Available online at www.sciencedirect.com

ScienceDirect

Discrete Applied Mathematics 155 (2007) 945–970

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Scheduling orders for multiple product types to minimize total weighted completion time

Joseph Y.-T. Leung[a], Haibing Li[a], Michael Pinedo[b]

[a] *Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA*
[b] *Stern School of Business, New York University, 40 West Fourth Street, New York, NY 10012, USA*

Received 2 September 2005; received in revised form 24 May 2006; accepted 27 September 2006
Available online 20 November 2006

Abstract

We consider the problem of scheduling orders for multiple different product types in an environment with m dedicated machines in parallel. The objective is to minimize the total weighted completion time. Each product type is produced by one and only one of the m dedicated machines; that is, each machine is dedicated to a specific product type. Each order has a weight and may also have a release date. Each order asks for certain amounts of various different product types. The different products for an order can be produced concurrently. Preemptions are not allowed. Even when all orders are available at time 0, the problem has been shown to be strongly NP-hard for any fixed number ($\geqslant 2$) of machines. This paper focuses on the design and analysis of efficient heuristics for the case without release dates. Occasionally, however, we extend our results to the case with release dates. The heuristics considered include some that have already been proposed in the literature as well as several new ones. They include various static and dynamic priority rules as well as two more sophisticated LP-based algorithms. We analyze the performance bounds of the priority rules and of the algorithms and present also an in-depth comparative analysis of the various rules and algorithms. The conclusions from this empirical analysis provide insights into the trade-offs with regard to solution quality, speed, and memory space.

Keywords: Order scheduling; Approximation algorithms; Design and analysis of algorithms

1. Introduction

Consider a facility with m different machines in parallel with each machine being dedicated to produce one and only one particular product type. Assume there are n orders that come in from n different clients at different times (release dates). Order j, $j = 1, 2, \ldots, n$, requests the product type i, $i = 1, 2, \ldots, m$, a quantity which requires a production or processing time $p_{ij} \geqslant 0$ on machine i. The processing of the various products for order j may be done on the dedicated machines concurrently. Order j has a weight w_j and may also have a release date r_j. Preemptions are not allowed. If the orders have different release dates, then unforced idleness is allowed, i.e., the decision-maker is allowed to keep a machine idle in anticipation of an order with a high weight coming in, even though another order may be waiting for its products. The completion time of order j is denoted by C_j and is the time when the processing of all the products for order j have been completed. If we denote C_{lj} as the finish time of job l of order j, it is clear that $C_j = \max_l\{C_{lj}\}$.

E-mail addresses: leung@njit.edu (J.Y.-T. Leung), hl27@njit.edu (H. Li), mpinedo@stern.nyu.edu (M. Pinedo).

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2006.09.012

With this notation, various objective functions of C_j can be defined similar to those for the classic scheduling model. However, in this paper we focus on the total weighted completion time objective, i.e., $\sum w_j C_j$.

The problem described above is classified as the *Order Scheduling Model*, which has numerous application areas, see Leung et al. [10,11] and Sung and Yoon [18]. The application areas range from order scheduling in manufacturing environments to the maintenance of large airplanes. Following the notation proposed by Leung et al. [10], we refer to the problem described above as $PD\|\sum w_j C_j$ when the number of machines m is arbitrary, and as $PDm\|\sum w_j C_j$ when m is fixed. If release date presents, we denote the two cases as $PD|r_j|\sum w_j C_j$ and $PDm|r_j|\sum w_j C_j$, respectively.

When all r_j are zero an important structural property can be shown to hold for this class of problems, see Wagneur and Sriskandarajah [19]. It can be shown that, if the cost function $f_j(C_j)$ is increasing in C_j for each order $j = 1, 2, \ldots, n$, then there exists an optimal schedule for the objective function $f_{\max}$ as well as an optimal schedule for the objective function $\sum f_j(C_j)$ in which all machines process the orders in the same sequence. Thus, if all r_j are zero it suffices to consider permutation schedules.

The total weighted completion time order scheduling problem was addressed first by Ahmadi and Bagchi [2]. With regard to its complexity, Ahmadi and Bagchi [3] and Sung and Yoon [18] independently showed that the problem $PD2\|\sum w_j C_j$ is strongly NP-hard. Ahmadi and Bagchi [3] also showed that the problem $PD\|\sum C_j$ is ordinary NP-hard. Wagneur and Sriskandarajah [19] presented a proof claiming that $PD2\|\sum C_j$ is strongly NP-hard. Unfortunately, their proof turned out to be incorrect, see Leung et al. [12]. Chen and Hall [6] showed that $PD4\|\sum C_j$ is strongly NP-Hard, Leung et al. [10] showed that $PD3\|\sum C_j$ is strongly NP-hard, and finally, Roemer [16] showed that $PD2\|\sum C_j$ is strongly NP-hard.

