove some states, to keep only the state with the smallest possible $f$.
Hence, based on:

- (136, 107, 56, 141) we remove: (136, 107, 56, 147).
- (136, 68, 95, 141) we remove: (136, 68, 95, 147).
- (204, 95, 0, 209) we remove: (204, 95, 0, 210).
- (136, 163, 0, 168) we remove: (136, 163, 0, 169).
- (136, 12, 151, 156) we remove: (136, 12, 151, 157)
- (41, 107, 151, 156) we remove: (41, 107, 151, 157).
- (192, 107, 0, 197) we remove: (192, 107, 0, 198).
- (41, 163, 95, 168) we remove: (41, 163, 95, 169).
- (97, 107, 95, 112) we remove: (97, 107, 95, 118).
- (148, 151, 0, 157) we remove: (148, 151, 0, 159).
- (192, 12, 95, 197) we remove: (192, 12, 95, 198).

Then, the new set is composed of 30 states.
Finally, the last set of partial solutions will be: $\chi_5 = \{(97,107,95,112)\}$, after keeping only the non-dominated states.

## 5.3 Optimized Dynamic Programming Algorithm

In the following subsections, we will introduce some heuristics/rules to optimize our DP. In these improvements, the idea is to quickly find a bound that will be used not to consider solutions that are dominated by this bound.

### 5.3.1 First improving rule (PH)

Here, we are using an upper bound on $C_{max}$, which can be computed as follows.

$$\frac{\sum_{i=1}^{n} p_i}{m} + \max_i p_i$$

At the beginning of the second step $(2.b)$ of the DP algorithm $A$, we have to check if the completion time of the job occupying the last position on machine $k$ is lower than this bound. If not, we do not add this solution to the current set of partial solutions. As a consequence, the modified algorithm $A^H$ will be as follows.

**Algorithm** $A^H$

1. Set $\chi_1 = \{[t_1 = p_1, t_2 = 0, ..., t_m = 0, p_1 + q_1]\}$.
2. For $j \in \{2, 3, ..., n\}$,
   (a) $\chi_j = \emptyset$.
   (b) For every state $[t_1, t_2, ..., t_m, f]$ in $\chi_{j-1}$.
      i. If $(t_k + p_j \leq \frac{\sum_{i=1}^{n} p_i}{m} + \max_i p_i)$
      Add $[t_1, ..., t_k + p_j, t_{(k+1)}, ..., t_m, max\{f, t_k + p_j + q_j\}]$ to $\chi_j$, avoiding symmetric solutions (for every $k = \{1, 2, ..., m\}$).
   (c) For every $(t_1, t_2, ..., t_m)$ (*i.e., for all equivalent partial solutions*): keep only the state with the smallest possible $f$.
   (d) Remove $\chi_{j-1}$.
3. Return the Pareto front of $\chi_n$, by only keeping non-dominated states.

In the following subsections, we will try to find better heuristics. Indeed, heuristic PH only gives a bound on $C_{max}$, not on $L_{max}$. In the next paragraphs, we will propose three heuristics, which will also be integrated at the beginning of the second step $(2.b)$ of the DP algorithm $A^H$.

### 5.3.2 Second improvement: Heuristic JOH

The bound obtained by this heuristic is computed by taking jobs one by one, and scheduling them on the less loaded machine. As before, the jobs are initially sorted using Jackson's order.

### 5.3.3 Third improvement: Heuristic RJOH

Here, we try to add a perturbation to the previous heuristic: the first job on each machine is randomly selected, then the remaining jobs are added on the less loaded machine according to Jackson's order.

### 5.3.4 Fourth improvement: Heuristic LPTH

This heuristic is quite different from the previous ones, as the scheduling is obtained in two steps:

- In a first step, the jobs are sorted according to the Longest Processing Time (LPT) rule. The jobs are taken one by one and scheduled on the less loaded machine.
- In a second step, this schedule is modified machine by machine, reordering jobs according to Jackson's order.

### 5.3.5 Illustrative Example

To illustrate our proposed heuristics, we consider the same example presented in the Subsection 5.2.1 on three machines.

#### 5.3.5.1 Results of the first improving rule (PH)

According to this heuristic (see 5.3.1), $\frac{\sum_{i=1}^{n} p_i}{m} + \max_i p_i \Longrightarrow 195$. Therefore, the jobs will be scheduled as follows.

**First Schedule.** To break symmetry, the first job is assigned to machine $M_1$, leading to $\chi_1 = \{(41,0,0,59)\}$.

**Second Step.** The second job is scheduled alternatively on machine $M_1$ and $M_2$, leading to two partial solutions in $\chi_2 = \{(53,0,0,68),(41,12,0,59)\}$. As we can see in $\chi_2$, all solutions have been added (i.e., no solution has been removed), because all values of $t$ are less than the heuristic bound (i.e., 195).

**Third Step.** The third job is scheduled alternatively on machines $M_1$, $M_2$ and $M_3$, completing each partial schedule of $\chi_2$. This gives 5 new partial solutions in $\chi_3 = \{(148,0,0,159),$ $(53,95,0,106),(136,12,0,147),(41,107,0,118),(41,12,95,106)\}$. Also, all solutions have been added.

**Fourth Step.** The fourth job is scheduled alternatively on machines $M_1$, $M_2$ and $M_3$, completing each partial schedule of $\chi_3$. This gives 13 new partial solutions in $\chi_4 = \{(148,56,0,159),(109,95,0,115),(53,151,0,157),(53,95,56,106),(192,12,0,198),(136,68,0,147),(136,12,56,147),(97,107,0,118),(41,163,0,169),(41,107,56,118),(97,12,95,106),(41,68,95,106),(41,12,151,157)\}$. At this step, the heuristic allowed us not to add one solution. Indeed, starting from previous state $(148,0,0,159)$, scheduling job 4 would lead to the state $(204,0,0,210)$, which $C_{max}$ is greater than the bound given by PH (i.e., 195).

**Fifth (Final) Step.** After having scheduled the last job (job 5), we have 19 new partial solutions in $\chi_5 = \{(136,107,56,141),(41,68,190,195),(97,107,95,112),(148,95,56,153),(53,95,151,156),(136,12,151,156),(136,68,95,141),(41,163,95,168),(136,163,0,168),(53,190,56,195),(41,107,151,156),(109,190,0,195),(192,12,95,197),(97,12,190,195),(53,151,95,157),(192,107,0,197),(148,151,0,157),(148,56,95,159),(109,95,95,115)\}$. Note that there are 20 solutions that have not been added. Finally, the last set of partial solutions: $\chi_5 = \{(97,107,95,112)\}$, after keeping only the non-dominated states.

#### 5.3.5.2 Results of the second improvement: Heuristic (JOH)

The bound given by the heuristic is: $(t,f) = (136,141)$ (see Fig. 5.5, which depicts the schedule obtained by applying this heuristic).

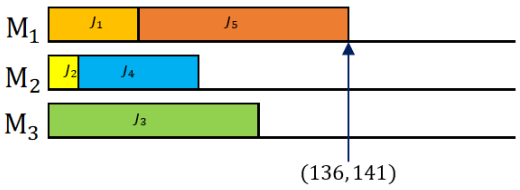



Fig. 5.5 Illustration of the second heuristic

Now, before adding a partial solution, we must check whether this solution is better than the one given by the heuristic bound $(136,141)$. If not, we do not add this solution. Thus, the jobs will be scheduled as follows.

**First Schedule.** To break symmetry, the first job is assigned to machine $M_1$, leading to $\chi_1 = \{(41,0,0,59)\}$.

**Second Step.** The second job is scheduled alternatively on machine $M_1$ and $M_2$, leading

to two partial solutions in $\chi_2 = \{(53,0,0,68),(41,12,0,59)\}$. As we can see, all solutions have been added as they are not dominated by the bound given by heuristic JOH $(136,141)$.

**Third Step.** The third job is scheduled alternatively on machines $M_1$, $M_2$ and $M_3$, completing each partial schedule of $\chi_2$. This gives 4 new partial solutions in $\chi_3 = \{(53,95,0,106),$ $(136,12,0,147),(41,107,0,118),(41,12,95,106)\}$. Note that the solution $(148,0,0,159)$ has not been added, because this solution is dominated by the bound given by heuristic JOH $(136,141)$.

**Fourth Step.** The fourth job is scheduled alternatively on machines $M_1$, $M_2$ and $M_3$, completing each partial schedule of $\chi_3$. This gives 8 new partial solutions in $\chi_4 = \{(109,95,0,115),$ $(53,95,56,106),(136,68,0,147),(136,12,56,147),(97,107,0,118),(41,107,56,118),$ $(97,12,95,106),(41,68,95,106)\}$. Thanks to our heuristic, the solutions $\{(53,151,0,157),$ $(192,12,0,198),(41,163,0,169),(41,12,151,157)\}$ have not been added.

**Fifth (Final) Step.** After having scheduled the last job (job 5), we have 4 new partial solutions in $\chi_5 = \{(136,107,56,141),(97,107,95,112),(136,68,95,141),(109,95,95,115)\}$. At this step, 20 solutions have not been added. Finally, the last set of partial solutions will be: $\chi_5 = \{(97,107,95,112)\}$, after keeping only the non-dominated states.

#### 5.3.5.3 Results of the third improvement: Heuristic (RJOH)

According to our third heuristic (see 5.3.3), the first job on each machine should be randomly selected. So, suppose the first scheduling jobs are:$J_3, J_1, J_4$. Hence, the bound given by the heuristic is: $(t,f) = (148,153)$ (see Fig. 5.6, which depicts the schedules obtained by applying this heuristic).

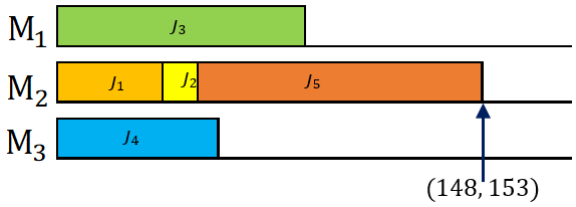



Fig. 5.6 Illustration of the third heuristic

As previously, if a partial solution is not dominated by the bound provided by RJOH, then we add it to the current set of partial solutions, otherwise we don't. Hence, the jobs will

be scheduled as follows.

**First Schedule.** To break symmetry, the first job is assigned to machine $M_1$, leading to $\chi_1 = \{(41,0,0,59)\}$.

**Second Step.** The second job is scheduled alternatively on machine $M_1$ and $M_2$, leading to two partial solutions in $\chi_2 = \{(53,0,0,68),(41,12,0,59)\}$. As we see in $\chi_2$, all solution has been added.

**Third Step.** The third job is scheduled alternatively on machines $M_1$, $M_2$ and $M_3$, completing each partial schedule of $\chi_2$. This gives 5 new partial solutions in $\chi_3 = \{(148,0,0,159)$, $(53,95,0,106),(136,12,0,147),(41,107,0,118),(41,12,95,106)\}$. Again, all solutions have been added.

**Fourth Step.** The fourth job is scheduled alternatively on machines $M_1$, $M_2$ and $M_3$, completing each partial schedule of $\chi_3$. This gives 9 new partial solutions in $\chi_4 = \{(148,56,0,159)$, $(109,95,0,115),(53,95,56,106),(136,68,0,147),(136,12,56,147),(97,107,0,118)$, $(41,107,56,118),(97,12,95,106),(41,68,95,106)\}$. Note that 5 solutions were not added $\{(204,0,0,210),(53,151,0,157),(192,12,0,198),(41,163,0,169),(41,12,151,157)\}$ because these solutions are dominated by the bound given by heuristic RJOH.

**Fifth (Final) Step.** After having scheduled the last job, we have 6 new partial solutions in $\chi_5 = \{(136,107,56,141),(97,107,95,112),(148,95,56,153),(148,56,95,159)$, $(136,68,95,141),(109,95,95,115)\}$. Note that there are 21 solutions that have not been added. Finally, the last set of partial solutions is: $\chi_5 = \{(97,107,95,112)\}$, after keeping only the non-dominated states.

#### 5.3.5.4 Results of the fourth improvement: Heuristic LPTH

The jobs sorted following LPT rule are: $J_3, J_5, J_4, J_1$ and $J_2$. The first step of this heuristic leads to the schedule depicted by Fig. 5.7. The obtained bound is $(t,f) = (107,122)$. During the second step of this heuristic, the schedule is modified machine by machine, reordering jobs according to Jackson's order (see Fig. 5.8). This leads to a better bound $(t,f) = (107,118)$.

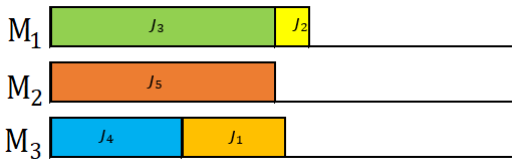



Fig. 5.7 Illustration of the first step of LPTH

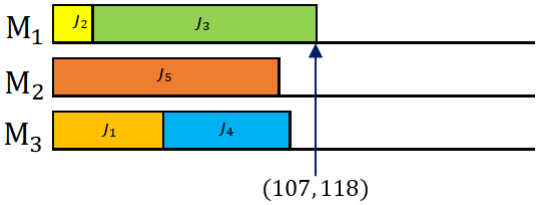



Fig. 5.8 Illustration of the fourth heuristic

Then, the jobs will be scheduled as follows.

**First Schedule.** To break symmetry, the first job is assigned to machine $M_1$, leading to $\chi_1 = \{(41,0,0,59)\}$.

**Second Step.** The second job is scheduled alternatively on machine $M_1$ and $M_2$, leading to two partial solutions in $\chi_2 = \{(53,0,0,68),(41,12,0,59)\}$. All solutions have been added.

**Third Step.** The third job is scheduled alternatively on machines $M_1$, $M_2$ and $M_3$, completing each partial schedule of $\chi_2$. This gives 3 new partial solutions in $\chi_3 = \{(53,95,0,106),$ $(41,107,0,118),(41,12,95,106)\}$. Noteworthy, the solutions $(136,12,0,147)$ and $(148,0,0,159)$ have not been added because they are dominated by the bound given by heuristic LPTH.

