Approximation algorithms for minimizing the maximum lateness and makespan on parallel machines

Gais Alhadi, Imed Kacem, Pierre Laroche, Izzeldin Osman

Noname manuscript No.
(will be inserted by the editor)

Approximation Algorithms for Minimizing the Maximum Lateness and Makespan on Parallel Machines

Gais Alhadi · Imed Kacem · Pierre Laroche · Izzeldin M. Osman

the date of receipt and acceptance should be inserted later

Abstract We consider a multiobjective scheduling problem, with the aim of minimizing the maximum lateness and the makespan 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 paper proposes an exact algorithm (based on a dynamic programming) 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 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.

Keywords Approximation · Maximum lateness · Makespan · Dynamic programming · PTAS · FPTAS.

1 Introduction

In this paper we deal with a multiobjective scheduling problem on two parallel machines, with the aim of minimizing the maximum lateness and makespan. Formally, the problem is defined as follows. We have to schedule a set J of n jobs 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

Gais Alhadi · Imed Kacem · Pierre Laroche
Université de Lorraine, LCOMS, F-57045 Metz, France,

Gais Alhadi
University of Gezira, Wad-Madani, Sudan,

Izzeldin M. Osman
Sudan University of Science and Technology, Khartoum, Sudan,
E-mail: gais.alhadi@uofg.edu.sd, alhadiba1@univ-lorraine.fr, imed.kacem@univ-lorraine.fr, pierre.laroche@univ-lorraine.fr, izzeldin@acm.org

time $t = 0$ and each of them can process at most one job at a time. The problem is to find a sequence of jobs, with the objective of minimizing the maximum lateness L_{max} and the makespan C_{max}. With no loss of generality, we consider that all data are integers and that jobs are indexed in non-increasing order of their delivery times $q_1 \geq q_2 \geq \ldots \geq q_n$.
For self-consistency, we recall some necessary definitions related to the approximation area. An algorithm A is called a $\rho-$*approximation algorithm* for a given problem, if for any instance I of that problem the algorithm A yields, within a polynomial time, a feasible solution with an objective value $A(I)$ such that: $A(I) \leq \rho.OPT(I)$, where $OPT(I)$ is the optimal value of I and ρ is the performance guarantee or the worst-case ratio of the *approximation algorithm A*. It can be a real number greater or equal to 1 for the minimization problems $\rho = 1+\varepsilon$ where $\varepsilon > 0$ (that it leads to inequality $A(I) \leq (1+\varepsilon)OPT(I)$), or it can be real number from the interval $[0,1]$ for the maximization problems $\rho = 1-\varepsilon$ (that it leads to inequality $A(I) \geq (1-\varepsilon)OPT(I)$). The reader is invited to consult the papers by Paschos (2018) [18], Demange and Paschos (2005) [7] and Escoffier and Paschos (2010) [8] for more detail on the approximation area. The Pareto-optimal solutions are the solutions that are not dominated by other solutions. Thus, we can consider that the solution is Pareto-optimal if there does not exist another solution that is simultaneously better for all the objectives. Noteworthy, Pareto-optimal solutions represent a range of reasonable optimal solutions for all possible functions based on the different objectives. A schedule is called Pareto-optimal if it is not possible to decrease the value of one objective without increasing the value of the other (see Ben Amor et al. (2017) [4] and Ben Amor and Martel (2014) [3]).

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 the scheduling problems on serial-batch machine, Geng et al. (2018) [12] 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. An approximate Pareto set of minimal size that approximates within an accuracy ε for multi-objective optimization problems have been studied by Bazgan et al. (2015) [2]. They proposed a 3-approximation algorithm for two objectives and also proposed a study of the greedy algorithm performance for a three-objective case when the points are given explicitly in the input. They showed that the three-objective case is NP-hard. Chen and Zou (2014) [6] 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. Florios and Mavrotas (2014) [9] used AUGMECON2, a multi-objective mathematical programming method (which is suitable for general multi-objective integer programming problems), to produce all the Pareto-optimal solutions for multi-objective traveling salesman and set covering problems. They showed that the performance of the algorithm is slightly better than the existing methods. Moreover, they showed that their results can be helpful for other multi-objective mathematical programming methods or even multi-objective meta-heuristics. In [19], Sabouni and Jolai (2010) proposed an optimal method for the problem of scheduling jobs on a single batch processing machine to minimize the

makespan and the maximum lateness. They showed that the proposed method is optimal when the set with maximum lateness objective has the same processing times. They also proposed an optimal method for the group that has the maximum lateness objective and the same processing times. Geng and Yuan (2015) [11] considered the scheduling problem on an unbounded p-batch machine with family jobs to find all Pareto-optimal points for minimizing the makespan and the maximum lateness. They presented a dynamic programming algorithm to solve the studied problem. He et al. (2015) [13] showed that the Pareto optimization scheduling problem on a single bounded serial-batching machine to minimize the makespan and the maximum lateness is solvable in $O(n^6)$. They also presented an $O(n^3)$-time algorithm to find all Pareto optimal solutions where the processing times and deadlines are agreeable. For the bi-criteria scheduling problem, He et al. (2013) [14] showed that the problem of minimizing the maximum cost and the makespan is solvable in $O(n^5)$ time. The authors presented a polynomial-time algorithm in order to find all Pareto optimal solutions. Also, He et al. (2013) [16] showed that the bi-criteria batching problem of minimizing the maximum cost and the makespan is solvable in $O(n^3)$ time. The bi-criteria scheduling problem on a parallel-batching machine to minimize the maximum lateness and the makespan have been considered by He et al. (2007) in [15]. The authors presented a polynomial-time algorithm in order to find all Pareto optimal solutions. Allahverdi and Aldowaisan (2004) [1] studied the no-wait flow-shop scheduling problem with bi-criteria of makespan or maximum lateness. They also proposed a dominance relation and a branch-and-bound algorithm and showed that these algorithms are quite efficient.