As for the algorithmic aspect, Leung et al. [10] introduced two priority rules for $PDm\|\sum C_j$ and compared their two rules with three other priority rules which were proposed earlier by Sung and Yoon [18] and by Wang and Cheng [20]. All but one of the five priority rules have an approximation ratio of m (the exception having an approximation ratio that is unbounded). Ahmadi et al. [4] also presented four m-approximation heuristics for $PDm\|\sum w_j C_j$. From the performance analyses of the simple heuristics, it is not clear whether they have tighter approximation ratios. Wang and Cheng [20] presented an LP-based algorithm which has an approximation ratio of $\frac{16}{3}$. Recently, we realized that Chen and Hall [6] had also obtained a 2-approximation LP-based algorithm. Their result is also among the results that we present in what follows (we obtained our results independently from Chen and Hall).

In this paper, we focus on the performance of a number of priority rules and approximation algorithms for $PDm\|\sum w_j C_j$. Occasionally, however, we extend our results to $PDm|r_j|\sum w_j C_j$. Note that a *ρ-approximation algorithm* is a polynomial-time algorithm that produces a solution with a cost that is at most ρ times the optimal cost. Usually, ρ is referred to as the *performance ratio* (or *approximation ratio*, *performance guarantee*, *worst-case ratio*, etc.) of the algorithm. The algorithms considered in this paper are of two types: priority rules (either static or dynamic) and LP-based algorithms. The priority rules are only applicable to $PDm\|\sum w_j C_j$, i.e., only when $r_j = 0$ for all j. An analysis of the priority rules shows that they are sensitive to the characteristics of the processing times of the orders. It appears that the more sophisticated LP-based algorithms are not as sensitive to the characteristics of the processing times. In order to obtain some insights into the performance of these algorithms when applied to $PDm\|\sum w_j C_j$ instances in practice, we present an extensive comparative analysis that takes into consideration solution quality, speed, memory space, and implementation complexity. The observations from our empirical analysis may facilitate the selection of an appropriate rule or algorithm in a real life situation.

This paper is organized as follows. Section 2 presents five priority rules and analyzes their performance. Section 3 studies the performance of these rules assuming additional constraints on the characteristics of the processing times of each order. Section 4 focuses on two LP-based approximation algorithms and in Section 5, we conduct an empirical analysis of the rules and algorithms. Finally, some concluding remarks are presented in Section 6.

2. Priority rules for $PDm\|\sum w_j C_j$

It is clear that for $m = 1$ the problem $PDm\|\sum w_j C_j$ can be solved by the weighted shortest processing time first (*WSPT*) rule due to Smith [17]. However, when $m \geqslant 2$, the problem $PDm\|\sum w_j C_j$ becomes NP-hard in the strong sense [18]. Thus, it is of interest to develop good heuristics.

Let Ω denote the set of orders that have not yet been scheduled. Assuming a partial schedule π, we consider five greedy ways of selecting the next order $j^* \in \Omega$ that has to be added to the partial schedule. The first two methods described below are basically static priority rules, i.e., the entire sequence can be determined at time $t = 0$ based only on

information pertaining to the orders. The third method is a two-pass rule, which schedules the orders in two passes (the schedule information obtained in the first pass provides the data necessary for doing the second pass). The fourth and fifth method are single pass dynamic priority rules. That is, the schedule can be developed in a single pass; however, it cannot be done using only information pertaining to the orders. In order to add an additional order to a partial schedule, information pertaining to the existing partial schedule has to be taken into account as well. The first two methods were first proposed by Sung and Yoon [18] for two machines, and generalized by Wang and Cheng [20] for m machines. The third method was introduced by Wang and Cheng [20]. The remaining two algorithms are new. The reason that we put the first three algorithms together with our two new ones as below is that, we would like to have analyze and compare these algorithms with ours in this paper, both theoretically and empirically. As we will show later, our *WECT* rule seems to outperform all others most of the time.