**Fourth Step.** The fourth job is scheduled alternatively on machines $M_1$, $M_2$ and $M_3$, completing each partial schedule of $\chi_3$. This gives 6 new partial solutions in $\chi_4 = \{(109,95,0,115),$ $(53,95,56,106),(97,107,0,118),(41,107,56,118),(97,12,95,106),(41,68,95,106)\}$. The solutions $\{(53,151,0,157),(41,163,0,169),(41,12,151,157)\}$ have not been added.

**Fifth (Final) Step.** After having scheduled the last job, we have two new partial solutions in $\chi_5 = \{(97,107,95,112),(109,95,95,115)\}$. It should be noted that there are 16 solutions

that have not been added. Finally, the last set of partial solutions is: $\chi_5 = \{(97,107,95,112)\}$, after keeping only the non-dominated states.

In this example, heuristics JOH, RJOH and LPTH are better than heuristic PH. Moreover, the LPTH heuristic is better than the other ones. As we can see in the last column of Table 5.2, for every step of the algorithm, the heuristic leads to smaller set of partial solutions than the other heuristics. Thus, it is faster. Furthermore, the RJOH heuristic is not efficient in this example, as it does not lead to a better bound than JOH, but in other examples, it can be more efficient than JOH. In the next Section 5.4, we will present some results for our heuristics.

Table 5.2 Size of solutions in the set of partial solutions, for the illustrative example 5.3.5

| #Set | Size of Partial Solutions Set | | | | |
|---|---|---|---|---|---|
| | DP | PH | JOH | RJOH | LPTH |
| Heuristic value | N/A | 195 | (136, 141) | (148, 153) | (107, 118) |
| $\chi_1$ | 1 | 1 | 1 | 1 | 1 |
| $\chi_2$ | 2 | 2 | 2 | 2 | 2 |
| $\chi_3$ | 5 | 5 | 4 | 5 | 3 |
| $\chi_4$ | 14 | 13 | 8 | 9 | 6 |
| $\chi_5$ | 30 | 19 | 4 | 6 | 2 |

## 5.4 Results of The DP Algorithm and Heuristics

The following results have been obtained after testing the quality of the DP algorithm and the heuristics. The code has been done in Java and the experiments were performed on an Intel(R) Core(TM)-i7 with 8GB RAM. To ensure the consistency of running times, each test has been run three times. Noteworthy, we have randomly generated four sets of instances, with various processing and delivery times:

- number of jobs: 5, 10, 15 and 20;
- processing times : from 1 to 20 and 1 to 100;
- delivery times : from 1 to 20, 1 to 100 and 1 to 500.

For each set of parameters, we have generated three different instances, which leads to 18 instances in each set of instances. The results have been tested on three machines and five machines.

In the previous Chapter 4, we also considered jobs with wider processing time (i.e., 1 to 500), but our DP algorithm did not achieve to find solutions within one hour. Thus, we did not consider these instances in this study.

In the following Subsections 5.4.1 and 5.4.2, we will present the results of three machines ($M = 3$) and five machines ($M = 5$).

## 5.4.1 Results on three machines

In this subsection, we present the results on three machines. First, we will present the results of the DP algorithm in Subsection 5.4.1.1 , then the results of the heuristics in Subsection 5.4.1.2 .

### 5.4.1.1 Results of the DP algorithm

Table 5.3 presents the results of our dynamic programming algorithm dependency on the number of jobs. Column 1 shows our four sets of instances. In column 2, we present the size of Pareto front. Columns 3 to 4 provide the average of our two studied objectives: $C^*_{max}$ and $L^*_{max}$. Finally, column 5 presents the average running times in seconds.

As we can see in column 2, the size of the Pareto front (i.e., the number of solutions) obtained by our DP algorithm increases as the number of jobs increases. Moreover, as we can see in columns 3 and 4, the average of $C^*_{max}$ and $L^*_{max}$ grows with the increase in the number of jobs. The average running times also grows (see column 5).

Table 5.3 Results of our DP algorithm vs. number of jobs (on three machines)

| #jobs | Size of Pareto front | $C^*_{max}$ | $L^*_{max}$ | Running Times (s) |
|---|---|---|---|---|
| 5 | 2.06 | 60.94 | 220.83 | 0.00 |
| 10 | 5.56 | 108.72 | 256.89 | 0.16 |
| 15 | 6.11 | 162.56 | 280.78 | 424.22 |
| 20 | 11.06 | 205.28 | 314.72 | 4152.75 |

We have also studied the influence of processing time and delivery time in the tables 5.4 and 5.5.

As we can see in Table 5.4, the size of the Pareto front increases when considering jobs having a wide range of processing times. When the processing time is low, a lot of solutions are similar. Therefore, the size of the Pareto front is relatively reduced compared to the high processing time. It should be noted that increased processing times increases the value of completion times. As they are more solutions, our DP algorithm had a lot of states to consider at each step, which explains the big difference of running times. When we consider a wider range of processing times (see column 5).

Table 5.4 Results of our DP algorithm vs. processing time ranges (on three machines)

| $p_i$ ranges | Size of Pareto front | $C^*_{max}$ | $L^*_{max}$ | Running Times (s) |
|---|---|---|---|---|
| 1-20 | 3.69 | 46.78 | 208.72 | 3.89 |
| 1-100 | 8.69 | 221.97 | 327.89 | 2284.67 |

Table 5.5 shows the results as a function of delivery times. As we can see in column 2, the size of the Pareto front for the jobs having a medium range of delivery times (i.e., 1 to 100) is quite big compared to the other ones. Indeed, with a medium range of delivery times, we can get many non-dominated solutions (i.e., we do not lose many solutions), leading to an increase in the size of the Pareto front. Hence, as we can see in the columns 3 and 4, the average of $C^*_{max}$ and $L^*_{max}$ grows with a wide range of delivery times. The average running times also grows (see column 5).

Table 5.5 Results of our DP algorithm vs. delivery time ranges (on three machines)

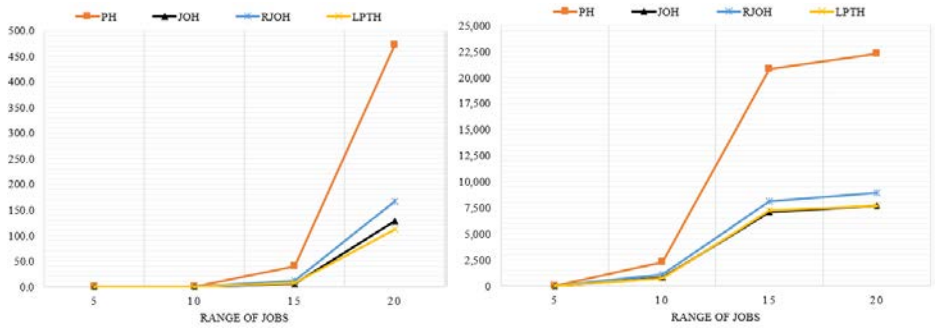
| $q_i$ ranges | Size of Pareto front | $C^*_{max}$ | $L^*_{max}$ | Running Times (s) |
|---|---|---|---|---|
| 1-20 | 5.13 | 131.46 | 136.79 | 951.93 |
| 1-100 | 7.33 | 134.96 | 179.58 | 1120.50 |
| 1-500 | 6.13 | 136.71 | 488.54 | 1360.42 |

#### 5.4.1.2 Results of heuristics

We have also studied the efficiency of our heuristics.

#### 5.4.1.2.1 As a function of the number of jobs

Fig. 5.9 presents a comparison of our heuristics as a function of the number of jobs. On these two figures (A and B), we can observe that: with small jobs, all the heuristics have the same average running times, and when increasing the number of jobs, the LPTH is faster than other proposed heuristics (see Fig. 5.9. (A)). We can also see that the size of the partial solutions set (PSS) grows with the increase in the number of jobs (see Fig. 5.9. (B)), which explains the average running times.




(A) Average running times (s) (B) Size of solutions in the PSS

Fig. 5.9 Results of heuristics vs. size of instances

As LPTH heuristic is faster than the other ones, we now present the efficiency of this heuristic vs. JOH. It should be noted that each heuristic finds one bound, which is used to reduce the complexity of our algorithm. The question is does a better bound will lead to reduced computing time?

For example, as we can see in Fig. 5.10 we have three cases. First case, LPTH finds a better bound than JOH (see Fig. 5.10 (A)). The second case in Fig. 5.10 (B), we can see that JOH finds a better bound than LPTH, and the last case in Fig. 5.10 (C) no one dominates.

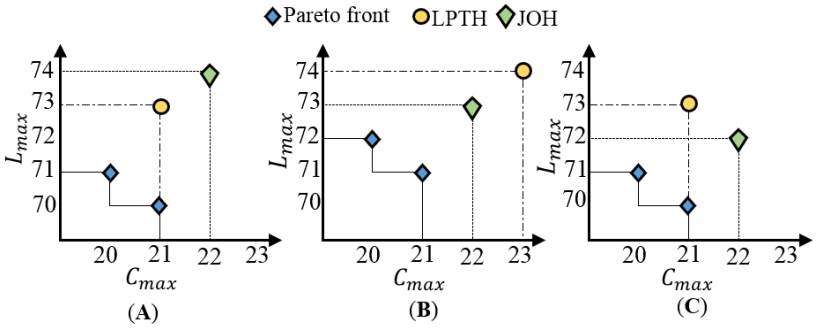



Fig. 5.10 Illustrative example for the results in Table 5.6

Table 5.6 shows the results of the efficiency of heuristic LPTH vs. JOH. We can see that LPTH is a little better than JOH (see columns 2 and 3), but in more than 56% (see column 4), we have two non-dominated solutions.

Table 5.6 Efficiency of heuristic LPTH vs. JOH as a function of the number of jobs

| #Jobs | LPTH is better | JOH is better | No one dominates |
|---|---|---|---|
| 5 | 17% | 0% | 83% |
| 10 | 44% | 0% | 56% |
| 15 | 28% | 0% | 72% |
| 20 | 39% | 0% | 61% |

We could conclude that LPTH is better than JOH. Indeed, as we see in Table 5.7, LPTH generally finds a better bound on $C_{max}$ and $L_{max}$, while JOH generally finds a better bound on $L_{max}$.

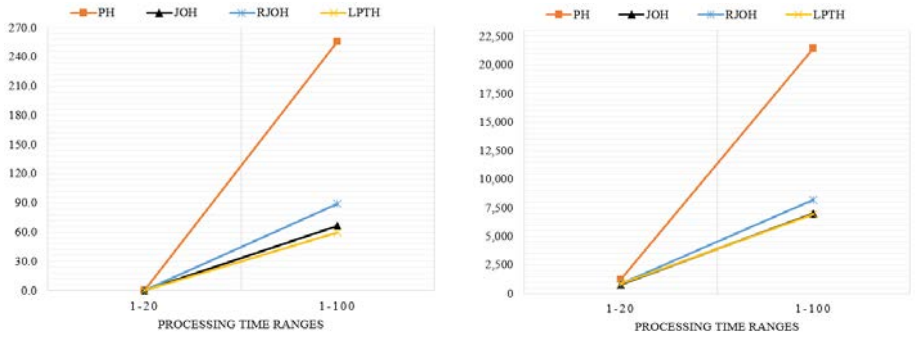
Table 5.7 Efficiency of heuristic LPTH vs. JOH for $C_{max}$, $L_{max}$ and both $C_{max}$ & $L_{max}$

| #jobs | LPTH is better for... | | | JOH is better for... | | |
|---|---|---|---|---|---|---|
| | $C_{max}$ | $L_{max}$ | $C_{max}$ and $L_{max}$ | $C_{max}$ | $L_{max}$ | $C_{max}$ and $L_{max}$ |
| 5 | 72% | 17% | 17% | 0% | 22% | 0% |
| 10 | 94% | 44% | 44% | 0% | 11% | 0% |
| 15 | 94% | 28% | 28% | 0% | 39% | 0% |
| 20 | 94% | 39% | 39% | 0% | 33% | 0% |

*Remark.* As a function of processing and delivery times, the results of the efficiency are consistent with the results as a function of the number of jobs (i.e., LPTH is better than JOH).

**5.4.1.2.2 As a function of processing and delivery times**

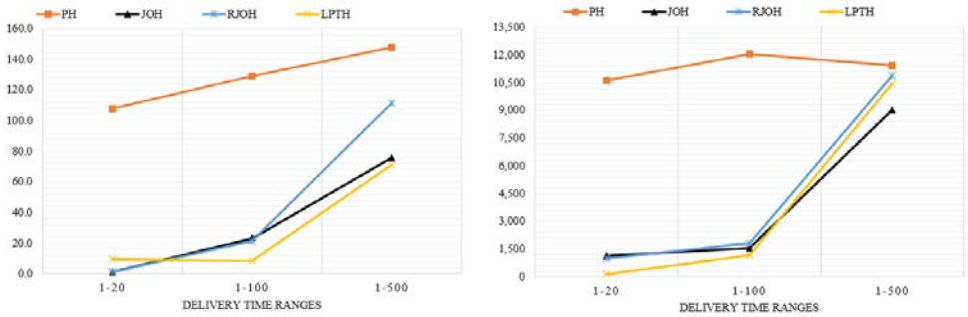
In Fig. 5.11 we present the results of our heuristics as a function of processing times. We can observe that: with a wide range of processing times, the average running times of our heuristics are increasing because we do not lose many solutions. Moreover, the LPTH is always faster than other proposed heuristics (see Fig. 5.11. (A)). We can also see that the size of the partial solutions set (PSS) increasing with a wide range of processing times (see Fig. 5.11. (B)), which explains the average running times.




(A) Average running times (s) (B) Size of solutions in the PSS

Fig. 5.11 Results of heuristics vs. processing time ranges

Fig. 5.12 presents the results of our heuristics as a function of delivery times, as we can see in Fig. 5.12. (A), once again the LPTH is faster than other proposed heuristics. Moreover, all heuristics become slower with a wide range of delivery times. Indeed, as we can see in Fig.5.12. (B), the size of the partial solutions set (PSS) increases with the increasing range of delivery times, which explains the average running times.