From the presented state-of-the-art, we can conclude that the multi-objective problem we consider in this paper 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 paper is organized as follows. In Section 2, we describe the proposed dynamic programming algorithm. Section 3 provides the description and the analysis of the PTAS based on the merging technique. Section 4 presents the steps and the analysis of the FPTAS. In Section 5 is described an improved FPTAS based on the merging technique. Section 6, presents some results obtained by using our algorithms. Finally, Section 7 concludes the paper.

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 every iteration j, a set χ_j composed of states is generated $(0 \leq j \leq n)$. Each state $[k, L_{max}, C_{max}]$ in χ_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 1.

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

Algorithm 1 Algorithm A

Set $\chi_1 = \{[1, p_1 + q_1, p_1]\}$.
for $j \in \{2, 3, \ldots, n\}$ **do**
  $\chi_j = \emptyset$
  for every state $[k, L_{max}, C_{max}]$ in χ_{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 χ_j

    (schedule job j on machine $1-k \Rightarrow C_{max}$ stays on machine k or C_{max} goes to machine $1-k$)
    if $(C_{max} \geq \sum_{i=1}^{j} p_i - C_{max})$ **then**
      add $[k, max\{L_{max}, \sum_{i=1}^{j} p_i - C_{max} + q_j\}, C_{max}]$ to χ_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 χ_j
    end if
  end for
  for every k **do**
    for every C_{max} **do**
      keep only one state with the smallest possible L_{max}
    end for
  end for
  Remove χ_{j-1}
end for
Return the Pareto front of χ_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)

C_{max} and k in χ_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.

3 New simplifications and PTAS

In this section, we describe our PTAS (Polynomial Time Approximation Scheme). It uses a simplification technique based on merging small jobs, inspired from Kacem and Kellerer (2014) [17]. This PTAS is described by Algorithm 2.

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 ε 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 round up every tail q_j to $q_j := \lfloor \lceil \frac{q_j}{\varepsilon q_{max}/2} \rceil \cdot \frac{\varepsilon q_{max}}{2} \rfloor$ where $q_{max} = max_{j \in J}\{q_j\}$. The new instance is denoted as I'.

Proposition 1 *The obtained instance I' can be obtained in $O(n)$ time and it can be done with no more than $(1 + \varepsilon/2)-$loss.*

Algorithm 2 PTAS Algorithm

1. Let $\varepsilon > 0$ (assume that $\frac{2}{\varepsilon}$ is an integer) and $J = \{1,2,\ldots,n\}$.
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 $\lfloor k \cdot \varepsilon q_{max} \rfloor$.
3. For every class $C_k (1 \leq k \leq 2/\varepsilon)$:
Divide the set of all available jobs into two subsets as follows:

The small jobs in S, where $S = \{j \in J | p_j < \varepsilon^2 P/8\}$
The large jobs in G, where $G = \{j \in J | p_j \geq \varepsilon^2 P/8)\}$

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.

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 reduction is done by merging 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 $\varepsilon^2 P/8$ and $\varepsilon^2 P/4$. The small jobs are taken in the order of their index in this merging procedure. At the end, it remains at most a single small job for each class C_k. At most, in the new instance, $(\frac{8}{\varepsilon^2} + \frac{2}{\varepsilon})$ 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 1 *The studied problem has a Polynomial Time Approximation Scheme (PTAS) with a time complexity of* $O(n \cdot 2^{(\frac{8}{\varepsilon^2} + \frac{2}{\varepsilon})})$.

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 Δ_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 Δ'_k), which is "close" enough to the value Δ_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 Δ_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 Δ_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 FPTAS

The main idea of our FPTAS is to remove a special part of the states generated by the dynamic programming algorithm. Therefore, the modified algorithm A' produces an approximation solution instead of the optimal solution (see the detail in Algorithm 3).

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 δ_1 and δ_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

Algorithm 3 FPTAS Algorithm A'

Let $\varepsilon > 0$ and $J = \{1, 2, \ldots, n\}$.
Split the interval $[0, C_{max}]$ and $[0, L_{max}]$ into sub-intervals respectively of sizes δ_1 and δ_2.
Set $\chi_1^{\#} = \{[1, p_1 + q_1, p_1]\}$.
for $j \in \{2, 3, \ldots, n\}$ **do**
  $\chi_j^{\#} = \phi$.
  for every state $[k, L_{max}, C_{max}]$ in $\chi_{j-1}^{\#}$ **do**
    (schedule job j 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 $1 - k$)
    if $(C_{max} \geq \sum_{i=1}^{j} p_i - C_{max})$ **then**
      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 $\chi_j^{\#}$
    end if
  end for
  Keep only one representative state for every couple of the defined sub-intervals produced from $[0, CMAX]$ and $[0, LMAX]$.
  Remove $\chi_{j-1}^{\#}$.
end for
Return the Pareto front from the solutions obtained in previous step, by only keeping non-dominated states.

$[0, LMAX]$. Thus, our FPTAS will generate approximate sets $\chi_j^{\#}$of states instead of χ_j. The following lemma shows the closeness of the result generated by the FPTAS compared to the dynamic programming.