- The weighted shortest total processing time first (*WSTP*) rule schedules the orders in increasing order of $\sum_{i=1}^{m} p_{ij}/w_j$. Ties are broken arbitrarily.
- The weighted shortest maximum processing time first (*WSMP*) rule schedules the orders in increasing order of $\max_i\{p_{ij}\}/w_j$. Ties are broken arbitrarily.
- The weighted smallest maximum completion time first (*WSMC*) rule first sequences the orders on each machine, $i = 1, 2, \ldots, m$, in increasing order of p_{ij}/w_j. (Note that the sequences of the orders on the various machines may be different.) The rule then computes the completion time for order j as $C'_j = \max_{i=1}^{m}\{C_{ij}\}$. In a second pass, the rule schedules the orders in increasing order of C'_j. Ties are broken arbitrarily.
- The rule that applies *WSPT* first to the machine with the largest current load (*WSPL*); it functions as a dynamic priority rule that generates a sequence of orders one at a time, each time selecting as the next order the order $j^* \in \Omega$ such that

$$j^* = \arg\min_{j\in\Omega}\left\{\frac{p_{i^*j}}{w_j}\right\},$$

  where i^* is the machine with the largest workload under the partial schedule π (if there are more than one machine which have the largest current load, for simplicity we just choose the one with the smallest label). Ties for jobs are broken arbitrarily.
- The weighted earliest completion time first (*WECT*) rule selects as the next order j^* which satisfies

$$j^* = \arg\min_{j\in\Omega}\left\{\frac{C_j - C_k}{w_j}\right\},$$

  where C_k is the finish time of the order that was scheduled immediately before order j^*. Ties may be broken arbitrarily.

For each heuristic described above, after the next order has been chosen, a postprocessing procedure can be applied. Let $[j]$ be the order scheduled in position j of S; for convenience, we denote this order as j^*. The postprocessing procedure works as follows: interchange order j^* with order $[j-1]$ if $C_{j^*} \leqslant C_{[j-1]}$. Such an interchange generates a solution that is at least as good as the original sequence. Note that the case $C_{j^*} \leqslant C_{[j-1]}$ occurs only when $p_{i^*j^*} = 0$, where i^* refers to the machine that determines the completion time of order $[j-1]$ (i.e., machine i^* has, among all machines, the latest finishing time for order $[j-1]$). If, after the swap of the two orders, $C_{j^*} \leqslant C_{[j-2]}$, then we follow up with an additional interchange of order j^* with order $[j-2]$. We continue with this interchange until C_{j^*} is larger than the completion time of the order that immediately precedes it. Note that after each swap, the finish time of j^* either decreases or remains unchanged, while the finish time of each order that is swapped with j^* remains unchanged. This is due to the fact that order j^* has zero processing time on the machine on which the swapped order has its largest finish time. Thus, the postprocessing, if any, produces a solution that is no worse than the one prior to the postprocessing.

Note that in each heuristic, there may at times be ties. Since ties may be broken arbitrarily, each heuristic could, for the same input, lead to various different schedules with different values of objective functions.

The above heuristics were developed based on various perspectives of the problem. Both *WSTP* and *WSMP* are generalizations of the well-known *WSPT* rule, which is optimal on a single machine for minimizing $\sum w_j C_j$. *WSTP* treats the m machines as a single machine, and applies the *WSPT* rule to the total processing time, i.e., $\sum_{i=1}^{m} p_{ij}$. On the

other hand, *WSMP* takes the largest processing time, i.e., $\max_i\{p_{ij}\}$, and then applies the *WSPT* rule to this processing time. *WSMC* decomposes the problem into m single-machine problems, and then solves each single-machine problem by the *WSPT* rule. Finally, it integrates the solutions by taking the largest completion time C'_j and schedules the orders in ascending order of C'_j. *WSPL* tries to equalize the workload of the machines by dynamically assigning the order with the smallest weighted processing time to the machine with the largest current work load. Finally, *WECT* is a dynamic rule that assigns the next order with the smallest weighted completion time.

Due to the greedy nature of these algorithms, they run very fast. Indeed, through a sorting algorithm, both *WSTP* and *WSMP* can be implemented to run in $O(mn + n \lg n)$ time, and *WSMC* can be implemented to run in $O(mn \lg n)$ time. Both *WSPL* and *WECT* can be implemented in a rather straightforward manner to run in $O(mn^2)$ time.

Now let us consider the performance of the five rules. When $m = 2$, Sung and Yoon [18] showed that both *WSTP* and *WSMC* have an approximation ratio of 2. Actually, Wang and Cheng [20] obtained the following result:

Theorem 1. *WSTP, WSMP and WSMC are all m-approximation algorithms for* $PDm\|\sum w_jC_j$.

As for *WSPL*, Leung et al. [10] showed that the algorithm is unbounded even when all $w_j = 1$. However, an empirical analysis showed that it performs very well in practice when $w_j = 1$. The *WECT* algorithm is in a sense a generalization of the m-approximation *ECT* algorithm introduced in Leung et al. [10]. If all $w_j = 1$, then *WECT* reduces to *ECT*. In what follows, we show that *WECT* is also an m-approximation algorithm for $PDm\|\sum w_jC_j$.