(A) Average running times (s) (B) Size of solutions in the PSS

Fig. 5.12 Results of heuristics vs. delivery time ranges

### 5.4.2 Results on five machines

In this subsection, we present the results on five machines. The results of the DP algorithm will be presented in Subsection 5.4.2.1, and the results of the heuristics will be in Subsection 5.4.2.2. The results are given for our two sets of instances, from 5 to 10 jobs.

#### 5.4.2.1 Results of the DP algorithm

Table 5.8 present the results of our DP dependency on the number of jobs.

Column 1 shows our two sets of instances. In column 2, we present the size of Pareto front for the considered machine (M= 3). Columns 3 to 4, provide the average of our two study objectives of our study: $C^*_{max}$ and $L^*_{max}$. Finally, column 5 presents the average running times in seconds.

As expected, the results are consistent with our previous results for three machines. Clearly, as we can see in column 2, the size of the Pareto front obtained by our DP algorithm increases as the number of jobs increases. Moreover, the average of $C^*_{max}$ and $L^*_{max}$ will also grow as we increase the number of jobs (see the columns 3 and 4). The average running times also grows (see column).

Table 5.8 Results of our DP algorithm vs. number of jobs (on five machines)

| #jobs | Size of Pareto front | $C^*_{max}$ | $L^*_{max}$ | Running Times (s) |
|---|---|---|---|---|
| 5 | 4.89 | 157.63 | 306.22 | 0.00 |
| 10 | 43.22 | 253.70 | 407.96 | 50.39 |

*Remark.* It should be noted that increasing the number of machines will increase the opportunities for scheduling jobs in the other machines, resulting in more solutions (i.e., a large size of Pareto front) and small value of $C_{max}$ and of course $L_{max}$ which depends on $C_{max}$. For these reasons, $M = 5$ has a large size of Pareto front compared with $M = 3$. In addition, for the two objectives of our study: the average value of $C^*_{max}$ and $L^*_{max}$ when $M = 5$ is always smaller than the average value when $M = 3$. Also, $M = 5$ has a big average running times compared with $M = 3$.

We have also studied the influence of processing time and delivery time in Tables 5.9 and 5.10.

As we can see in Table 5.9, with a medium range of processing times (i.e., 1 to 100), we have a big size of the Pareto front, because we do not lose many solutions. Moreover, the average value of $C^*_{max}$ and $L^*_{max}$ increases with a wide range of processing times, and also the average running times will increase as shown in column 5.

Table 5.9 Results of our DP algorithm vs. processing time ranges (on five machines)

| $p_i$ ranges | Size of Pareto front | $C^*_{max}$ | $L^*_{max}$ | Running Times (s) |
|---|---|---|---|---|
| 1-20 | 24.83 | 20.89 | 195.06 | 4.64 |
| 1-100 | 32.06 | 101.11 | 258.61 | 34.06 |
| 1-500 | 15.28 | 495.00 | 617.61 | 36.89 |

In Table 5.10, we can see that the size of the Pareto front with delivery times: 1 to 20 and 1 to 500, is smaller than the size of the Pareto front with delivery times: 1 to 100 (as in $M = 3$). In addition, the average of $C^*_{max}$ for the jobs having a medium range of delivery times are quite big compared with others ranges. While the average of $L^*_{max}$ grows with increasing the ranges of delivery times. Furthermore, as we can see in column 5, the running times is growing but slower than the delivery times ranges.

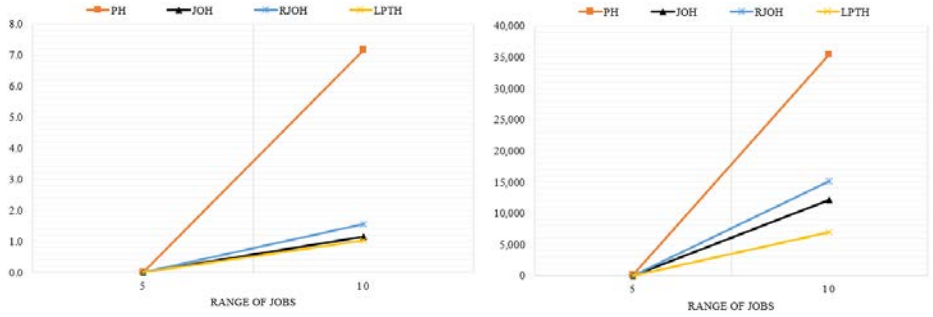
Table 5.10 Results of our DP algorithm vs. delivery time ranges (on five machines)

| $q_i$ ranges | Size of Pareto front | $C^*_{max}$ | $L^*_{max}$ | Running Times (s) |
|---|---|---|---|---|
| 1-20 | 15.61 | 195.00 | 205.06 | 24.56 |
| 1-100 | 39.39 | 215.39 | 278.11 | 24.86 |
| 1-500 | 17.17 | 206.61 | 588.11 | 26.18 |

#### 5.4.2.2 Results of heuristics

We have also studied the influence of heuristics on five machines. The results are given for our two sets of instances, from 5 to 10 jobs. It should be noted that the results of set three (i.e., number of jobs: 15) were not presented because it was too slow, so we stopped after one hour of launch. In addition, the results are consistent with the results of the heuristics on three machines (see Section 5.4.1.2).

Fig. 5.13 presents a comparison of our heuristics. On these two figures (A and B), we can observe that: with a small jobs, all the heuristics have the same average running times, and with the increase in the number of jobs, the LPTH is faster than other proposed heuristics (see Fig. 5.13. (A)). We can also see that the size of the partial solutions set (PSS) grows with the increase in the number of jobs (see Fig. 5.13. (B)), which is consistent with the average running times.

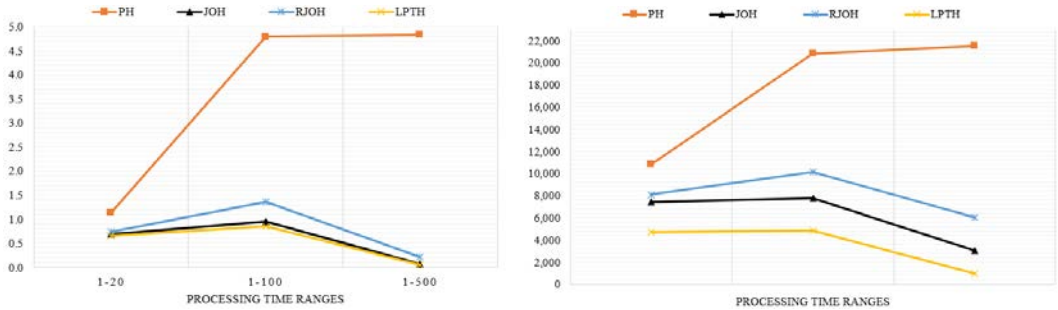



(A) Average running times (s) (B) Size of solutions in the PSS

Fig. 5.13 Results of heuristics vs. size of instances

As a function of processing times, in Fig. 5.14, we can observe that: with a wide range of processing times (i.e., 1 to 500), the average running times of heuristics JOH, RJOH and LPTH are decreasing because we remove many solutions. While PH is not really affected by high processing times, as we have almost the same PSS of medium processing times.

Moreover, the LPTH is always faster than other proposed heuristics (see Fig. 5.14. (A)). We can also see that the size of the partial solutions set (PSS) decreasing with a big range of processing times. Indeed, we remove many solutions (see Fig. 5.14. (B)), which is consistent with the average running times.

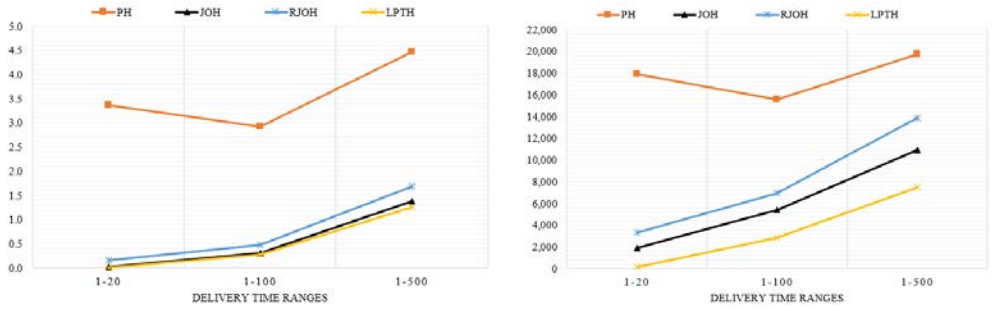



(A) Average running times (s) (B) Size of solutions in the PSS

Fig. 5.14 Results of heuristics vs. processing time ranges

Fig. 5.15 presents the results of our heuristics as a function of delivery times, as we can see in Fig. 5.12. (A), as previously, the LPTH is faster than other proposed heuristics. Moreover, all heuristics become slower with a wide range of delivery times, as we do not remove too many states. We can also see that heuristic PH achieves small running times with a medium range of delivery times compared with other ranges, as it has a small size of

solutions in PSS (see Fig. 5.15. (B)). In addition, the size of the partial solutions set (PSS) increases with the increasing range of delivery times as we see in Fig.5.12. (B), which is consistent with the average running times.




(A) Average running times (s) (B) Size of solutions in the PSS

Fig. 5.15 Results of heuristics vs. delivery time ranges

## 5.5 New Simplifications and PTAS

In this section, we describe our PTAS. It uses a simplification approach, which consists in merging some similar jobs (small and having equivalent tails) and which is adapted for the maximum lateness criterion. This approach is inspired from Kacem and Kellerer [6]. This PTAS is an extension of the one proposed in the previous chapter and it is described by the following algorithm.

**PTAS Algorithm**

1. Let $\varepsilon > 0$ (assume that $\frac{2}{\varepsilon}$ is an integer) and $J = \{1,2,...,n\}$.
2. Adjust every tail $q_j (j \in J)$ as follows: $q_j := \lfloor \lceil \frac{q_j}{\varepsilon q_{max}/2} \rceil \cdot \frac{\varepsilon q_{max}}{2} \rfloor$ and define the $\frac{2}{\varepsilon}$ classes $C_k (1 \leq k \leq 2/\varepsilon)$ such that the tail of every job in $C_k$ is equal to $\lfloor k.\varepsilon q_{max}/2 \rfloor$.
3. For every class $C_k (1 \leq k \leq 2/\varepsilon)$ do the following:
   (a) Divide the set of all available jobs into two subsets as follows:
      i. The small jobs in $S$, where $S = \{j \in J | p_j < \varepsilon^2 P/8.m\}$ .
      ii. The large jobs in $G$, where $G = \{j \in J | p_j \geq \varepsilon^2 P/8.m)\}$ .
   (b) Merge jobs in $S$ until the processing time $p$ of the obtained job will satisfy $\frac{\varepsilon^2 P}{8.m} \leq p < \frac{\varepsilon^2 P}{4.m}$ or it remains a single small job in the class.

4. After merging the small jobs, the new instance $I^{''}$ is optimally solved by using the dynamic programming algorithm.

5. This schedule of $I^{''}$ is used to schedule the jobs of instance $I'$: when a job $j$ of instance $I^{''}$ is scheduled on a machine, then all the corresponding jobs of instance $I'$ are also scheduled on this same machine.

6. Return the Pareto front from the solutions obtained in Step (5), by only keeping non-dominated states.

In the following, the two main steps of the algorithm are described and the corresponding proofs are given.

**STEP 1:**

Let $L^*_{max}(I)$ denote the minimal maximum lateness for instance $I$.
First, we simplify the difficult instance $I$ into a more primitive instance $I^{'}$ which is easier to deal with, depending on the desired precision $\varepsilon$ of approximation. The simplification will be as follows:
Given an arbitrary $\varepsilon > 0$. We assume that $\frac{2}{\varepsilon}$ is an integer (to have an integer number of job classes, and to have the same delivery time value for each class) and we split the interval $[0, q_{max}]$ in $\frac{2}{\varepsilon}$ equal length classes. In class $C_k (1 \leq k \leq 2/\varepsilon)$ we round up every tail $q_j$ to $q_j := \lfloor \lceil \frac{q_j}{\varepsilon q_{max}/2} \rceil \cdot \frac{\varepsilon q_{max}}{2} \rfloor$. The transformed instance is called $I^{'}$.

**Proposition 5.5.0.1.** *The modified instance $I^{'}$ is computed in $O(n)$ time and this can be done with no more than $(1+\varepsilon/2)-$loss.*

*Proof.* The proof is the same as in Chapter 4. □

**STEP 2:**

In this step, $J$ is divided into at most $2/\varepsilon$ subsets $C_k$ $(1 \leq k \leq 2/\varepsilon)$, with a unique tail equal to $\lfloor k.\varepsilon q_{max}/2 \rfloor$ for every job in $C_k$. Then, we reduce the number of small jobs in every subset $C_k$. Indeed, for each class we distinguish the large and small jobs. The small jobs have processing time less than $\frac{\varepsilon^2 P}{8.m}$ and the large jobs have processing time greater or equal to $\frac{\varepsilon^2 P}{8.m}$.

The reduction is obtained by using the merging approach the small jobs in each class $C_k$. In every class, we merge jobs until having a new greater job that has a processing time between $\frac{\varepsilon^2 P}{8.m}$ and $\frac{\varepsilon^2 P}{4.m}$. The small jobs are considered one by one according to the Jackson's order when applying the merging procedure. At the end, it remains at most a single small job for each class $C_k$. At most, in the new instance, $(\frac{8.m}{\varepsilon^2} + \frac{2}{\varepsilon})$ jobs can remain. Clearly, $\frac{2}{\varepsilon}$ for the

small jobs of every subset $C_k$ and $\frac{8.m}{\varepsilon^2}$ for the large jobs.

After merging the small jobs, a new instance $I^{''}$ is obtained. Thus, a matching array is created to identify for each job of instance $I^{''}$ the corresponding jobs in the instance $I'$. Moreover, the instance $I^{''}$ will be solved by using the dynamic programming algorithm $A$.

Finally, the obtained schedule of $I^{''}$ is used to schedule the jobs of instance $I'$. When a job $j^{'}$ of instance $I^{''}$ is scheduled on a machine, then all the corresponding jobs of instance $I'$ are also scheduled on the same machine. Obviously, in instance $I'$ jobs are scheduled on every machine by using Jackson's order.