Lemma 1 *For every state $[k, L_{max}, C_{max}] \in \chi_j$ there exists at least one approximate state $[m, L_{max}^{\#}, C_{max}^{\#}] \in \chi_j^{\#}$ such that:*

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

and

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

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 holds true up to level $j-1$. Consider an arbitrary state $[k, L_{max}, C_{max}] \in \chi_j$. Algorithm A introduces this state into χ_j when job j is added to some feasible state 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 will prove the statement 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. However, it may be removed when reducing the state subset. 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:

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

In addition,

$$\mu \leq C'^{\#}_{max}+p_j+\delta_1 \leq C'_{max}+(j-1)\delta_1+p_j+\delta_1 = C_{max}+j\delta_1. \qquad (2)$$

and,

$$\mu \geq C'^{\#}_{max}+p_j-\delta_1 \geq C'_{max}-(j-1)\delta_1+p_j-\delta_1 \geq C_{max}-j\delta_1. \qquad (3)$$

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. However, it may be removed when reducing the state subset. 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:

$$\begin{aligned}
\lambda &\leq \max\{L'^{\#}_{max}, \sum_{i=1}^{j} p_i - C'^{\#}_{max}+q_j\}+\delta_2 \\
&\leq \max\{L'_{max}+(j-1)\max\{\delta_1,\delta_2\}, \sum_{i=1}^{j} p_i - (C'_{max}-(j-1)\delta_1)+q_j\}+\delta_2 \\
&\leq \max\{L'_{max}, \sum_{i=1}^{j} p_i - C'_{max}+q_j\}+(j-1)\max\{\delta_1,\delta_2\}+\delta_2 \\
&\leq L_{max}+(j-1)\max\{\delta_1,\delta_2\}+\delta_2 \\
&< L_{max}+j\max\{\delta_1,\delta_2\} \qquad (4)
\end{aligned}$$

Moreover,

$$\mu \leq C'^{\#}_{max}+\delta_1 \leq C'_{max}+(j-1)\delta_1+\delta_1 = C_{max}+j\delta_1. \qquad (5)$$

And,

$$\mu \geq C'^{\#}_{max}-\delta_1 \geq C'_{max}-(j-1)\delta_1-\delta_1 = C_{max}-j\delta_1. \qquad (6)$$

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. However, it may be removed when reducing the state subset. 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:

$$\begin{aligned}
\lambda &\leq \max\{L'^{\#}_{max}, \sum_{i=1}^{j} p_i - C'^{\#}_{max} + q_j\} + \delta_2 \\
&\leq \max\{L'_{max} + (j-1)\max\{\delta_1, \delta_2\}, \sum_{i=1}^{j} p_i - (C'_{max} - (j-1)\delta_1) + q_j\} + \delta_2 \\
&\leq \max\{L'_{max}, \sum_{i=1}^{j} p_i - C'_{max} + q_j\} + (j-1)\max\{\delta_1, \delta_2\} + \delta_2 \\
&\leq L_{max} + (j-1)\max\{\delta_1, \delta_2\} + \delta_2 \\
&< L_{max} + j\max\{\delta_1, \delta_2\}
\end{aligned} \tag{7}$$

Moreover,

$$\mu \leq \sum_{i=1}^{j} p_i - C'^{\#}_{max} + \delta_1 \tag{8}$$

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

$$\mu \leq \sum_{i=1}^{j} p_i - C'_{max} + j\delta_1 \leq C_{max} + j\delta_1 (\text{since} \sum_{i=1}^{j} p_i - C'_{max} \leq C'_{max}). \tag{9}$$

And,

$$\mu \geq \sum_{i=1}^{j} p_i - C'^{\#}_{max} - \delta_1 \geq C'^{\#}_{max} - \delta_1 \geq C_{max} - j\delta_1. \tag{10}$$

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. However, it may be removed when reducing the state subset. 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:

$$\begin{aligned}
\lambda &\leq \max\{L'^{\#}_{max}, \sum_{i=1}^{j} p_i - C'^{\#}_{max} + q_j\} + \delta_2 \\
&\leq \max\{L'_{max} + (j-1)\max\{\delta_1,\delta_2\}, \sum_{i=1}^{j} p_i - (C'_{max} - (j-1)\delta_1) + q_j\} + \delta_2 \\
&\leq \max\{L'_{max}, \sum_{i=1}^{j} p_i - C'_{max} + q_j\} + (j-1)\max\{\delta_1,\delta_2\} + \delta_2 \\
&\leq L_{max} + (j-1)\max\{\delta_1,\delta_2\} + \delta_2 \\
&\leq L_{max} + j\max\{\delta_1,\delta_2\} \qquad (11)
\end{aligned}$$

and

$$\mu \leq \sum_{i=1}^{j} p_i - C'^{\#}_{max} + \delta_1 \leq \sum_{i=1}^{j} p_i - (C'_{max} - (j-1)\delta_1) + \delta_1 \leq C_{max} + j\delta_1. \qquad (12)$$

In the other hand, we have

$$\mu \geq \sum_{i=1}^{j} p_i - C'^{\#}_{max} - \delta_1 \geq \sum_{i=1}^{j} p_i - (C'_{max} + (j-1)\delta_1) - \delta_1 \geq C_{max} - j\delta_1. \qquad (13)$$