Let $\varrho_j = \max(p_{1j}, \ldots, p_{mj})$, $j = 1, 2, \ldots, n$. Let $C_j(WECT)$ and $C_j(OPT)$ denote the completion time of order j in the WECT schedule and the optimal schedule, respectively. We assume without loss of generality that the orders are labeled in such a way that

$$\frac{\varrho_1}{w_1} \leqslant \frac{\varrho_2}{w_2} \leqslant \cdots \leqslant \frac{\varrho_n}{w_n}. \tag{1}$$

Furthermore, let $[j]$ refer to the order that appears in position j of a schedule.

First, we show the following lemma that is a key observation with regard to *WECT*:

Lemma 2. *For any schedule generated by WECT,*

$$\frac{C_{[j]}(WECT) - C_{[j-1]}(WECT)}{w_{[j]}} \leqslant \frac{\varrho_j}{w_j}, \quad j = 2, 3, \ldots, n.$$

Proof. [1] By *WECT*, the first job is scheduled in position 1. Suppose we want to schedule a job in the j^*th position which is larger than 2. It is clear that at least one job j', $2 \leqslant j' \leqslant j^*$, should be in the unscheduled job set Ω. For any position j, $1 \leqslant j \leqslant n$, it is easy to see that

$$C_{[j]}(WECT) - C_{[j-1]}(WECT) \leqslant \varrho_{[j]}. \tag{2}$$

Let C' be the completion time of job j' if it is put in position j^*. By the ordering in (2) and the *WECT* rule, we have

$$\frac{C_{[j^*]}(WECT) - C_{[j^*-1]}(WECT)}{w_{[j^*]}} \leqslant \frac{C' - C_{[j^*-1]}(WECT)}{w_{j'}} \leqslant \frac{\varrho_{j'}}{w_{j'}} \leqslant \frac{\varrho_{j^*}}{w_{j^*}}.$$

This completes the proof. □

Based on Lemma 2, we can establish the following upper bound on the objective of the schedule generated by *WECT*.

Lemma 3. *For any schedule generated by WECT,*

$$\sum_{j=1}^{n} w_{[j]}C_{[j]}(WECT) \leqslant \sum_{j=1}^{n} w_j \sum_{k=1}^{j} \varrho_k.$$

[1] This simpler proof was suggested by one of the referees.

Proof. Let $\varrho'_{[1]} = \varrho_1$, $w_{[1]} = w_1$ and let

$$\varrho'_{[j]} = C_{[j]}(WECT) - C_{[j-1]}(WECT), \quad j = 2, 3, \ldots, n.$$

Note that, from Lemma 2, we have

$$\frac{\varrho'_{[j]}}{w_{[j]}} \leqslant \frac{\varrho_j}{w_j}, \quad j = 2, 3, \ldots, n; \tag{3}$$

and

$$\varrho'_{[j]} \leqslant \varrho_{[j]}, \quad j = 2, 3, \ldots, n. \tag{4}$$

It is clear that from the *WECT* schedule each $\varrho'_{[j]}$ can be determined easily. Based on this notation, we have

$$\sum_{j=1}^{n} w_{[j]} C_{[j]}(WECT) = \sum_{j=1}^{n} w_{[j]} \sum_{k=1}^{j} \varrho'_{[k]}. \tag{5}$$

In order to prove the lemma, it suffices to show that

$$\sum_{j=1}^{n} w_{[j]} \sum_{k=1}^{j} \varrho'_{[k]} \leqslant \sum_{j=1}^{n} w_j \sum_{k=1}^{j} \varrho_k. \tag{6}$$

In order to prove inequality (6), consider an artificial instance of the classical problem $1\|\sum w_j C_j$ with jobs that have the following pairs of processing times and weights:

$$(\varrho_1, w_1), (\varrho_2, w_2), \ldots, (\varrho_n, w_n).$$

Here, each pair of ϱ_j and w_j are exactly the same as those in (1). Clearly, if we sequence the above jobs according to the classical *WSPT* rule, then the objective of the schedule is

$$\sum_{j=1}^{n} w_j \sum_{k=1}^{j} \varrho_k,$$

due to the assumption of the ordering in (1). For convenience, we refer to this very first sequence as

$$S_1 : \quad \langle 1, 2, \ldots, n \rangle.$$

We now change in S_1 the processing time of the job with the same label as that of the order scheduled in the second position of *WECT*. Note that this order is labeled as [2] in *WECT* and it corresponds to one specific job scheduled in S_1. For convenience and consistency, we refer the label of this job in S_1 as [2]. Since the *WECT* algorithm always chooses order 1 as [1], it is clear that $[2] \in \{2, 3, \ldots, n\}$. With [2], we can rewrite S_1 as