**Theorem 5.5.0.1.** *For every fixed value of m, the studied problem has a PTAS with a time complexity of* $O(n.m^{(\frac{8.m}{\varepsilon^2}+\frac{2}{\varepsilon})})$.

*Proof.* The proof is based on the analysis of the previous steps. By Proposition 5.5.0.1, we need to prove that the second step will not cost more than $(1+\varepsilon)-$loss. The instance $I^{''}$ obtained after the second step, has a limited number of jobs. All their possible assignments on the $m$-machine can be determined in $O(m^{(\frac{8.m}{\varepsilon^2}+\frac{2}{\varepsilon})})$. Then, we can derive all the associated feasible solutions for $I'$ in $O(n.m^{(\frac{8.m}{\varepsilon^2}+\frac{2}{\varepsilon})})$ and select the best schedule. Let $\Delta_k$ be the length of jobs to be assigned from class $C_k$ on the first machine in an optimal schedule/assignment of instance $I'$. We will show that the PTAS generates a feasible assignment $AS$ with a length of jobs assigned from class $C_k$ on the first machine, (this length is denoted as $\Delta'_k$ ), which is "close" enough to the value $\Delta_k$. In this assignment $AS$, the large jobs, not merged, to be optimally assigned from class $C_k$ on the first machine are assumed to be known (by guessing). We have to approximate the optimal remaining length of the other (small) jobs on the first machine. W.L.O.G., we consider that this remaining length is equal to $\Delta_k$ (since the large jobs, not merged, to be optimally assigned on the first machine can be correctly guessed). Take now the merged jobs (and possibly the only remaining small job after the merging step) from class $C_k$ and add their processing times one by one in any order until we get a total processing time $g$ minimally greater or equal to $\Delta_k$. Obviously, we have $g-\Delta_k \leq \varepsilon^2 P/4.m$. Then, take $\Delta'_k = g$. For the same class of jobs, we apply the similar process to the next machines, except the last one (since the remaining processing time will be less than the optimal value $\Delta_k$ on this last machine). Since there are at most $\frac{2}{\varepsilon}$ classes, the total loss is bounded by $\varepsilon P/2.m$, which is less or equal to $\varepsilon L^*_{max}(I)/2$. This loss is limited for the two criteria. In overall, the solution yielded by the PTAS is $(1+\varepsilon)-$approximation. □

## 5.6 FPTAS

The main principle of the proposed FPTAS is to eliminate a specific subset of the states generated by the dynamic programming algorithm $A$. Therefore, the modified algorithm $A'$ produces an approximation solution instead of the optimal solution (see the detail in the FPTAS Algorithm).
Given an arbitrary $\varepsilon > 0$, we define the following parameters:

$$\delta_1 = \frac{\varepsilon.P}{n.m},$$

and

$$\delta_2 = \frac{\varepsilon.(P+q_{max})}{n.(m+1)}.$$

where $q_{max}$ is the maximum delivery time and $P$ is the total sum of processing times.
Let $L^*_{max}$ and $C^*_{max}$ be the minimal values and let $LMAX$ and $CMAX$ be the upper bounds for the two considered objectives, such that,

$$L^*_{max} \leq LMAX = P + q_{max} \leq (m+1).L^*_{max}$$

$$C^*_{max} \leq CMAX = P \leq m.C^*_{max}$$

We divide the intervals $[0, CMAX]$ and $[0, LMAX]$ into equal sub-intervals respectively of lengths $\delta_1$ and $\delta_2$. Then, an FPTAS is defined by applying the DP, except the fact that it will keep at every iteration $j$ only one representative state for every couple of the defined subintervals produced from $[0, CMAX]$ and $[0, LMAX]$. Thus, our FPTAS will generate approximate sets $\chi^{\#}_j$ of states instead of $\chi_j$.

**FPTAS Algorithm** $A'$

1. Let $\varepsilon > 0$ and $j = \{1, 2, ..., n\}$.
2. Split the interval $[0, CMAX]$ and $[0, LMAX]$ into sub-intervals respectively of sizes $\delta_1$ and $\delta_2$.
3. Set $\chi^{\#}_1 = \{[t_1 = p_1, t_2 = 0, ..., t_m = 0, p_1 + q_1]\}$.
4. For $j \in \{2, 3, ..., n\}$ do
   (a) $\chi^{\#}_j = \emptyset$.
   (b) For each state $[t_1, t_2, ..., t_m, f]$ in $\chi^{\#}_{j-1}$ do

i. Add $[t_1,...,t_k+p_j,t_{(k+1)},...,t_m,max\{f,t_k+p_j+q_j\}]$ to $\chi_j^{\#}$, avoiding symmetric solutions (for every $k=\{1,2,...,m\}$).

(c) For every couple of the defined sub-intervals produced from $[0,CMAX]$ and $[0,LMAX]$: keep only one representative state with the smallest possible $t_1$.

(d) Remove $\chi_{j-1}^{\#}$.

5. Return the Pareto front from the solutions obtained in previous step, by only keeping non-dominated states.

**Theorem 5.6.0.1.** *For a fixed value m, the studied problem has an FPTAS with time complexity of* $O(n.(\frac{n.m}{\varepsilon})^m \cdot \frac{n.(m+1)}{\varepsilon})$.

*Remark.* The proof uses close ideas compared to the one exposed in the Chapter 4.

## 5.7 Results of our PTAS and FPTAS algorithms

The following results have been obtained after testing the performance of the proposed algorithms. As previously (see Section 5.4), the code has been done in Java and the experiments were performed on an Intel(R) Core(TM)-i7 with 8GB RAM. The results have been tested with different values of $\varepsilon$: 0.8, 0.4 and 0.2. To ensure the consistency of running times, each test has been run three times.

As in Chapter 4, we will use the HV ratio (denoted by $HV_r$) to compare the quality of our approximation algorithms. Furthermore, to compare the two objectives of our PTAS and FPTAS algorithms, we will use the average quality of the two objectives: $L_{max}^P/L_{max}^*$ and $C_{max}^P/C_{max}^*$ for our PTAS (respectively, $L_{max}^F/L_{max}^*$ and $C_{max}^F/C_{max}^*$ for our FPTAS). It should be noted, as already mentioned 5.4, we have randomly generated three sets of instances, with various processing and delivery times:

- number of jobs: 5, 10 and 15;
- processing times : from 1 to 20, 1 to 100 and 1 to 500;
- delivery times : from 1 to 20, 1 to 100 and 1 to 500;

For each set of parameters, we have generated three different instances. That gave us 27 instances in each set of instances. The results have been tested on three machines and five machines (i.e., $M$ : 3 and 5). Noteworthy, the results of set three will not be presented with the results of five machines. In fact, our dynamic programming algorithm with set three was too slow, so we stopped after one hour of launch. Also, for $M=3$, we will not present the

results of set three with processing times of 1 to 500, where we stopped after one hour of launch.

The remainder of this section is organized as follows. The results of the proposed PTAS are presented in Subsection 5.7.1. Finally, Subsection 5.7.2 presents the results of the proposed FPTAS.

## 5.7.1 Results of the PTAS algorithm vs. DP algorithm

In this section, we will present the results of the PTAS algorithm on three machines ($M = 3$) in Subsection 5.7.1.1 and the results of five machines ($M = 5$) will be presented in Subsection 5.7.1.2.

### 5.7.1.1 Results of the PTAS on three machines

#### 5.7.1.1.1 *Quality of the PTAS algorithm*

In this subsection, we will see the quality of the PTAS algorithm using Hypervolume ratios $HV_r$. We will also present the results for the average values of $L_{max}$ and $C_{max}$. The results are given for our two sets of instances (from 5 to 10 jobs) on three machines.

As we can see in the tables 5.12, 5.13 and 5.14, the size of Pareto front obtained by our algorithms increases as the number of jobs increases. Indeed, the size of the Pareto front for our PTAS is very small, when it is between 2.19 and 5.41 for DP, it is always less than 2.19 for our PTAS. Moreover, the HV ratio is very bad and strongly decreases with the size of instances (we have a good quality of solutions with small instances). This can be explained as follows: we can see in Table 5.11, whatever the size of instances, we approximately have the same number of jobs in the simplified instance $I'$. Indeed, as the number of jobs is small (from 5 to 10), it is difficult to find small jobs to merge.

Table 5.11 Average number of jobs in the simplified instance $I'$ with $\varepsilon$ = 0.8, 0.4, 0.2 and 0.1

| #jobs | Average number of jobs in $I'$ | | | I |
|---|---|---|---|---|
| | $\varepsilon = 0.8$ | $\varepsilon = 0.4$ | $\varepsilon = 0.2$ | |
| 5 | 5 | 5 | 5 | 5 |
| 10 | 10 | 10 | 10 | 10 |

Tables 5.12, 5.13, and 5.13 present the results of our PTAS as a function of the number of jobs, for three different values of $\varepsilon$ (0.8, 0.4 and 0.2). Column 1 shows our two sets of

instances. In the columns 2 and 3, we present the size of Pareto front for the two algorithms (DP and PTAS). Column 4 presents the average number of jobs in the simplified instance $I'$. In the column 5 and 6, we will provide the average of our PTAS results for the two objectives of our study: $L^P_{max}/L^*_{max}$ and $C^P_{max}/C^*_{max}$. Finally, in column 7 we present the results of the HV ratios.

We can see that the PTAS algorithm finds solutions closer to the optimal ones with small $\varepsilon$ value. Indeed, there is no big influence for $\varepsilon$ because we use small jobs (from 5 to 10).

It is worth-mentioning that the difference between the average approximate makespan and the average optimal value of this criterion is decreasing when the number of jobs increases. The other objective ($L_{max}$) has the opposite behavior. $L_{max}$ values are closer to the optimum than $C_{max}$ values, which is not surprising, as we use small jobs.

Table 5.12 Quality of the PTAS algorithm vs. number of jobs with $\varepsilon$= 0.8

| #jobs | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 2.19 | 1.15 | 5 | 1.32750E+00 | 1.13679E+00 | 89.91% |
| 10 | 5.41 | 1.63 | 10 | 1.27610E+00 | 1.20777E+00 | 52.50% |

Table 5.13 Quality of the PTAS algorithm vs. number of jobs with $\varepsilon$= 0.4

| #jobs | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 2.19 | 1.15 | 5 | 1.32807E+00 | 1.13690E+00 | 89.89% |
| 10 | 5.41 | 1.78 | 10 | 1.27157E+00 | 1.19547E+00 | 52.95% |

Table 5.14 Quality of the PTAS algorithm vs. number of jobs with $\varepsilon$= 0.2

| #jobs | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 2.19 | 1.15 | 5 | 1.32769E+00 | 1.13723E+00 | 89.90% |
| 10 | 5.41 | 1.74 | 10 | 1.27543E+00 | 1.19732E+00 | 52.66% |

In the rest of this section, we will present the results of $\varepsilon = 0.2$, the analysis of the results are the same with $\varepsilon$ = 0.8 and 0.4.

We have also studied the influence of processing time in Table 5.15. The results are also given for our two sets of instances, from 5 to 10 jobs.

The results showed that the instances having a wide range of job processing times are more difficult to solve to optimality. Moreover, with a wide range of processing times, as we use small jobs, it is difficult to find small jobs to merge, so we do not lose solution (see column 4 in Table 5.15). Nevertheless, as we can see in the columns 5 and 6, the average of $C_{max}$ is decreasing with a wide range of processing times but, $L_{max}$ is growing. In addition, the size of the Pareto front found by the PTAS is increasing with a wide range of processing times.

Table 5.15 Quality of the PTAS algorithm vs. processing time ranges with $\varepsilon$= 0.2

| $p_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 3.28 | 1.28 | 7.5 | 1.35743E+00 | 1.01215E+00 | 97.84% |
| 1-100 | 4.33 | 1.44 | 7.5 | 1.32160E+00 | 1.09466E+00 | 63.24% |
| 1-500 | 3.78 | 1.61 | 7.5 | 1.28465E+00 | 1.24300E+00 | 52.76% |

* The size of the original instance $I = 7.5$.

Moreover, we have studied the influence of delivery time in Table 5.16. The results are given for our three sets of instances, from 5 to 15 jobs (without processing times from 1 to 500, for set three). The results showed that the delivery times have no real influence on the quality obtained by the PTAS. Furthermore, as we can see in the columns 5 and 6, the average of $C_{max}$ is growing when the jobs have a wide range of delivery times, but, $L_{max}$ is not really affected by delivery times. In addition, the delivery times have no real influence on the size of the Pareto front by our algorithms.

Table 5.16 Quality of the PTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.2

| $q_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 4.42 | 1.42 | 9.375 | 1.18593E+00 | 1.18495E+00 | 82.20% |
| 1-100 | 4.67 | 1.67 | 9.375 | 1.30459E+00 | 1.23696E+00 | 74.10% |
| 1-500 | 4.04 | 1.38 | 9.375 | 1.31835E+00 | 1.10034E+00 | 76.14% |

* The size of the original instance $I = 9.375$.

#### 5.7.1.1.2 *Average running times of the PTAS vs. DP*

In this subsection, the average running times are given in the figures 5.16 and 5.17. They compare the DP and PTAS algorithms, considering different values of $\varepsilon$ = 0.8, 0.4 and 0.2. All values are in milliseconds (ms).

Fig. 5.16 shows that all algorithms are slower when the number of jobs is growing. Moreover, the running time in PTAS decreases when the value of $\varepsilon$ grows, which is consistent with the theory.