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. However, it may be removed when reducing the state subset. 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:

$$\begin{aligned}
\lambda &\leq \max\{L'^{\#}_{max}, \sum_{i=1}^{j} p_i - C'^{\#}_{max} + q_j\} + \delta_2 \\
&\leq \max\{L'_{max} + (j-1)\max\{\delta_1,\delta_2\}, \sum_{i=1}^{j} p_i - (C'_{max} - (j-1)\delta_1) + q_j\} + \delta_2 \\
&\leq \max\{L'_{max}, \sum_{i=1}^{j} p_i - C'_{max} + q_j\} + (j-1)\max\{\delta_1,\delta_2\} + \delta_2 \\
&\leq L_{max} + (j-1)\max\{\delta_1,\delta_2\} + \delta_2 \\
&\leq L_{max} + j\max\{\delta_1,\delta_2\} \qquad (14)
\end{aligned}$$

and

$$\mu \leq C'^{\#}_{max} + \delta_1 \leq C'_{max} + (j-1)\delta_1 + \delta_1 \leq C_{max} + j\delta_1. \qquad (15)$$

In the other hand, we have

$$\mu \geq C'^{\#}_{max} - \delta_1 \geq \sum_{j=1}^{i} p_j - C'^{\#}_{max} - \delta_1 \geq \sum_{i=1}^{j} p_i - C'_{max} - j\delta_1 = C_{max} - j\delta_1. \qquad (16)$$

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

In conclusion, the statement holds also for level i in the third case, and this completes our inductive 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 2 *The studied problem has a Fully Polynomial Time Approximation Scheme (FPTAS) with a time complexity of* $O(n^3/\varepsilon^2)$.

5 Improved FPTAS

In this section, we describe an improved FPTAS (Algorithm 4). 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 6, we will present some numerical experiments in order to compare the proposed algorithms.

Algorithm 4 Improved FPTAS

Let $\varepsilon > 0$ (assume that $\frac{2}{\varepsilon}$ is an integer)
Let $J = \{1, 2, \ldots, n\}$.
Split the interval $[0, \max_{j \in J}\{q_j\}]$ in $\frac{2}{\varepsilon}$ equal-length sub-intervals.
for every $j \in C_k (1 \leq k \leq 2/\varepsilon)$ **do**
  Round up every tail $q_j$as follows: $q_j := \lfloor \lceil \frac{q_j}{\varepsilon q_{max}/2} \rceil \cdot \frac{\varepsilon q_{max}}{2} \rfloor$.
  Divide the set of all available jobs into two subsets as follows:
    The small jobs S, where $S = \{j \in J | p_j < \varepsilon^2 P/8\}$.
    The large jobs G, where $G = \{j \in J | p_j \geq \varepsilon^2 P/8)\}$.
  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.
end for
After merging the small jobs, the new instance I'' will be solved using the FPTAS algorithm presented in Algorithm 3

6 Results

The following results have been obtained after testing the performance of the proposed algorithms. 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 ε: 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) [5] 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 paper, we use a program by Fonseca et al. (2018) [10] 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) [5]). 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. 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%

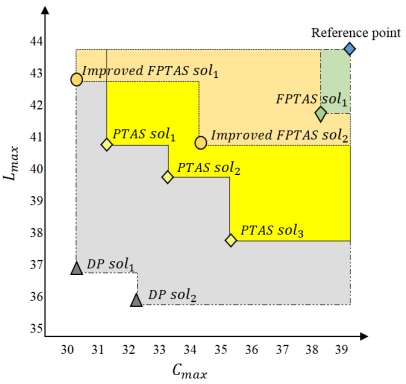


Fig. 1: Comparison of two dimensions Pareto sets using Hypervolume

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 6.1, we will present the results of the small instances. The results of the big instances are presented in Subsection 6.2. Finally, Subsection 6.3 compares the results of our algorithms.

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).

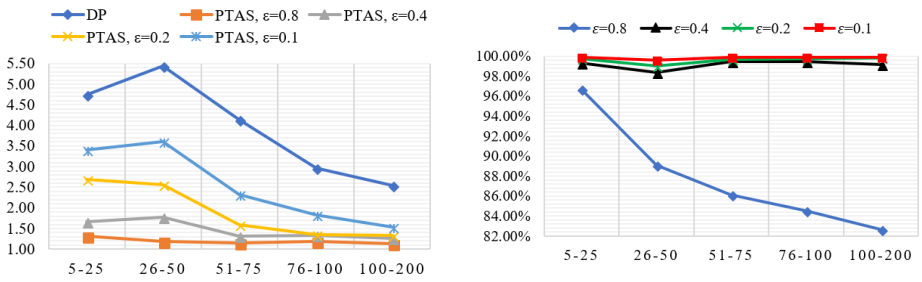
It is worth mentioning that we will present the results of the PTAS in Subsection 6.1.1. Subsection 6.1.2 provides the results of the FPTAS. Finally, the improved FPTAS results will be presented in Subsection 6.1.3.

6.1.1 Results of our PTAS algorithm vs. DP algorithm

6.1.1.1 Quality of our PTAS algorithm.

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

Fig. 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 ε value, the size of the Pareto front is near the size of the DP, and with big ε value, we lose a lot of solutions (see Fig. 2(a)). We can see that with good ε (small ε), we have good solutions (see Fig. 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 Ratios (HV_r)

Fig. 2: Quality of our PTAS algorithm with $\varepsilon = 0.8, 0.4, 0.2$ and 0.1

Another remark can be done from Fig. 2: the results of $\varepsilon = 0.8$ are very bad. First, (Fig. 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. 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 1 (ε=0.8), whatever the size of instances, we approximately have the same number of jobs in the simplified instance I'. Indeed, when ε 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 ε 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 ε, instance I' becomes very close to the original instance I (see columns 4, 5, and 6 in Table 1). Thus, in the following, we will not present the results for $\varepsilon = 0.8$.