$$S_1 : \quad \langle 1, 2, \ldots, [2]-1, [2], [2]+1, \ldots, n \rangle.$$

Now, for job [2] in S_1, we reset $\varrho_{[2]}$ to be a new value $\varrho'_{[2]}$. According to (4), the new value of $\varrho_{[2]}$ is equal to or smaller than its original value, while $w_{[2]}$ remains unchanged. It follows that the objective cost of S_1 remains unchanged or decreases after setting $\varrho_{[2]} = \varrho'_{[2]}$, i.e.,

$$\sum w_j C_j(S_1) \leqslant \sum_{j=1}^{n} w_j \sum_{k=1}^{j} \varrho_k. \tag{7}$$

Note that if $[2] = 2$, the above inequality still holds and this case is simple. Thus, we need to focus only on the case $[2] \neq 2$. After letting $\varrho_{[2]} = \varrho'_{[2]}$, we obtain from (1) and (3) a new ordering of ϱ_j / w_j as follows:

$$\frac{\varrho_1}{w_1} \leqslant \frac{\varrho_{[2]}}{w_{[2]}} \leqslant \frac{\varrho_2}{w_2} \leqslant \cdots \leqslant \frac{\varrho_{[2]-1}}{w_{[2]-1}} \leqslant \frac{\varrho_{[2]+1}}{w_{[2]+1}} \leqslant \cdots \leqslant \frac{\varrho_n}{w_n}. \tag{8}$$

If we swap in S_1 the position of job [2] with that of job [2] − 1, we obtain a new sequence

$$S_2: \quad \langle 1, 2, \ldots, [2]-2, [2], [2]-1, [2]+1, \ldots, n\rangle.$$

According to (8), since

$$\frac{\varrho_{[2]}}{w_{[2]}} \leqslant \cdots \leqslant \frac{\varrho_{[2]-1}}{w_{[2]-1}},$$

it is easy to show via an adjacent interchange argument [14] that

$$\sum w_j C_j(S_2) \leqslant \sum w_j C_j(S_1). \tag{9}$$

We keep on swapping the position of job [2] with that of the job positioned immediately before it until we obtain the sequence

$$S_3: \langle 1, [2], 2, \ldots, [2]-1, [2]+1, \ldots, n\rangle,$$

and we have

$$\sum w_j C_j(S_3) \leqslant \sum w_j C_j(S_2). \tag{10}$$

After repeating the above procedure for jobs [3], [4], ..., [n] corresponding to the labels of the orders in *WECT*, we finally obtain a sequence

$$S_4: \langle 1, [2], [3], \ldots, [n-1], [n]\rangle$$

and we have

$$\sum w_j C_j(S_4) \leqslant \sum w_j C_j(S_3). \tag{11}$$

From (7),(9),(10) and (11), we have

$$\sum w_j C_j(S_4) \leqslant \sum_{j=1}^{n} w_j \sum_{k=1}^{j} \varrho_k. \tag{12}$$

Note that in S_4, for each $j = 2, 3, \ldots, n$,

$$\varrho_{[j]} = \varrho'_{[j]}.$$

It follows that

$$\sum w_j C_j(S_4) = \sum_{j=1}^{n} w_{[j]} \sum_{k=1}^{j} \varrho'_{[k]}. \tag{13}$$

From (5), (12) and (13), the result follows. □

Theorem 4. *For PDm*$\|\sum w_j C_j$,

$$\frac{\sum w_j C_j(WECT)}{\sum w_j C_j(OPT)} \leqslant m.$$

Proof. Even without the postprocessing we have from Lemma 3 that

$$\sum w_j C_j(WECT) \leqslant \sum_{j=1}^{n} w_j \sum_{k=1}^{j} \varrho_k. \tag{14}$$

For the optimal schedule we have

$$C_{[j]}(OPT) = \max_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{i[k]}\right\} \geqslant \sum_{k=1}^{j}\left(\sum_{i=1}^{m} p_{i[k]}\right)\Big/ m \geqslant \sum_{k=1}^{j} \max_{1\leqslant i\leqslant m}\{p_{i[k]}\}\Big/ m.$$

It follows that

$$\sum_{j=1}^{n} w_j C_j(OPT) \geqslant \sum_{j=1}^{n} w_{[j]} \sum_{k=1}^{j} \max_{1\leqslant i\leqslant m}\{p_{i[k]}\}\Big/ m \geqslant \sum_{j=1}^{n} w_j \sum_{k=1}^{j} \varrho_k \Big/ m. \tag{15}$$

Note that the last "$\geqslant$" in (15) is due to Smith's *WSPT* rule. It follows from (14) and (15) that the performance ratio of *WECT* without postprocessing is at most m. Since the postprocessing procedure could lead to a better solution, it can only help to improve the performance of *WECT*. □