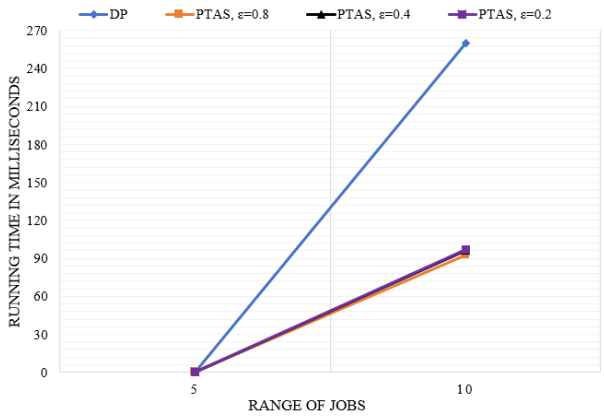
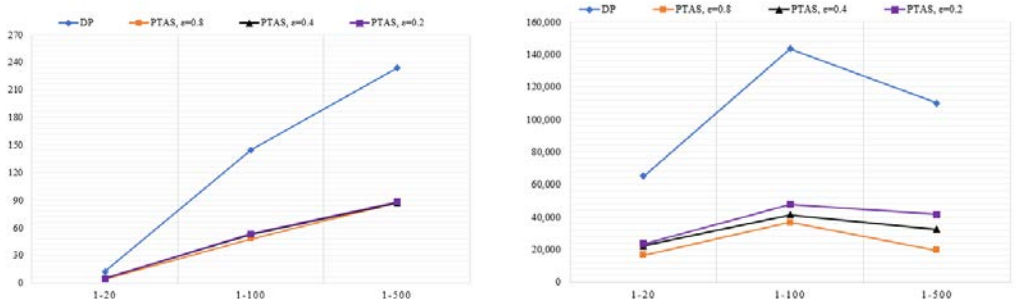



Fig. 5.16 PTAS: Average running times (ms) vs. size of instances

We have also studied the average running times depending on processing and delivery times, in Fig.5.16. The left part of this figure shows the average running times (ms) vs. processing times found by the DP algorithm and the PTAS. We can see that all algorithms are slower when the processing time is growing. It should be noted that increased processing times increases the value of running times. As they are more solutions, our algorithms have a lot of states to consider at each step.

On the right part of the same figure are given the average running times (ms) vs. delivery times. As we can see, the DP algorithm with a medium range of delivery times is very slow. Indeed, with this range (i.e, from 1 to 500), we do not lose many solutions (see Table 5.16 column 3). In addition, the delivery times have no real influence on the running times obtained by the PTAS algorithm: running times are growing, but slower than the delivery times ranges. Furthermore, the running time in PTAS decreases when the value of $\varepsilon$ grows, which is consistent with theory.




Running times vs. processing times Running times vs. delivery times

Fig. 5.17 Average running times (ms) vs. processing and delivery times ranges

### 5.7.1.2 Results of the PTAS on five machines

In this section, we will present the results of the PTAS algorithm. All the results are given for our two sets of instances (from 5 to 10 jobs) on five machines.

#### 5.7.1.2.1 *Quality of the PTAS algorithm*

In this subsection, we will see the quality of the PTAS algorithm using Hypervolume ratios $HV_r$. We will also present the results for the average values of $L_{max}$ and $C_{max}$.

As we can see in the tables 5.17, 5.18 and 5.19, the size of Pareto front obtained by the DP algorithm increases as the number of jobs increases, the PTAS has an opposite behaviour. Moreover, the HV ratio is very bad and strongly decreases with the size of instances (we have a good quality of solutions with small instances). Hence, the results are consistent with the PTAS on three machines (see Section 5.7.1.1). In addition, $C_{max}$ and $L_{max}$ is growing with increasing number of jobs.

Table 5.17 Quality of the PTAS algorithm vs. number of jobs with $\varepsilon$= 0.8

| #jobs | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 4.89 | 2.07 | 5 | 1.25235E+00 | 1.15227E+00 | 89.78% |
| 10 | 43.22 | 2.11 | 10 | 1.29839E+00 | 1.15443E+00 | 62.28% |

Table 5.18 Quality of the PTAS algorithm vs. number of jobs with $\varepsilon$= 0.4

| #jobs | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 4.89 | 2.07 | 5 | 1.25235E+00 | 1.15227E+00 | 89.78% |
| 10 | 43.22 | 2.04 | 10 | 1.29839E+00 | 1.15443E+00 | 62.28% |

Table 5.19 Quality of the PTAS algorithm vs. number of jobs with $\varepsilon$= 0.2

| #jobs | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 4.89 | 2.07 | 5 | 1.25235E+00 | 1.15227E+00 | 89.78% |
| 10 | 43.22 | 2.04 | 10 | 1.29839E+00 | 1.15443E+00 | 62.28% |

In the rest of this section, we will present the results of $\varepsilon = 0.2$, the analysis of the results is the same with $\varepsilon$ = 0.8 and 0.4.

We have also studied the influence of processing time and delivery time in Tables 5.20 and 5.21.

As in PTAS on three machines 5.7.1, the instances having a wide range of job processing times are more difficult to solve to optimality. Moreover, as we can see in the columns 5 and 6, the processing times have no real influence on $C_{max}$, but, $L_{max}$ is growing with a wide range of processing times. In addition, the processing times have no real influence on the size of the Pareto front found by our algorithms.

In Table 5.21, we can see that delivery times have no real influence on the quality obtained by the PTAS algorithm (as in PTAS on three machines). Furthermore, as we can see in the columns 5 and 6, the average of $C_{max}$ is growing when the jobs have a wide range of delivery times, and $L_{max}$ is decreasing. In addition, the delivery times have no real influence on the size of the Pareto front by our algorithms.

Table 5.20 Quality of the PTAS algorithm vs. processing time ranges with $\varepsilon$= 0.2

| $p_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 24.83 | 1.94 | 7.5 | 1.17287E+00 | 1.00997E+00 | 98.49% |
| 1-100 | 32.06 | 1.83 | 7.5 | 1.28626E+00 | 1.10183E+00 | 66.79% |
| 1-500 | 15.28 | 2.39 | 7.5 | 1.28418E+00 | 1.22047E+00 | 62.83% |

* The size of the original instance $I = 7.5$.

Table 5.21 Quality of the PTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.2

| $q_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 15.61 | 2.00 | 7.5 | 1.23191E+00 | 1.28177E+00 | 79.26% |
| 1-100 | 39.39 | 2.00 | 7.5 | 1.29920E+00 | 1.26568E+00 | 73.76% |
| 1-500 | 17.17 | 2.17 | 7.5 | 1.30761E+00 | 1.05573E+00 | 75.08% |

* The size of the original instance $I = 7.5$.

#### 5.7.1.2.2 *Average running times of the PTAS vs. DP*

In this subsection, the average running times are given in the figures 5.18 and 5.19. They compare the DP and PTAS algorithms, considering different values of $\varepsilon$ = 0.8, 0.4 and 0.2. It should be noted that since the running times are quite big, they have been expressed in seconds (s).

It is worth mentioning all the results are consistent with PTAS on three machines 5.7.1. Clearly, Fig. 5.18 shows that all algorithms are slower when the number of jobs is growing.

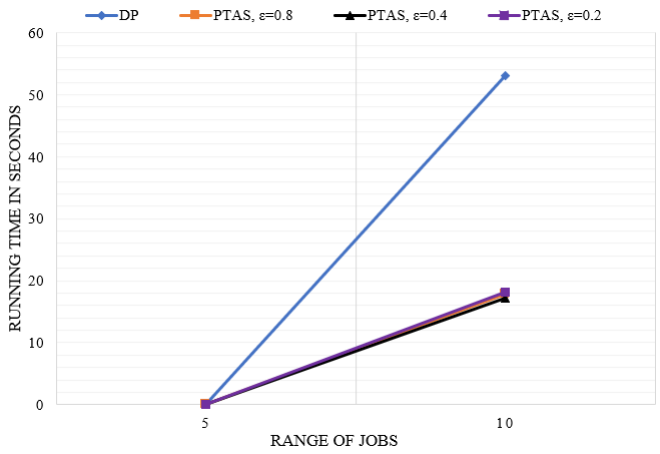
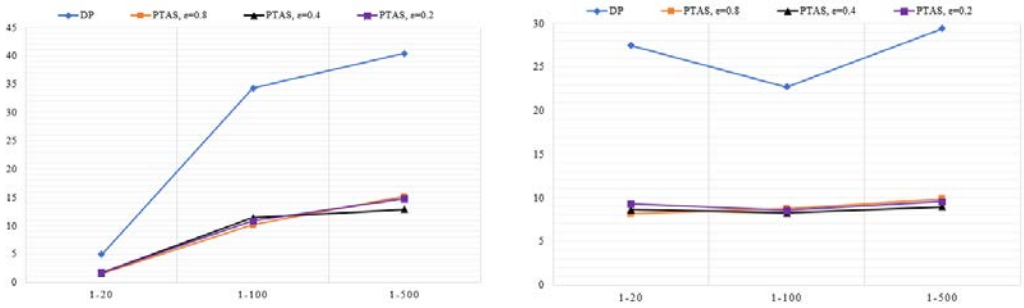



Fig. 5.18 PTAS: Average running times (s) vs. size of instances

Moreover, in Fig.5.18, the left part of this figure shows the average running times (s) vs. processing times found by the DP algorithm and the PTAS. We can see that all algorithms are slower when the processing time is growing.

On the right part of the same figure are given the average running times (s) vs. delivery times. As we see, the delivery times have no real influence on the running times obtained by the PTAS algorithm.




Running times vs. processing times Running times vs. delivery times

Fig. 5.19 Average running times (s) vs. processing and delivery times ranges

## 5.7.2 Results of the FPTAS algorithm vs. DP algorithm

In this section, we will present the results of the FPTAS algorithm on three machines ($M = 3$) in Subsection 5.7.2.1 and the results of five machines ($M = 5$) will be presented in Subsection 5.7.2.2.

#### 5.7.2.1 Results of the FPTAS on three machines

##### 5.7.2.1.1 *Quality of the FPTAS algorithm*

In this subsection, we will see the quality of the FPTAS algorithm using Hypervolume ratios $HV_r$. We will also present the results for the average values of $L_{max}$ and $C_{max}$. As earlier, the results are given for our two sets of instances (from 5 to 10 jobs) on three machines.

Tables 5.22, 5.23, and 5.23 present the results of the FPTAS as a function of the number of jobs, for three different values of $\varepsilon$ (0.8, 0.4 and 0.2).

We can see that with good $\varepsilon$ (small $\varepsilon$), we have a good solution, which is consistent with the theory. Moreover, in columns 2 and 3, we can see the size of the Pareto front obtained by our algorithms increases as the number of jobs increases (as in PTAS). It is worth-mentioning that the FPTAS can find solutions closer to the optimal ones when the number of jobs decreases (as in PTAS), while $C_{max}$ and $L_{max}$ increases.

Table 5.22 Quality of the FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.8

| #jobs | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 2.19 | 1.33 | 1.00533E+00 | 1.00278E+00 | 99.82% |
| 10 | 5.41 | 1.39 | 1.03066E+00 | 1.03349E+00 | 93.56% |

Table 5.23 Quality of the FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.4

| #jobs | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 2.19 | 1.37 | 1.00437E+00 | 1.00311E+00 | 99.83% |
| 10 | 5.41 | 1.96 | 1.02622E+00 | 1.02860E+00 | 94.64% |

Table 5.24 Quality of the FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.2

| #jobs | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 2.19 | 1.37 | 1.00437E+00 | 1.00311E+00 | 99.83% |
| 10 | 5.41 | 1.85 | 1.02015E+00 | 1.02371E+00 | 96.34% |

In the rest of this section, we will present the results of $\varepsilon = 0.2$, the analysis of the results are the same with $\varepsilon$ = 0.8 and 0.4.

We have also studied the influence of processing time in Table 5.25. As earlier, the results are given for our two sets of instances, from 5 to 10 jobs.

As in PTAS, the results showed that the instances having a wide range of job processing times are more difficult to solve to optimality. Moreover, $C_{max}$ is not really affected by processing times, while $L_{max}$ is growing. In addition, the size of the Pareto front found by the FPTAS algorithm is increasing with a wide range of processing times (as in PTAS).

Table 5.25 Quality of the FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.2

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 3.28 | 1.28 | 1.00803E+00 | 1.00028E+00 | 99.94% |
| 1-100 | 4.33 | 1.72 | 1.00665E+00 | 1.00178E+00 | 98.91% |
| 1-500 | 3.78 | 1.83 | 1.01678E+00 | 1.02434E+00 | 95.41% |

In addition, we have studied the influence of delivery time in Table 5.26. The results are given for our three sets of instances, from 5 to 15 jobs (without processing times from 1 to 500, for set three). The results showed that the delivery times have no real influence on the quality obtained by the FPTAS (as in PTAS). Furthermore, as we can see in the columns 5 and 6, $C_{max}$ is not really affected by delivery times, while $L_{max}$ is decreasing with a wide range of delivery times. Moreover, as in the PTAS, the delivery times have no real influence on the size of the Pareto front by our algorithms.

Table 5.26 Quality of the FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.2

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 4.42 | 1.92 | 1.01709E+00 | 1.02024E+00 | 98.45% |
| 1-100 | 4.67 | 3.00 | 1.01109E+00 | 1.01527E+00 | 98.42% |
| 1-500 | 4.04 | 2.04 | 1.01342E+00 | 1.01815E+00 | 98.69% |

#### 5.7.2.1.2 *Average running times of the FPTAS vs. DP*

In this subsection, the average running times are given in the figures 5.20 and 5.21. They compare the DP and FPTAS, considering different values of $\varepsilon$ = 0.8, 0.4 and 0.2. All values are in milliseconds (ms).

As a function of the number of jobs, as shown in Fig. 5.20, the FPTAS is slower when the number of states is growing. Hence, the results are consistent with the PTAS.