Table 1: Average number of jobs in the simplified instance I' with ε = 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 2, 3, and 4 present the results of our PTAS as a function of the number of jobs, for three different values of ε (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 ε is small (see the tables 3 and 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 2: Quality of our PTAS algorithm vs. number of jobs with ε= 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 3: Quality of our PTAS algorithm vs. number of jobs with ε= 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: Quality of our PTAS algorithm vs. number of jobs with ε= 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 ε = 0.8, 0.4 and 0.1.

We have also studied the influence of processing time and delivery time in Table 5 and 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 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 5: Quality of our PTAS algorithm vs. processing time ranges with ε= 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 6: Quality of our PTAS algorithm vs. delivery time ranges with ε= 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 $I = 75.5$.

6.1.1.2 Average running times of our PTAS vs. DP

In this subsection, the average running times are given in the figures 3 and 4. They compare the DP and PTAS algorithms, considering different values of ε = 0.8, 0.4, 0.2 and 0.1. All values are in seconds.

Fig. 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 ε grows, which is consistent with theory. Hence, as we have seen before, we can get a very good solution when ε is small, but, as it can be seen in Fig. 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.

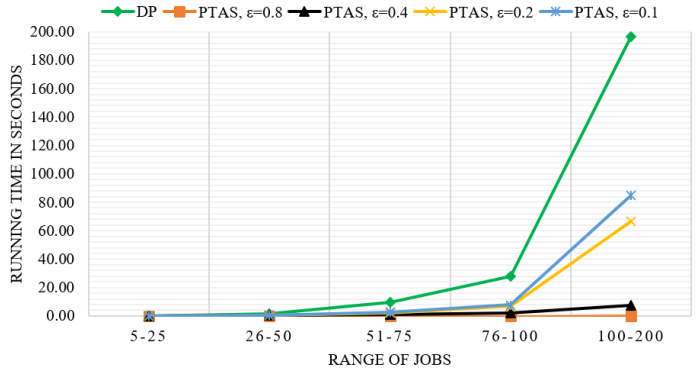
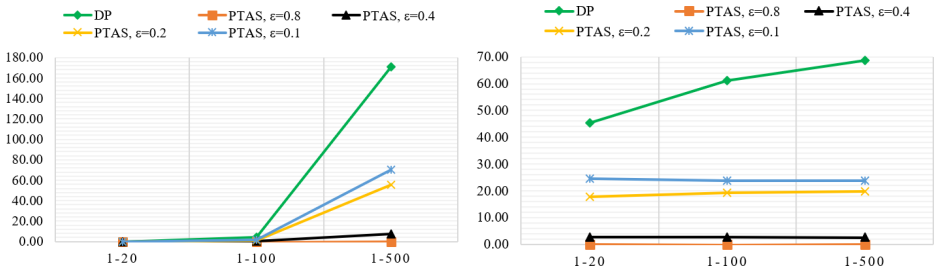


Fig. 3: PTAS: Average running times (s) vs. size of instances

We have also studied the average running times depending on processing and delivery times, in Fig. 4. 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 ε grows, which is consistent with theory. As expected, our PTAS is faster than DP algorithm, especially with big values of ε. The running times are consistent with the complexity $O(n \cdot 2^{(\frac{8}{\varepsilon^2}+\frac{2}{\varepsilon})})$.



(a) Running times vs. processing times

(b) Running times vs. delivery times

Fig. 4: Average running times (s) vs. processing and delivery times ranges

6.1.2 Results of our FPTAS algorithm vs. DP algorithm

6.1.2.1 Quality of our FPTAS algorithm

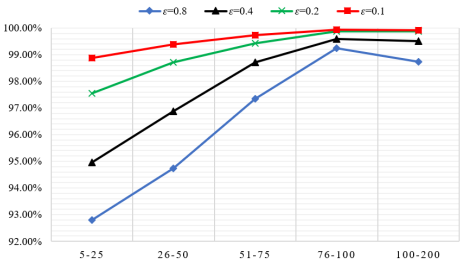
In this section, we will see the quality of our FPTAS using Hypervolume ratios. We will also present the results for the average values of L_{max} and C_{max}.

Fig. 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. 5(a), we can see that, with a small ε value, the size of Pareto front obtained by our FPTAS algorithm decreases as the number of jobs increases, with big ε value, we do not have the same behavior. Indeed, when ε is big, the length of FPTAS intervals will be very big, so we remove some solutions. Hence, we can see that with good ε (small ε), we have a good solution (See Fig. 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 Ratios (HV_r)

Fig. 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 6.1.1.1. However, it still has a less quality than ε = 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 ε = 0.8, 0.4 and 0.1.

As we can see in Table 7, as a function of a number of jobs, we have the same previous behavior of the PTAS in the section 6.1.1.1. The FPTAS can find solutions closer to the optimal ones when the number of jobs increases, while C_{max} and L_{max} decreases.

Table 7: Quality of our FPTAS algorithm vs. number of jobs with ε= 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 8 and 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 8: Quality of our FPTAS algorithm vs. processing time ranges with ε= 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 9: Quality of our FPTAS algorithm vs. delivery time ranges with ε= 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%

6.1.2.2 Average running times of our FPTAS vs. DP
In this section, the average running times are given in the figures 6 and 7. They compare our DP and FPTAS, considering different values of ε = 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. 6, our FPTAS is slower when the number of states is growing. Hence, the results are consistent with PTAS.