3. Analysis of priority rules when processing times are restricted

There may be situations in which customers may place orders that are well-balanced in such a way that the processing times of different product types are subject to constraints that ensure some form of regularity. It would not be surprising that the priority rules would perform better under such constraints. Sung and Yoon [18] showed that for $m = 2$ the performance ratio of *WSTP* can be reduced to $\frac{3}{2}$ when the processing times satisfy the additional constraint set $(p_{1j} + p_{2j})/2 \geqslant |p_{1j} - p_{2j}|$ for each $j = 1, 2, \ldots, n$. In what follows we obtain tighter bounds for the priority rules when the processing times are subject to certain additional constraints. In the remaining part of this section we use the following notation:

- Let $\delta_j = \max_{1\leqslant i\leqslant m}\{p_{ij}\} - \min_{1\leqslant i\leqslant m}\{p_{ij}\}$ for each $j = 1, 2, \ldots, n$.
- Let $\min^{(k)}_{1\leqslant i\leqslant m}\{a_i\}$ denote the kth smallest item among $\{a_1, a_2, \ldots, a_m\}$.
- Let $[j]$ denote the order scheduled in position j of a schedule.

3.1. Additional constraint set $\sum_{i=1}^{m} p_{ij}/m \geqslant \delta_j$

With the additional constraint $\sum_{i=1}^{m} p_{ij}/m \geqslant \delta_j$ for each order $j = 1, 2, \ldots, n$, we have the following result for *WSTP*.

Theorem 5. *If*

$$\sum_{i=1}^{m} p_{ij}/m \geqslant \delta_j$$

for $j = 1, 2, \ldots, n$, *then*

$$\frac{\sum w_j C_j(WSTP)}{\sum w_j C_j(OPT)} \leqslant 2 - \frac{1}{m}.$$

Proof. Without loss of generality, we may assume that

$$\frac{\sigma_1}{w_1} \leqslant \frac{\sigma_2}{w_2} \leqslant \cdots \leqslant \frac{\sigma_n}{w_n}, \tag{16}$$

where $\sigma_j = \sum_{i=1}^{m} p_{ij}$, $j = 1, 2, \ldots, n$. According to such notation, we have

$$\delta_j \leqslant \sigma_j/m. \tag{17}$$

It is clear that

$$\sum w_j C_j(OPT) \geqslant \sum_{j=1}^{n} w_{[j]} \sum_{k=1}^{j} \sigma_{[k]} \Big/ m \geqslant \sum_{j=1}^{n} w_j \sum_{k=1}^{j} \sigma_k \Big/ m. \tag{18}$$

Again, the last "$\geqslant$" in (18) is due to Smith's *WSPT* rule.

Now consider the schedule generated by *WSTP*. Clearly, in this schedule, the order scheduled in position j is order j itself (if there are ties, we can always relabel the orders such that they satisfy the ordering in (16)). Suppose the latest finishing time of order j takes place on machine i^*, and the earliest finishing time of order j is on machine i'. Now

$$\max_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{ik}\right\} - \min_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{ik}\right\} = \sum_{k=1}^{j} p_{i^*k} - \sum_{k=1}^{j} p_{i'k} = \sum_{k=1}^{j}(p_{i^*k} - p_{i'k}) \leqslant \sum_{k=1}^{j} \delta_k.$$

Therefore, we have for each $i = 1, 2, \ldots, m$ the following relationship:

$$\max_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{ik}\right\} - \overset{(i)}{\min_{1\leqslant l\leqslant m}}\left\{\sum_{k=1}^{j} p_{lk}\right\} \leqslant \max_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{ik}\right\} - \min_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{ik}\right\} = \sum_{k=1}^{j} \delta_k.$$

Or, equivalently,

$$\overset{(i)}{\min_{1\leqslant l\leqslant m}}\left\{\sum_{k=1}^{j} p_{lk}\right\} \geqslant \max_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{ik}\right\} - \sum_{k=1}^{j} \delta_k. \tag{19}$$

Since

$$\sum_{k=1}^{j} \sigma_k = \sum_{i=1}^{m} \overset{(i)}{\min_{1\leqslant l\leqslant m}}\left\{\sum_{k=1}^{j} p_{lk}\right\} = \max_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{ik}\right\} + \sum_{i=1}^{m-1} \overset{(i)}{\min_{1\leqslant l\leqslant m}}\left\{\sum_{k=1}^{j} p_{lk}\right\},$$

we have

$$\begin{aligned}\max_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{ik}\right\} &= \sum_{k=1}^{j} \sigma_k - \sum_{i=1}^{m-1} \overset{(i)}{\min_{1\leqslant l\leqslant m}}\left\{\sum_{k=1}^{j} p_{lk}\right\} \\ &\leqslant \sum_{k=1}^{j} \sigma_k - (m-1)\left(\max_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{ik}\right\} - \sum_{k=1}^{j} \delta_k\right).\end{aligned}$$