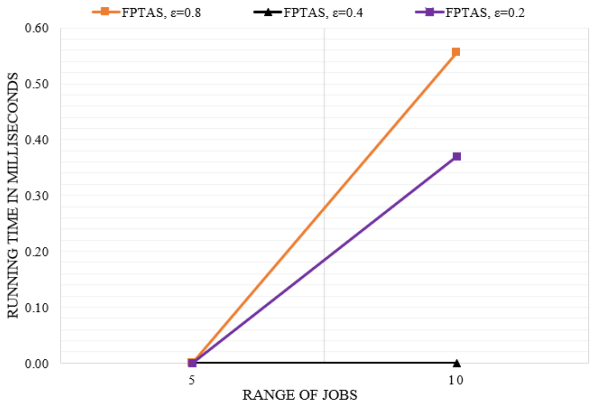
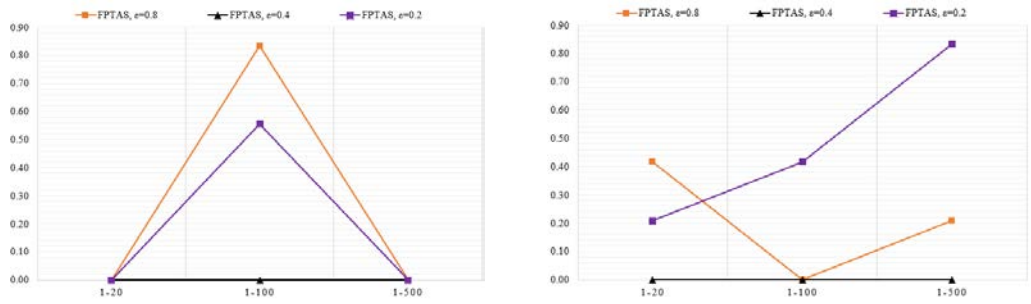



Fig. 5.20 FPTAS: Average running times (ms) vs. size of instances

*Remark.* Since we use small jobs, the FPTAS give results less than a second, so there is no real influence for $\varepsilon$.

We have also studied the average running times depending on processing and delivery times, in Fig. 5.20. The left part of this figure shows the average running times (ms) vs. processing times found by the FPTAS. We can see that the FPTAS algorithm is slower when the jobs have a medium range of processing time.

On the right part of the same figure are given the average running times (ms) vs. delivery times. As we can see, the FPTAS is slower when the delivery times are growing. For $\varepsilon = 0.8$, with small delivery times, the running times are growing, but as we can see the running times slower than one second.




Running times vs. processing times Running times vs. delivery times

Fig. 5.21 Average running times (ms) vs. processing and delivery times ranges

*Remark.* As DP goes to 110,003 milliseconds, we did not include it here, because the FPTAS running time is too small (the average running times of DP can be seen in Figures. 5.16 and 5.17).

#### 5.7.2.2 Results of the FPTAS on five machines

In this section, we will present the results of the FPTAS algorithm. As previously 5.7.1, all the results are given for our two sets of instances (from 5 to 10 jobs) on five machines.

##### 5.7.2.2.1 *Quality of the FPTAS algorithm*

In this subsection, we will see the quality of the FPTAS using Hypervolume ratios $HV_r$. We will also present the results for the average values of $L_{max}$ and $C_{max}$.

It should be noted that the results are consistent with the previous results (see Section 5.7.2.1). Clearly, as we can see in the tables 5.27, 5.28 and 5.29, the size of Pareto front obtained by our algorithms increases as the number of jobs increases. Furthermore, with good $\varepsilon$ (small $\varepsilon$), we have a good solution, which is consistent with the theory. It is worth-mentioning that the FPTAS can find solutions closer to the optimal ones when the number of jobs decreases. In addition, $C_{max}$ and $L_{max}$ is growing with increasing number of jobs. Hence, the results are also consistent with the PTAS with five machines (see Section 5.7.1.2).

Table 5.27 Quality of the FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.8

| #jobs | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 4.89 | 1.26 | 1.00000E+00 | 1.00181E+00 | 100.00% |
| 10 | 43.22 | 1.78 | 1.03299E+00 | 1.01734E+00 | 95.43% |

Table 5.28 Quality of the FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.4

| #jobs | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 4.89 | 1.26 | 1.00000E+00 | 1.00181E+00 | 100.00% |
| 10 | 43.22 | 1.59 | 1.02336E+00 | 1.00890E+00 | 96.33% |

Table 5.29 Quality of the FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.2

| #jobs | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5 | 4.89 | 1.26 | 1.00000E+00 | 1.00181E+00 | 100.00% |
| 10 | 43.22 | 1.67 | 1.02190E+00 | 1.00980E+00 | 96.42% |

In the rest of this section, we will present the results of $\varepsilon = 0.2$, the analysis of the results are the same with $\varepsilon$ = 0.8 and 0.4.

We have also studied the influence of processing time and delivery time in Tables 5.20 and 5.21.

As in FPTAS on three machines (Section 5.7.2.1), the instances having a wide range of job processing times are more difficult to solve to optimality (as in PTAS Section 5.7.1.2). $L_{max}$ is not really affected by processing times, while $C_{max}$ decreases with a wide range of processing times. The size of the Pareto front found by the FPTAS algorithm, is also decreasing with a wide range of processing times.

The results of delivery times are also consistent with the FPTAS on three machines. Clearly, in Table 5.31, we can see that delivery times have no real influence on the quality

obtained by the FPTAS (as in PTAS Section 5.7.1.2). Furthermore, as we can see in the columns 5 and 6, the average of $C_{max}$ is not really affected by delivery times, while $L_{max}$ is decreasing with a wide range of delivery times. Moreover, the size of the Pareto front by the FPTAS, is also decreasing with a wide range of processing times.

Table 5.30 Quality of the FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.2

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 24.83 | 1.67 | 1.02394E+00 | 1.00142E+00 | 99.80% |
| 1-100 | 32.06 | 1.50 | 1.01648E+00 | 1.00773E+00 | 97.00% |
| 1-500 | 15.28 | 1.22 | 1.01246E+00 | 1.00738E+00 | 97.82% |

Table 5.31 Quality of the FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.2

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 15.61 | 1.61 | 1.02023E+00 | 1.02167E+00 | 97.62% |
| 1-100 | 39.39 | 1.50 | 1.00619E+00 | 1.01019E+00 | 98.41% |
| 1-500 | 17.17 | 1.28 | 1.01479E+00 | 9.99244E-01 | 98.60% |

#### 5.7.2.2.2 *Average running times of the FPTAS vs. DP*

It is worth-mentioning that FPTAS algorithm with these small jobs (i.e., from 5 to 10 jobs) is very fast. Indeed, as DP goes to 54 seconds (see figures 5.18 and 5.19), the FPTAS running times are always less than 0.5 milliseconds.

## 5.8 Conclusions and Perspectives

The $m$-parallel machines scheduling problem has been considered to minimize the maximum lateness and the makespan. We have proposed an exact algorithm (based on DP) to generate the complete Pareto Frontier in a pseudo-polynomial time for a fixed value of $m$. Moreover, four heuristics (PH, JOH, RJOH and LPTH) have been proposed in order to optimize this

DP algorithm. In addition, we present a PTAS and an FPTAS to solve the problem. For the proposed algorithms, we randomly generated several instances with different ranges, and, for each job $j$, its processing time $p_j$ and delivery time $q_j$ are set to be integer numbers. The results of the experiments showed that the proposed algorithms for the considered problem are very satisfactory.

In our future works, we look to study the problems of scheduling jobs on multiple machines to achieve other objectives.

# Bibliography


[1] Y.K. Lin, J.W. Fowler, and M.E. Pfund (2013). Multiple-objective heuristics for scheduling unrelated parallel machines. *European Journal of Operational Research*, 227(2):239-253.

[2] Y. Huo and H. Zhao (2013). Bi-criteria Scheduling on Multiple Machines Subject to Machine Availability Constraints. *In Frontiers in Algorithmics and Algorithmic Aspects in Information and Management, Springer, Berlin, Heidelberg*, 325-338.

[3] K. Lee, J.Y.T. Leung, and M.L. Pinedo (2013). Makespan minimization in online scheduling with machine eligibility. *Annals of Operations Research*, 204(1):189-222.

[4] X. Li, F. Yalaoui, L. Amodeo, and H. Chehade (2012). Metaheuristics and exact methods to solve a multiobjective parallel machines scheduling problem. *Journal of Intelligent Manufacturing*, 23(4):1179-1194.

[5] Y.H. Chung and L.I. Tong (2011). Makespan minimization for m-machine permutation flowshop scheduling problem with learning considerations. *The International Journal of Advanced Manufacturing Technology*, 56(1-4):355-367.

[6] L.H. Su (2009). Scheduling on identical parallel machines to minimize total completion time with deadline and machine eligibility constraints. *The International Journal of Advanced Manufacturing Technology*, 40(5-6):572-581.

[7] Y. He and Z. Tan (2002). Ordinal on-line scheduling for maximizing the minimum machine completion time. *Journal of Combinatorial Optimization*, 6(2):199-206.

[8] E. Angel, E. Bampis, and A. Kononov (2001). A FPTAS for approximating the unrelated parallel machines scheduling problem with costs. *In European Symposium on Algorithms, Springer, Berlin, Heidelberg*,194-205.

[9] N. Alon, Y. Azar, G.J. Woeginger, and T. Yadid (1998). Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55-66.

# Conclusions and Perspectives

The scheduling problems have been well-studied for more than sixty years. This observation shows the importance of these problems in many domains and applications. Most of these challenging problems belong to the famous class of the NP-hard problems, especially when the environment is constrained of inducing many goals/objectives to handle. In this context, several scheduling problems have been recently addressed in multi-objective and/or constrained contexts. Nonetheless, there are relatively few internal references works for this type of problem, contrary to its many applications. Therefore, the investigation of these problems is particularly motivating. In this thesis, we consider the scheduling problems on parallel machines, with and without non-availability constraints, and the design of approximation methods dedicated to solving this problem in a multi-objective context. For solving these problems, we developed and tested many optimization methods. These methods include different approaches such as heuristics of guaranteed performance, dynamic programming algorithms, PTAS, and FPTAS. In particular, in this thesis, we presented the main generalities about scheduling problems and the approximation algorithms that are the focus of our study. Furthermore, we covered all the complementary aspects related to the multi-objective combinatorial optimization.

In the constrained context, we studied the two-parallel machine scheduling problem with a non-availability interval, with the aim of minimizing the maximum lateness when every job has a positive tail. We proposed an approximation heuristic which has the performance ratio at the worst case not more than 3/2. Moreover, we presented a dynamic programming algorithm and we showed that the problem has an FPTAS, with a strongly polynomial running time, which has been shown to be very efficient. The presented results showed that the proposed algorithms for the considered problem are very efficient, especially for big instances composed of a lot of jobs.

In the context of the multi-objective scheduling problem, we have two contributions, with the aim of minimizing the maximum lateness and the makespan. In the first contribution, we studied the scheduling problem on the two-parallel machine. We have proposed an exact algorithm (based on a dynamic algorithm) to generate the complete Pareto Frontier in a

pseudo-polynomial time. Then, we presented a PTAS to generate an approximate Pareto Frontier. In this scheme, we used a simplification technique based on the merging of jobs. Furthermore, we introduced two FPTAS to generate an approximate Pareto Frontier, the first one is based on the conversion of the dynamic programming, the second one is applied to the simplified instances given by the PTAS. The proposed FPTAS algorithms are strongly polynomial. For the proposed algorithms, we randomly generated several instances. The results of the experiments showed that the proposed algorithms for the considered problem are very efficient, especially for big instances composed of a lot of jobs. In the second contribution, we extended the study of the multi-objective problem to the problem of scheduling jobs on $m$ parallel machines. We have proposed an exact algorithm (based on DP) to generate the complete Pareto Frontier in a pseudo-polynomial time for a fixed value of $m$. Moreover, four heuristics (PH, JOH, RJOH and LPTH) have been proposed in order to optimize this DP algorithm. In these improvements, the idea is to quickly find a bound that will be used not to consider solutions that are dominated by this bound. In addition, different extensions and algorithms have been proposed: a PTAS and an FPTAS for fixed value of $m$. Experimental tests were conducted and allowed us to evaluate and compare the performance of these methods. The results showed that the proposed algorithms for the considered problem are satisfactory. We can conclude that, in this thesis, we developed and tested many optimization methods to tackle the scheduling problem on parallel machines. According to our best knowledge, the problems we considered in this thesis has not been studied before in the literature. For this reason, the elaboration of efficient algorithms for these multiobjective and constrained problems is a new attempt (since it allows us to guarantee the performance of the solutions in reasonable computation time).

**As perspectives of this work, many related questions seems to be interesting to investigate:**

We look forward to extending our results on Chapter 3 to other variants of this problem. We will consider for instance other practical constraints such as release dates. It is clear that adding the release dates will lead to a strongly NP-hard problem. Nevertheless, the problem will keep its approximability property (constant approximation). One of the important questions is whether a PTAS exists or not for such a problem.

The other perspective for the same problem, we will investigate, is related to some multi-objective extensions in which we will try to optimize not only the maximum lateness, but also other criteria, e.g. the maximum completion time and/or the total completion time.

In our third perspective, we look to study the problems of scheduling jobs on multiple machines to achieve other objectives (including the total completion time). One important question is the existence or not of PTAS for solving the problem with an arbitrary number

of machines (our result in this thesis indicates the existence of PTAS for a fixed number of machines, but the answer to the case of arbitrary number is open). This question seems to be particularly challenging.
Finally, it will be also interesting, to extend the studied algorithms to other types of scheduling problems. For example, the problem of minimizing these criteria on parallel machines is surely a nice challenge

# Bibliography


[1] A. A. Lazarev, D. I. A. and Werner, F. (2017). Scheduling jobs with equal processing times on a single machine: minimizing maximum lateness and makespan. *Optimization Letters*, 11(1):165–177.

[2] A. Agnetis, P. D. and Martineau, P. (2017). Scheduling nonpreemptive jobs on parallel machines subject to exponential unrecoverable interruptions. *Computers & Operations Research*, 79:109–118.

[3] A. Liefooghe, L. J. and Talbi, E. (2009). A unified model for evolutionary multi-objective optimization and its implementation in a general purpose software framework. *IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM)*, pages 88–95.

[4] A. Lotov, V. A. and Kamenev, G. (2013). *Interactive decision maps: Approximation and visualization of Pareto frontier*, volume 89. Springer Science & Business Media.

[5] Alharkan, I. M. (2005). *Algorithms for sequencing and scheduling*. Industrial Engineering Department, King Saud University, Saudi Arabia.

[6] Allahverdi, A. and Aldowaisan, T. (2004). No-wait flowshops with bicriteria of makespan and maximum lateness. *European Journal of Operational Research*, 152(1):132–147.

[7] Brucker, P. (2007). *Scheduling Algorithms*. Springer, Berlin.