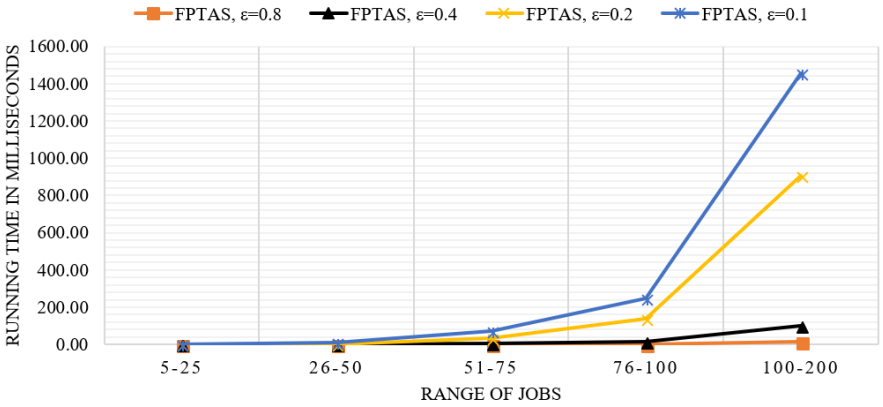
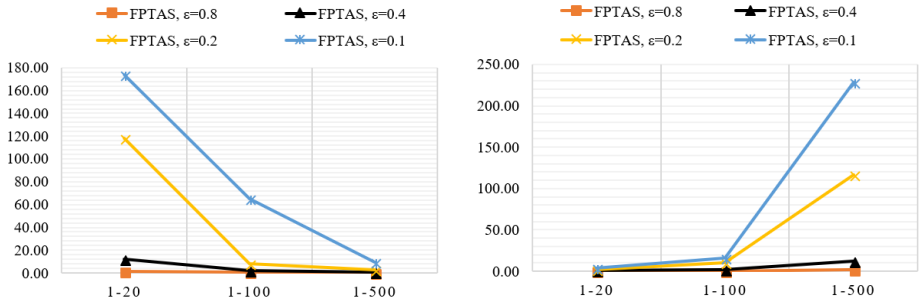


Fig. 6: FPTAS: Average running times (ms) vs. size of instances

We have also studied the average running times of processing and delivery times, in Fig. 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).

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



(a) Running times vs. processing times

(b) Running times vs. delivery times

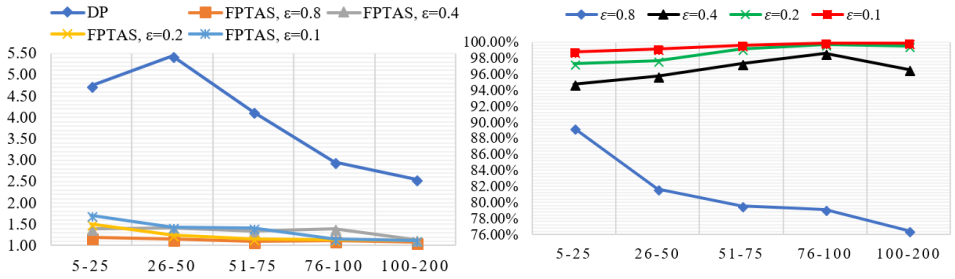
Fig. 7: Average running times (ms) vs. processing and delivery times ranges

6.1.3 Results of the improved FPTAS vs. the DP algorithm

6.1.3.1 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. 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. 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. 8(b), the improved FPTAS algorithm achieves better results with small ε 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 Ratios (HV_r)

Fig. 8: Quality of the improved FPTAS 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 ε = 0.8, 0.4 and 0.1.

As we can see in Table 10, as a function of the number of jobs, the improved FPTAS has the same analysis of the PTAS results in section 6.1.1.1.

Table 10: Quality of the improved FPTAS vs. number of jobs with ε= 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 11 and 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 11: Quality of the improved FPTAS vs. processing time ranges with ε= 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 12: Quality of the improved FPTAS vs. delivery time ranges with ε= 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%

6.1.3.2 Average running times of our improved FPTAS vs. DP

In this section, the average running times are given in the figures 9 and 10. They compare our DP algorithm and improved FPTAS, considering different values of ε = 0.8, 0.4, 0.2 and 0.1. All values are in milliseconds.

Fig. 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.

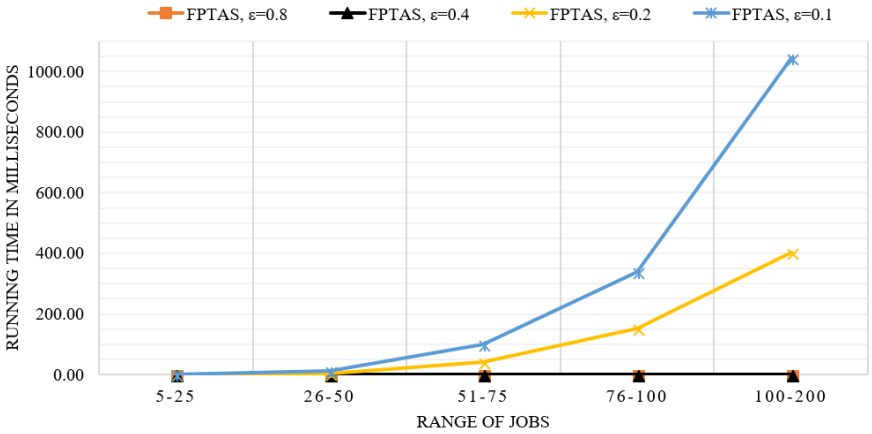
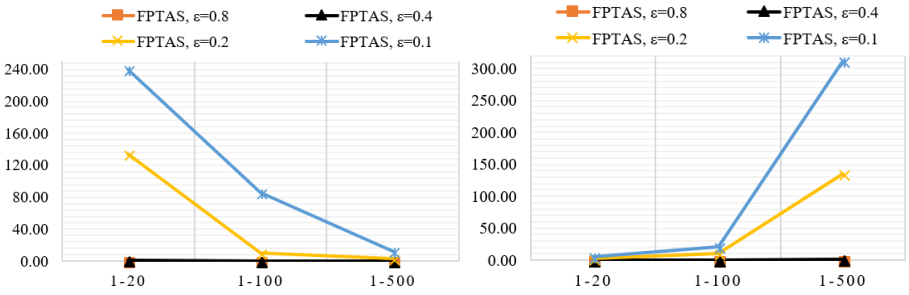