The "$\leqslant$" is due to (19). We rewrite the above inequality as

$$m\left(\max_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{ik}\right\}\right) \leqslant \sum_{k=1}^{j} \sigma_k + (m-1)\sum_{k=1}^{j} \delta_k.$$

It follows that,

$$C_j(WSTP) = \max_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j} p_{ik}\right\} \leqslant \frac{\sum_{k=1}^{j}\sigma_k + (m-1)\sum_{k=1}^{j}\delta_k}{m}.$$

By (17) we have

$$C_j(WSTP) \leqslant \sum_{k=1}^{j} \frac{(2m-1)\sigma_k}{m^2}. \tag{20}$$

From (18) and (20), the result follows. □

It is clear that the bound is monotonically increasing with m. When $m = 1$ it is 1 and when $m = 2$ it is $\frac{3}{2}$. When m goes to ∞, the bound goes to 2. We have the following result for *WSMP*.

Theorem 6. *If*

$$\sum_{i=1}^{m} p_{ij}/m \geqslant \delta_j$$

for $j = 1, 2, \ldots, n$, *then*

$$\frac{\sum w_j C_j(WSMP)}{\sum w_j C_j(OPT)} \leqslant \frac{1}{1 + \mathscr{H}(m) - \mathscr{H}(2m-1)},$$

where $\mathscr{H}(k) \equiv 1 + \frac{1}{2} + \cdots + 1/k$ *is the harmonic series.*

Proof. Without loss of generality we may assume that

$$\frac{\varrho_1}{w_1} \leqslant \frac{\varrho_2}{w_2} \leqslant \cdots \leqslant \frac{\varrho_n}{w_n}, \tag{21}$$

where $\varrho_j = \max_{1 \leqslant i \leqslant m}\{p_{ij}\}$, $j = 1, 2, \ldots, n$.

In the schedule generated by *WSMP*, the order scheduled in position j is order j itself (again, if there are ties, we can always relabel the orders to guarantee the ordering in (21)). It is easy to see that

$$\sum w_j C_j(WSMP) \leqslant \sum_{j=1}^{n} w_j \sum_{k=1}^{j} \varrho_k. \tag{22}$$

Now let us consider the order scheduled in the jth position of an optimal solution. Its completion time is

$$C_{[j]}(OPT) = \max_{1 \leqslant i \leqslant m} \left\{ \sum_{k=1}^{j} p_{i[k]} \right\} \geqslant \frac{\sum_{k=1}^{j} \sum_{i=1}^{m} p_{i[k]}}{m}. \tag{23}$$

For the order scheduled in the kth position of the optimal schedule we have

$$\frac{(m-1)\max_{1 \leqslant i \leqslant m}\{p_{i[k]}\} + \min_{1 \leqslant i \leqslant m}\{p_{i[k]}\}}{m} \geqslant \frac{\sum_{i=1}^{m} p_{i[k]}}{m} \geqslant \max_{1 \leqslant i \leqslant m}\{p_{i[k]}\} - \min_{1 \leqslant i \leqslant m}\{p_{i[k]}\}.$$

It follows that

$$\min_{1 \leqslant i \leqslant m}\{p_{i[k]}\} \geqslant \frac{\varrho_{[k]}}{m+1}.$$

More generally, for each $i = 1, 2, \ldots, m$ we have

$$\begin{aligned} \frac{(m-i)\max_{1 \leqslant i \leqslant m}\{p_{i[k]}\} + i \cdot \min^{(i)}_{1 \leqslant l \leqslant m}\{p_{l[k]}\}}{m} &\geqslant \frac{\sum_{i=1}^{m} p_{i[k]}}{m} \geqslant \max_{1 \leqslant i \leqslant m}\{p_{i[k]}\} - \min_{1 \leqslant i \leqslant m}\{p_{i[k]}\} \\ &\geqslant \max_{1 \leqslant i \leqslant m}\{p_{i[k]}\} - \min^{(i)}_{1 \leqslant i \leqslant m}\{p_{i[k]}\}. \end{aligned}$$

Or, equivalently,

$$\min^{(i)}_{1 \leqslant i \leqslant m}\{p_{i[k]}\} \geqslant \frac{i}{m+i} \cdot \varrho_{[k]}.$$