[8] C. Bazgan, F. J. and Vanderpooten, D. (2015). Approximate pareto sets of minimal size for multi-objective optimization problems. *Operations Research Letters*, 43(1):1–6.

[9] C. He, H. L. and Lin, Y. (2015). Bounded serial-batching scheduling for minimizing maximum lateness and makespan. *Discrete Optimization*, 16:70–75.

[10] C. He, Y. L. and Yuan, J. (2007). Bicriteria scheduling on a batching machine to minimize maximum lateness and makespan. *Theoretical Computer Science*, 381(1-3):234–240.

[11] C. He, Y. L. and Yuan, J. (2009). A dp algorithm for minimizing makespan and total completion time on a series-batching machine. *Information Processing Letters*, 109(12):603–607.

[12] C. He, H. Lin, Y. L. and Tian, J. (2013). Bicriteria scheduling on a series-batching machine to minimize maximum cost and makespan. *Central European Journal of Operations Research*, pages 1–10.

[13] C. He, X.M. Wang, Y. L. and Mu, Y. (2013). An improved algorithm for a bicriteria batching scheduling problem. *RAIRO-Operations Research*, 47(1):1–8.

[14] Chakhar, S. and Martel, J. (2004). Towards a spatial decision support system: Multicriteria evaluation functions inside geographical information systems. 2:97.

[15] Cheng, R. and Gen, M. (1996). Parallel machine scheduling problems using memetic algorithms. *IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No. 96CH35929)*, 4:2665–2670.

[16] Cui, W. and Lu, Z. (2017). Minimizing the makespan on a single machine with flexible maintenances and jobs' release dates. *Computers & Operations Research*, 80:11–22.

[17] D. Oron, D. S. and Steiner, G. (2017). Approximation algorithms for the workload partition problem and applications to scheduling with variable processing times. *European Journal of Operational Research*, 256(2):384–391.

[18] Diakonikolas, I. and Yannakakis, M. (2011). Approximation of multiobjective optimization problems. *Columbia University, New York, NY*.

[19] Dorit, S. and David, B. (1988). A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM journal on computing*, 17(3):539–551.

[20] E. Hebrard, M. Huguet, N. J. A. M. C. P. and Verfaillie, G. (2016). Approximation of the parallel machine scheduling problem with additional unit resources. *Discrete Applied Mathematics*, 215:126–135.

[21] Escoffier, B. and Paschos, V. (2010). A survey on the structure of approximation classes. *Computer Science Review*, 4(1):19–40.

[22] Florios, K. and Mavrotas, G. (2014). Generation of the exact pareto set in multiobjective traveling salesman and set covering problems. *Applied Mathematics and Computation*, 237:1–19.

[23] G. Sapienza, G. Brestovac, R. G. and Seceleanu, T. (2016). On applying multiple criteria decision analysis in embedded systems design. *Design automation for embedded systems*, 20(3):211–238.

[24] Geng, Z. and Yuan, J. (2015). Pareto optimization scheduling of family jobs on a p-batch machine to minimize makespan and maximum lateness. *Theoretical Computer Science*, 570:22–29.

[25] Greenwood, G. W. (2001). Finding solutions to np problems. *In Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, 2:815–822.

[26] Györgyi, P. and Kis, T. (2017). Approximation schemes for parallel machine scheduling with non-renewable resources. *European Journal of Operational Research*, 258(1):113–123.

[27] Halp, L. and Shmoyst, D. (1989). Approximation schemes for constrained scheduling problems. *30th Annual Symposium on Foundations of Computer Science*, pages 134–139.

[28] I. Kacem, A. N. and Seifaddini, M. (2014). Maximum lateness minimization with positive tails on a single machine with an unexpected non-availability interval. *World Congress on Computer Applications and Information Systems (WCCAIS)*, pages 1–5.

[29] I. Kacem, H. K. and Seifaddini, M. (2016). Efficient approximation schemes for the maximum lateness minimization on a single machine with a fixed operator or machine non-availability interval. *Journal of Combinatorial Optimization*, 32(3):970–981.

[30] I. Kacem, M. S. and Schmidt, G. (2016). Strongly fully polynomial time approximation scheme for the weighted completion time minimization problem on two-parallel capacitated machines. *IFAC-PapersOnLine*, 49(12):425–430.

[31] I. Kacem, Y. L. and Sahnoune, M. (2011). Strongly fully polynomial time approximation scheme for the two-parallel capacitated machines scheduling problem. *International Journal of Planning and Scheduling*, 1(1-2):32–41.

[32] J. Wang, G. Fan, Y. Z. C. Z. and Leung, J. Y. (2017). Two-agent scheduling on a single parallel-batching machine with equal processing time and non-identical job sizes. *European Journal of Operational Research*, 258(2):478–490.

[33] K. Fang, N. A. Uhan, F. Z. and Sutherland, J. W. (2016). Scheduling on a single machine under time-of-use electricity tariffs. *Annals of Operations Research*, 238(1-2):199–227.

[34] Kacem, I. (2009). Approximation algorithms for the makespan minimization with positive tails on a single machine with a fixed non-availability interval. *Journal of Combinatorial Optimization*, 17(2):117–133.

[35] Kacem, I. and Chu, C. (2008). Worst-case analysis of the wspt and mwspt rules for single machine scheduling with one planned setup period. *European Journal of Operational Research*, 187(3):1080–1089.

[36] Kacem, I. and Haouari, M. (2009). Approximation algorithms for single machine scheduling with one unavailability period. *4OR: A Quarterly Journal of Operations Research*, 7(1):79–92.

[37] Kacem, I. and Hifi, M. (2009). Makespan minimization on two parallel machines with release dates. *International Conference on Computers & Industrial Engineering, 2009. CIE 2009*, pages 296–299.

[38] Kacem, I. and Kellerer, H. (2011). Fast approximation algorithms to minimize a special weighted flow-time criterion on a single machine with a non-availability interval and release dates. *Journal of Scheduling*, 14(3):257–265.

[39] Kacem, I. and Levner, E. (2016). An improved approximation scheme for scheduling a maintenance and proportional deteriorating jobs. *Journal of Industrial and Management Optimization*, 12(3):811–817.

[40] Kacem, I. and Mahjoub, A. (2009). Fully polynomial time approximation scheme for the weighted flow-time minimization on a single machine with a fixed non-availability interval. *Computers & Industrial Engineering*, 56(4):1708–1712.

[41] Kovalyov, M. and Kubiak, W. (1998). A fully polynomial approximation scheme for minimizing makespan of deteriorating jobs. *Journal of Heuristics*, 3(4):287–297.

[42] L. Wan, J. Y. and Wei, L. (2016). Pareto optimization scheduling with two competing agents to minimize the number of tardy jobs and the maximum cost. *Applied Mathematics and Computation*, 273:912–923.

[43] L. Wan, L. Wei, N. X. J. Y. and Xiong, J. (2017). Pareto optimization for the two-agent scheduling problems with linear non-increasing deterioration based on internet of things. *Future Generation Computer Systems*, 76:293–300.

[44] Leung, J. and Ng, C. (2017). Fast approximation algorithms for uniform machine scheduling with processing set restrictions. *European Journal of Operational Research*, 260(2):507–513.

[45] Lin, B. and Jeng, A. (2004). Parallel-machine batch scheduling to minimize the maximum lateness and the number of tardy jobs. *International Journal of Production Economics*, 91(2):121–134.

[46] Martí, R. and Reinelt, G. (2011). *The Linear Ordering Problem*. Springer-Verlag, Berlin.

[47] O. bdel Raouf, M. A.-B. and El-Henawy, I. (2014). An improved chaotic bat algorithm for solving integer programming problems. *International Journal of Modern Education and Computer Science*, 6:18–24.

[48] Pinedo, M. (2016). *Scheduling Theory, Algorithms, and Systems*. Springer, New York.

[49] Puchinger, J. and Raidl, G. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization. *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 41–53.

[50] Q. Feng, J. Yuan, H. L. and He, C. (2013). A note on two-agent scheduling on an unbounded parallel-batching machine with makespan and maximum lateness objectives. *Applied Mathematical Modelling*, 37(10):7071–7076.

[51] Rudek, R. (2017). Scheduling on parallel processors with varying processing times. *Computers & Operations Research*, 81:90–101.

[52] Sabouni, M. T. Y. and Jolai, F. (2010). Optimal methods for batch processing problem with makespan and maximum lateness objectives. *Applied Mathematical Modelling*, 34(2):314–324.

[53] Sarkar, D. and Modak, J. (2005). Pareto-optimal solutions for multi-objective optimization of fed-batch bioreactors using nondominated sorting genetic algorithm. *Chemical Engineering Science*, 60(2):481–492.

[54] Schuurman, P. and Woeginger, G. (2011). *Lectures on Scheduling*. University of Technology,NL-5600 MB Eindhoven,The Netherlands.

[55] Seifaddini, M. (2014). Absolute and differential approximation algorithms for scheduling under non- availability constraints.

[56] Silva, C. and Biscaia, E. (2003). Genetic algorithm development for multi-objective optimization of batch free-radical polymerization reactors. *Computers & chemical engineering*, 27(8):1329–1344.

[57] Skiena, S. S. (2008). *The Algorithm Design Manual*. Springer Science & Business Media, London.

[58] T. Sen, B. N. B. and Foong, . (1992). An algorithm to minimize maximum job lateness and flowtime in two-machine system. *The 24th Southeastern Symposium on System Theory and The 3rd Annual Symposium on Communications, Signal Processing Expert Systems, and ASIC VLSI Design*, pages 185–188.

[59] Talbi, E. (2009). *Metaheuristics: From Design to Implementation. 2009*, volume 74. John Wiley & Sons.

[60] T'kindt, V. and Billaut, J. (2006). *Multicriteria scheduling: theory, models and algorithms*. Springer Science & Business Media.

[61] Wan, L. (2013). A note on a fully polynomial-time approximation scheme for minimizing makespan of deteriorating jobs. *Mathematical Problems in Engineering*, 2013.

[62] Wikipedia (2016). Np-completeness. [online] https://en.wikipedia.org/wiki/NP-completeness.

[63] Williamson, D. and Shmoys, D. (2010). *The Design of Approximation Algorithms*. Cambridge university press, New York.

# Appendix A

# PTAS Appendix

## A.1 Appendix of small instances

Table A.1 Quality of our PTAS algorithm vs. number of jobs with $\varepsilon$= 0.8

| #jobs | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 1.31 | 9.09 | 1.00040E+00 | 1.01124E+00 | 96.71% |
| 26-50 | 5.44 | 1.19 | 8.34 | 1.01023E+00 | 1.01871E+00 | 89.07% |
| 51-75 | 4.13 | 1.15 | 8.09 | 1.01658E+00 | 1.02202E+00 | 86.11% |
| 76-100 | 2.96 | 1.19 | 7.90 | 1.02448E+00 | 1.02857E+00 | 84.53% |
| 100-200 | 2.55 | 1.13 | 7.98 | 1.01953E+00 | 1.02226E+00 | 82.64% |

Table A.2 Quality of our PTAS algorithm vs. processing time ranges with $\varepsilon$= 0.8

| $p_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.25 | 8.22 | 1.02048E+00 | 1.05111E+00 | 99.43% |
| 1-100 | 3.99 | 1.16 | 8.26 | 1.01959E+00 | 1.02901E+00 | 90.11% |
| 1-500 | 5.25 | 1.15 | 8.28 | 1.01830E+00 | 1.02005E+00 | 53.04% |

Table A.3 Quality of our PTAS algorithm vs. processing time ranges with $\varepsilon$= 0.4

| $p_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.27 | 31.07 | 9.99901E-01 | 1.01025E+00 | 99.86% |
| 1-100 | 3.99 | 1.34 | 31.27 | 9.99782E-01 | 1.00350E+00 | 97.62% |
| 1-500 | 5.25 | 1.77 | 31.21 | 9.99926E-01 | 1.00082E+00 | 92.35% |

Table A.4 Quality of our PTAS algorithm vs. processing time ranges with $\varepsilon$= 0.1

| $p_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.63 | 75.12 | 9.99960E-01 | 1.00287E+00 | 99.97% |
| 1-100 | 3.99 | 2.47 | 74.98 | 9.99934E-01 | 1.00053E+00 | 99.78% |
| 1-500 | 5.25 | 3.30 | 74.88 | 9.99973E-01 | 1.00011E+00 | 98.69% |

Table A.5 Quality of our PTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.8

| $q_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 1.11 | 8.23 | 1.02094E+00 | 1.02137E+00 | 83.19% |
| 1-100 | 2.97 | 1.14 | 8.24 | 1.01672E+00 | 1.01872E+00 | 77.05% |
| 1-500 | 6.59 | 1.32 | 8.28 | 1.01812E+00 | 1.02815E+00 | 82.34% |

Table A.6 Quality of our PTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.4

| $q_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 1.41 | 30.70 | 9.99998E-01 | 1.00013E+00 | 97.55% |
| 1-100 | 2.97 | 1.35 | 31.30 | 9.99968E-01 | 1.00090E+00 | 95.96% |
| 1-500 | 6.59 | 1.61 | 31.56 | 9.99740E-011 | 1.00386E+00 | 96.32% |

Table A.7 Quality of our PTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.1

| $q_i$ ranges | Size of Pareto front | | PTAS | | | |
|---|---|---|---|---|---|---|
| | DP | PTAS | $I'$ jobs | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 2.02 | 74.58 | 1.00000E+00 | 1.00000E+00 | 99.76% |
| 1-100 | 2.97 | 2.05 | 75.15 | 9.99996E-01 | 1.00014E+00 | 99.27% |
| 1-500 | 6.56 | 3.34 | 75.24 | 9.99902E-01 | 1.00072E+00 | 99.41% |

## A.2 Appendix of big instances

Table A.8 Quality of our PTAS algorithm vs. processing time ranges with $\varepsilon$= 0.8

| $p_i$ ranges | Size of Pareto front | | PTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.056 | 1.02836E+00 | 1.03167E+00 | 98.48% |
| 1-100 | 2.111 | 1.056 | 1.03398E+00 | 1.03447E+00 | 40.44% |