Fig. 9: FPTAS: Average running times (ms) vs. size of instances

We have also studied the average running times of processing and delivery times, in Fig. 10. Indeed, the analysis of the results is the same as in the subsection 6.1.2.2 (see Fig. 7).



(a) Running times vs. processing times

(b) Running times vs. delivery times

Fig. 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 ε grows, which is consistent with theory. As expected, the improved FPTAS is faster than the DP algorithm, especially with big values of ε.

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 13, 14 and 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 ε value grows. The FPTAS has an opposite behavior.

Table 13: Quality of our PTAS with 1000 jobs

ε 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 14: Quality of our FPTAS with 1000 jobs

ε 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 15: Quality of our improved FPTAS with 1000 jobs

ε 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 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 ε is big, which is consistent with the theory.

Furthermore, the improved FPTAS has better running times compared with the PTAS and FPTAS algorithms.

Table 16: Average running times (seconds) with ε = 0.8, 0.4, 0.2 and 0.1

DP time	ε 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 17, 18, 19, 20, 21 and 22 (the analysis of the results are the same with ε=0.1, 0.4 and 0.8). Moreover, the results of running times are consistent with the results of smaller instances.

Table 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 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 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 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 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 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

6.3 Comparison of the algorithms' results

6.3.1 Quality of our algorithms

As a function of a number of jobs: as we can see in Fig. 11, with big instances, the FPTAS has better quality than the PTAS and the improved FPTAS. But, with small instances, as mentioned in Section 6.1.1.1, the PTAS achieves better results as we decrease the number of jobs.

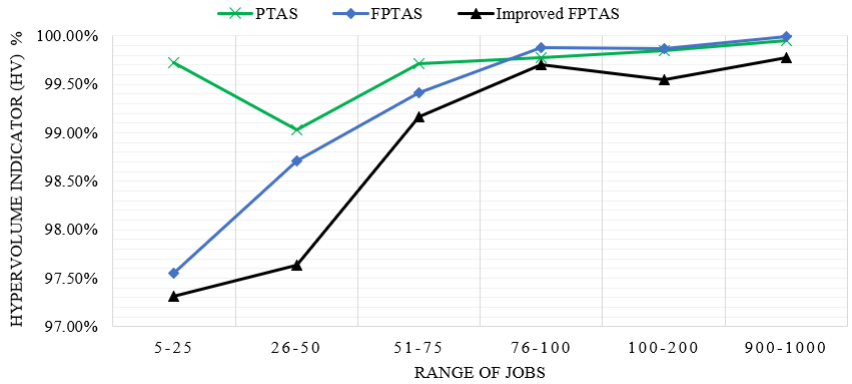


Fig. 11: Quality of our algorithms as a function of a number of jobs with ε=0.2

As a function of processing time: with big instances, the improved FPTAS always has better quality than PTAS and FPTAS (see Section 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. 12).

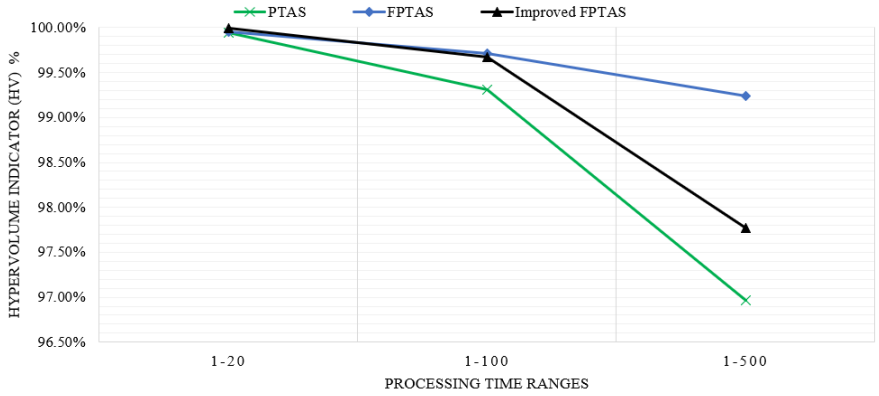


Fig. 12: Quality of our algorithms as a function of processing time with ε=0.2

As a function of delivery time: 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. 13).