Table A.9 Quality of our PTAS algorithm vs. processing time ranges with $\varepsilon$= 0.4

| $p_i$ ranges | Size of Pareto front | | PTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.444 | 1.00036E+00 | 1.00221E+00 | 99.86% |
| 1-100 | 2.111 | 1.111 | 1.00031E+00 | 1.00084E+00 | 95.45% |

Table A.10 Quality of our PTAS algorithm vs. processing time ranges with $\varepsilon$= 0.1

| $p_i$ ranges | Size of Pareto front | | PTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.611 | 1.00000E+00 | 1.00034E+00 | 100.00% |
| 1-100 | 2.111 | 1.722 | 1.00000E+00 | 1.00002E+00 | 99.85% |

Table A.11 Quality of our PTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.8

| $q_i$ ranges | Size of Pareto front | | PTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.083 | 1.03540E+00 | 1.03542E+00 | 67.49% |
| 1-100 | 2.000 | 1.000 | 1.03206E+00 | 1.03251E+00 | 68.78% |
| 1-500 | 2.167 | 1.083 | 1.03150E+00 | 1.03399E+00 | 72.11% |

Table A.12 Quality of our PTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.4

| $q_i$ ranges | Size of Pareto front | | PTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.333 | 1.00007E+00 | 1.00014E+00 | 97.28% |
| 1-100 | 2.000 | 1.250 | 1.00000E+00 | 1.00026E+00 | 98.73% |
| 1-500 | 2.167 | 1.250 | 1.00089E+00 | 1.00284E+00 | 96.95% |

Table A.13 Quality of our PTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.1

| $q_i$ ranges | Size of Pareto front | | PTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^P_{max}/C^*_{max}$ | $L^P_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 2.000 | 1.00000E+00 | 1.00000E+00 | 100.00% |
| 1-100 | 2.000 | 1.833 | 1.00000E+00 | 1.00000E+00 | 99.95% |
| 1-500 | 2.167 | 1.167 | 1.00000E+00 | 1.00023E+00 | 99.83% |

# Appendix B

# FPTAS Appendix

## B.1 Appendix of small instances

Table B.1 Quality of our FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.8

| #jobs | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 1.47 | 1.00522E+00 | 1.01360E+00 | 92.81% |
| 26-50 | 5.44 | 1.56 | 1.00251E+00 | 1.00793E+00 | 94.74% |
| 51-75 | 4.13 | 1.59 | 1.00164E+00 | 1.00559E+00 | 97.34% |
| 76-100 | 2.96 | 1.53 | 1.00098E+00 | 1.00477E+00 | 99.25% |
| 100-200 | 2.55 | 1.54 | 1.00056E+00 | 1.00283E+00 | 98.37% |

Table B.2 Quality of our FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.4

| #jobs | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 1.56 | 1.00252E+00 | 1.00702E+00 | 94.95% |
| 26-50 | 5.44 | 1.59 | 1.00094E+00 | 1.00519E+00 | 96.87% |
| 51-75 | 4.13 | 1.67 | 1.00063E+00 | 1.00438E+00 | 98.72% |
| 76-100 | 2.96 | 1.58 | 1.00044E+00 | 1.00357E+00 | 99.59% |
| 100-200 | 2.55 | 1.58 | 1.00025E+00 | 1.00229E+00 | 99.52% |

Table B.3 Quality of our FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.1

| #jobs | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 1.80 | 1.00028E+00 | 1.00220E+00 | 98.88% |
| 26-50 | 5.44 | 1.72 | 9.99984E-01 | 1.00192E+00 | 99.38% |
| 51-75 | 4.13 | 1.55 | 1.00002E+00 | 1.00095E+00 | 99.73% |
| 76-100 | 2.96 | 1.33 | 1.00002E+00 | 1.00111E+00 | 99.94% |
| 100-200 | 2.55 | 1.24 | 1.00002E+00 | 1.00043E+00 | 99.91% |

Table B.4 Quality of our FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.8

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.45 | 1.00113E+00 | 1.01629E+00 | 99.56% |
| 1-100 | 3.99 | 1.79 | 1.00111E+00 | 1.00962E+00 | 93.38% |
| 1-500 | 5.25 | 1.37 | 1.00117E+00 | 1.00291E+00 | 69.34% |

Table B.5 Quality of our FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.4

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.36 | 1.00042E+00 | 1.01068E+00 | 99.91% |
| 1-100 | 3.99 | 1.86 | 1.00044E+00 | 1.00725E+00 | 99.08% |
| 1-500 | 5.25 | 1.56 | 1.00051E+00 | 1.00214E+00 | 98.48% |

Table B.6 Quality of our FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.1

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.20 | 1.00000E+00 | 1.00000E+00 | 99.97% |
| 1-100 | 3.99 | 1.77 | 1.00004E+00 | 1.00256E+00 | 99.87% |
| 1-500 | 5.25 | 1.55 | 1.00002E+00 | 1.00054E+00 | 99.72% |

Table B.7 Quality of our FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.8

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 1.24 | 1.00109E+00 | 1.00151E+00 | 88.96% |
| 1-100 | 2.97 | 1.51 | 1.00123E+00 | 1.00295E+00 | 85.58% |
| 1-500 | 6.59 | 1.86 | 1.00115E+00 | 1.00910E+00 | 87.74% |

Table B.8 Quality of our FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.4

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 1.26 | 1.00043E+00 | 1.00084E+00 | 99.96% |
| 1-100 | 2.97 | 1.54 | 1.00063E+00 | 1.00206E+00 | 99.34% |
| 1-500 | 6.59 | 1.98 | 1.00042E+00 | 1.00701E+00 | 98.17% |

Table B.9 Quality of our FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.1

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 1.13 | 1.00004E+00 | 1.00017E+00 | 100.00% |
| 1-100 | 2.97 | 1.41 | 1.00005E+00 | 1.00044E+00 | 99.91% |
| 1-500 | 6.59 | 1.98 | 9.99985E-01 | 1.00193E+00 | 99.65% |

## B.2 Appendix of big instances

Table B.10 Quality of our FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.8

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.167 | 1.00001E+00 | 1.00095E+00 | 98.58% |
| 1-100 | 2.111 | 1.278 | 1.00000E+00 | 1.00014E+00 | 43.93% |

Table B.11 Quality of our FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.4

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.000 | 1.00001E+00 | 1.00002E+00 | 99.94% |
| 1-100 | 2.111 | 1.111 | 1.00000E+00 | 1.00003E+00 | 98.73% |

Table B.12 Quality of our FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.1

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.000 | 1.00000E+00 | 1.00000E+00 | 99.98% |
| 1-100 | 2.111 | 1.056 | 1.00000E+00 | 1.00006E+00 | 99.92% |

Table B.13 Quality of our FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.8

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.000 | 1.00000E+00 | 1.00000E+00 | 68.28% |
| 1-100 | 2.000 | 1.167 | 1.00001E+00 | 1.00009E+00 | 72.40% |
| 1-500 | 2.167 | 1.500 | 1.00001E+00 | 1.00076E+00 | 73.07% |

Table B.14 Quality of our FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.4

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.000 | 1.00000E+00 | 1.00000E+00 | 99.93% |
| 1-100 | 2.000 | 1.000 | 1.00001E+00 | 1.00001E+00 | 99.77% |
| 1-500 | 2.167 | 1.167 | 1.00001E+00 | 1.00007E+00 | 98.30% |

Table B.15 Quality of our FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.1

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.000 | 1.00000E+00 | 1.00000E+00 | 100.00% |
| 1-100 | 2.000 | 1.083 | 1.00000E+00 | 1.00014E+00 | 100.00% |
| 1-500 | 2.167 | 1.000 | 1.00000E+00 | 1.00002E+00 | 99.97% |

# Appendix C

# Improved FPTAS Appendix

## C.1 Appendix of small instances

Table C.1 Quality of our improved FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.8

| #jobs | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 1.19 | 1.01094E+00 | 1.02213E+00 | 89.23% |
| 26-50 | 5.44 | 1.15 | 1.01891E+00 | 1.02591E+00 | 81.64% |
| 51-75 | 4.13 | 1.10 | 1.02640E+00 | 1.03410E+00 | 79.58% |
| 76-100 | 2.96 | 1.12 | 1.03763E+00 | 1.04183E+00 | 79.10% |
| 100-200 | 2.55 | 1.08 | 9.22517E-01 | 9.26654E-01 | 76.50% |

Table C.2 Quality of our improved FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.4

| #jobs | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 1.39 | 1.00148E+00 | 1.01135E+00 | 94.74% |
| 26-50 | 5.44 | 1.41 | 1.00092E+00 | 1.00893E+00 | 95.79% |
| 51-75 | 4.13 | 1.34 | 1.00066E+00 | 1.00609E+00 | 97.30% |
| 76-100 | 2.96 | 1.40 | 1.00093E+00 | 1.00357E+00 | 98.62% |
| 100-200 | 2.55 | 1.14 | 9.01855E-01 | 9.04252E-01 | 96.60% |

Table C.3 Quality of our improved FPTAS algorithm vs. number of jobs with $\varepsilon$= 0.1

| #jobs | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 5-25 | 4.74 | 1.70 | 1.00023E+00 | 1.00314E+00 | 98.80% |
| 26-50 | 5.44 | 1.44 | 9.99905E-01 | 1.00179E+00 | 99.17% |
| 51-75 | 4.13 | 1.41 | 9.99972E-01 | 1.00146E+00 | 99.59% |
| 76-100 | 2.96 | 1.16 | 9.99993E-01 | 1.00101E+00 | 99.87% |
| 100-200 | 2.55 | 1.12 | 9.00875E-01 | 9.01843E-01 | 99.83% |

Table C.4 Quality of our improved FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.8

| $p_i$ ranges | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.16 | 5.44336E+00 | 4.89758E+00 | 99.96% |
| 1-100 | 3.99 | 1.12 | 1.57103E+00 | 1.57503E+00 | 98.41% |
| 1-500 | 5.25 | 1.08 | 7.23381E-01 | 7.26678E-01 | 91.82% |

Table C.5 Quality of our improved FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.4

| $p_i$ ranges | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.24 | 5.33070E+00 | 4.78834E+00 | 99.99% |
| 1-100 | 3.99 | 1.47 | 1.53108E+00 | 1.53486E+00 | 99.29% |
| 1-500 | 5.25 | 1.28 | 7.02239E-01 | 7.05176E-01 | 94.87% |

Table C.6 Quality of our improved FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.1

| $p_i$ ranges | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.34 | 1.13 | 5.32395E+00 | 4.76659E+00 | 100.00% |
| 1-100 | 3.99 | 1.52 | 1.52963E+00 | 1.52707E+00 | 99.90% |
| 1-500 | 5.25 | 1.38 | 7.01505E-01 | 7.02666E-01 | 98.88% |

Table C.7 Quality of our improved FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.8

| $q_i$ ranges | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 1.04 | 1.09006E+00 | 1.09264E+00 | 97.30% |
| 1-100 | 2.97 | 1.11 | 8.98335E-01 | 9.02605E-01 | 94.93% |
| 1-500 | 6.59 | 1.22 | 1.09779E+00 | 1.10442E+00 | 97.96% |

Table C.8 Quality of our improved FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.4

| $q_i$ ranges | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 1.19 | 1.05772E+00 | 1.05964E+00 | 98.75% |
| 1-100 | 2.97 | 1.28 | 8.74031E-01 | 8.76893E-01 | 96.53% |
| 1-500 | 6.59 | 1.53 | 1.07209E+00 | 1.07881E+00 | 98.86% |

Table C.9 Quality of our improved FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.1

| $q_i$ ranges | Size of Pareto front | | Improved FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.02 | 1.14 | 1.05646E+00 | 1.05733E+00 | 99.76% |
| 1-100 | 2.97 | 1.33 | 8.73230E-01 | 8.74282E-01 | 99.28% |
| 1-500 | 6.59 | 1.56 | 1.07095E+00 | 1.07133E+00 | 99.75% |

## C.2 Appendix of big instances

Table C.10 Quality of our improved FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.8

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 0.986 | 4.80804E+00 | 4.81219E+00 | 100.00% |
| 1-100 | 2.111 | 1.000 | 2.22129E-01 | 2.23076E-01 | 99.95% |

Table C.11 Quality of our improved FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.4

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.056 | 4.64620E+00 | 4.64684E+00 | 100.00% |
| 1-100 | 2.111 | 1.222 | 2.16033E-01 | 2.17069E-01 | 99.98% |

Table C.12 Quality of our improved FPTAS algorithm vs. processing time ranges with $\varepsilon$= 0.1

| $p_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | FPTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.111 | 4.63680E+00 | 4.63792E+00 | 100.00% |
| 1-100 | 2.111 | 1.000 | 2.15666E-01 | 2.15725E-01 | 99.99% |

Table C.13 Quality of our improved FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.8

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.000 | 1.03293E+00 | 1.03357E+00 | 100.00% |
| 1-100 | 2.000 | 1.000 | 1.04161E+00 | 1.04168E+00 | 99.99% |
| 1-500 | 2.167 | 1.000 | 1.03259E+00 | 1.03681E+00 | 99.94% |

Table C.14 Quality of our improved FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.4

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.167 | 9.96822E-01 | 9.97274E-01 | 100.00% |
| 1-100 | 2.000 | 1.000 | 1.00607E+00 | 1.00621E+00 | 99.99% |
| 1-500 | 2.167 | 1.250 | 1.00304E+00 | 1.00572E+00 | 99.98% |

Table C.15 Quality of our improved FPTAS algorithm vs. delivery time ranges with $\varepsilon$= 0.1

| $q_i$ ranges | Size of Pareto front | | FPTAS | | |
|---|---|---|---|---|---|
| | DP | PTAS | $C^F_{max}/C^*_{max}$ | $L^F_{max}/L^*_{max}$ | $HV_r$ |
| 1-20 | 2.000 | 1.083 | 9.95493E-01 | 9.95652E-01 | 100.00% |
| 1-100 | 2.000 | 1.000 | 1.00453E+00 | 1.00453E+00 | 100.00% |
| 1-500 | 2.167 | 1.083 | 1.00000E+00 | 1.00097E+00 | 99.99% |