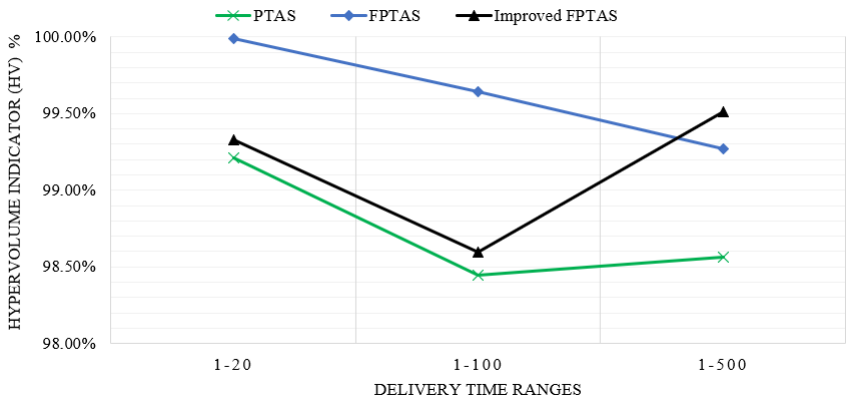


Fig. 13: Quality of our algorithms as a function of delivery time with ε=0.2

6.3.2 Average running times of our algorithms

In general, as we have seen in Section 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 6.1.1.2, 6.1.2.2, and 6.1.3.2, when ε is too big, the average running times given by the improved FPTAS is smaller than the DP, PTAS, and FPTAS.
In fact, as a function of a number of jobs, all algorithms are slower when the number of jobs increases. Hence, when ε is big, the improved FPTAS is faster than other algorithms. On the other side, when ε is small, the FPTAS is better, but, the difference is not big.

Moreover, as a function of processing time, when the PTAS algorithm becomes slower, the FPTAS and improved FPTAS have an opposite behavior and vice versa.

In addition, as a function of delivery times, when ε is big, the improved FPTAS is faster than other algorithms. On the other side, when ε and delivery times are smaller, the FPTAS is better, but, the difference is not too big.

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}). We have also shown the importance of the values of processing times, delivery times, and ε value.

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

References

1. A. Allahverdi and T. Aldowaisan. No-wait flowshops with bicriteria of makespan and maximum lateness. *European Journal of Operational Research*, Vol 152(1), pp 132–147 (2004).
2. C. Bazgan, F. Jamain, and D. Vanderpooten. Approximate Pareto sets of minimal size for multi-objective optimization problems. *Operations Research Letters*, Vol 43(1), pp 1–6 (2015).
3. S. Ben Amor, J.-M. Martel. A new distance measure including the weak preference relation: Application to the multiple criteria aggregation procedure for mixed evaluations. *European Journal of Operational Research*, Vol 237(3), pp 1165–1169 (2014).
4. S. Ben Amor, K. Zaras, E. A. Aguayo. The value of additional information in multicriteria decision making choice problems with information imperfections. *Annals of Operations Research*, Vol 253(1), pp 61–76 (2017).
5. L. Bradstreet. The hypervolume indicator for multi-objective optimisation: calculation and use. University of Western Australia (2011).
6. Y. Chen and X. Zou. Runtime analysis of a multi-objective evolutionary algorithm for obtaining finite approximations of Pareto fronts. *Information Sciences*, Vol 262, pp 62–77 (2014).
7. M. Demange and V. Th. Paschos. Polynomial approximation algorithms with performance guarantees: An introduction-by-example. *European Journal of Operational Research*, Vol 165(3), pp 555–568 (2005).
8. B. Escoffier and V. Th. Paschos. A survey on the structure of approximation classes. *Computer Science Review*, Vol 4(1), pp 19–40 (2010)
9. K. Florios and G. Mavrotas. Generation of the exact Pareto set in Multi-Objective Traveling Salesman and Set Covering Problems. *Applied Mathematics and Computation*, Vol 237, pp 1–19 (2014).
10. Carlos M. Fonseca, Manuel López-Ibáñez, Luís Paquete and Andreia P. Guerreiro. Computation of the Hypervolume Indicator. http://lopez-ibanez.eu/hypervolume,(2018).
11. Z. Geng and J. Yuan. Pareto optimization scheduling of family jobs on a p-batch machine to minimize makespan and maximum lateness. *Theoretical Computer Science*, Vol 570, pp 22–29 (2015).
12. Z. Geng, J. Yuan and J. Yuan. Scheduling with or without precedence relations on a serial-batch machine to minimize makespan and maximum cost. *Applied Mathematics and Computation*, 332, pp.1-18 (2018).
13. C. He, H. Lin, and Y. Lin. Bounded serial-batching scheduling for minimizing maximum lateness and makespan. *Discrete Optimization*, Vol 16, pp 70–75 (2015).
14. C. He, H. Lin, Y. Lin, and J. Tian. Bicriteria scheduling on a series-batching machine to minimize maximum cost and makespan. *Central European Journal of Operations Research*, pp 1–10 (2013).
15. C. He, Y. Lin, and J. Yuan. Bicriteria scheduling on a batching machine to minimize maximum lateness and makespan. *Theoretical Computer Science*, Vol 381(1-3), pp 234–240 (2007).
16. C. He, X. M. Wang, Y. X. Lin, and Y. D. Mu. An Improved Algorithm for a Bicriteria Batching Scheduling Problem. *RAIRO-Operations Research*, Vol 47(1), pp1–8 (2013).
17. I. Kacem and H. Kellerer. Approximation algorithms for no-idle time scheduling on a single machine with release dates and delivery times. *Discrete Applied Mathematics*, Vol 164, pp 154–160 (2014)
18. V. Th. Paschos. Combinatorial approximation of maximum k-vertex cover in bipartite graphs within ratio 0.7. *RAIRO - Operations Research*, Vol 52, pp 305–314 (2018)
19. M. T. Y. Sabouni and F. Jolai. Optimal methods for batch processing problem with makespan and maximum lateness objectives. *Applied Mathematical Modelling*, Vol 34(2), pp 314–324 (2010).