Therefore, by (23) we have

$$C_{[j]}(OPT) \geqslant \frac{\sum_{k=1}^{j}(\varrho_{[k]} + \sum_{i=1}^{m-1}(i/(m+i)) \cdot \varrho_{[k]})}{m} = \frac{\sum_{k=1}^{j}(1 + \sum_{i=1}^{m-1}(1 - (m/(m+i))) \cdot \varrho_{[k]}}{m}$$
$$= \left(1 - \sum_{i=1}^{m-1} \frac{1}{m+i}\right) \cdot \sum_{k=1}^{j} \varrho_{[k]} = \left(1 - \left(\sum_{i=1}^{2m-1} \frac{1}{i} - \sum_{i=1}^{m} \frac{1}{i}\right)\right) \cdot \sum_{k=1}^{j} \varrho_{[k]}$$
$$= (1 + \mathscr{H}(m) - \mathscr{H}(2m-1)) \cdot \sum_{k=1}^{j} \varrho_{[k]}.$$

Thus,

$$\sum w_j C_j(OPT) = \sum_{j=1}^{n} w_{[j]} C_{[j]}(OPT) \geqslant (1 + \mathscr{H}(m) - \mathscr{H}(2m-1)) \sum_{j=1}^{n} w_{[j]} \sum_{k=1}^{j} \varrho_{[k]}$$
$$\geqslant (1 + \mathscr{H}(m) - \mathscr{H}(2m-1)) \sum_{j=1}^{n} w_j \sum_{k=1}^{j} \varrho_k. \tag{24}$$

From (22) and (24), the result follows. □

Note that the bound increases with m. To see this, we consider the bound for m machines and for $m+1$ machines, respectively. For convenience, we let

$$\Gamma = (1 + \mathscr{H}(m+1) - \mathscr{H}(2m+1))(1 + \mathscr{H}(m) - \mathscr{H}(2m-1)).$$

The gap between the two bounds for the respective number of machines is

$$\Delta = \frac{1}{1 + \mathscr{H}(m+1) - \mathscr{H}(2m+1)} - \frac{1}{1 + \mathscr{H}(m) - \mathscr{H}(2m-1)}$$
$$= \left(\frac{1}{2m} + \frac{1}{2m+1} - \frac{1}{m+1}\right) \Big/ \Gamma = \frac{3m+1}{2m(m+1)(2m+1) \cdot \Gamma} > 0.$$

Clearly, when $m = 1$, the bound is 1 and when $m = 2$ the bound is $\frac{3}{2}$. Finally,

$$\lim_{m \to \infty} \frac{1}{1 + \mathscr{H}(m) - \mathscr{H}(2m-1)} = \frac{1}{1 - \ln 2} \approx 3.259.$$

We now have the following result for *WECT*.

Theorem 7. *If*

$$\sum_{i=1}^{m} p_{ij}/m \geqslant \delta_j$$

for each order $j = 1, 2, \ldots, n$, *then*

$$\frac{\sum w_j C_j(WECT)}{\sum w_j C_j(OPT)} \leqslant \frac{1}{1 + \mathscr{H}(m) - \mathscr{H}(2m-1)}.$$

Proof. The result follows immediately from Lemma 3 and (24) in Theorem 6. □

3.2. Additional constraint set $\max_{1 \leqslant i \leqslant m}\{p_{ij}\} \leqslant a \cdot \min_{1 \leqslant i \leqslant m}\{p_{ij}\}$

Now consider the processing times subject to the following constraints:

$$\max_{1 \leqslant i \leqslant m}\{p_{ij}\} \leqslant a \cdot \min_{1 \leqslant i \leqslant m}\{p_{ij}\}, \quad j = 1, 2, \ldots, n,$$

where $a \geqslant 1$. With these additional constraints, we obtain the following result for *WSTP*.

Theorem 8. *For some* $a \geqslant 1$, *if*

$$\max_{1 \leqslant i \leqslant m} \{p_{ij}\} \leqslant a \cdot \min_{1 \leqslant i \leqslant m} \{p_{ij}\}$$

for $j = 1, 2, \ldots, n$, *then*

$$\frac{\sum w_j C_j(WSTP)}{\sum w_j C_j(OPT)} \leqslant a - \frac{a^2 - a}{a + m - 1}.$$

Proof. Without loss of generality, we assume that

$$\frac{\sigma_1}{w_1} \leqslant \frac{\sigma_2}{w_2} \leqslant \cdots \leqslant \frac{\sigma_n}{w_n},$$

where $\sigma_j = \sum_{i=1}^{m} p_{ij}$, $j = 1, 2, \ldots, n$. First of all, it is clear that

$$\sum w_j C_j(OPT) \geqslant \sum_{j=1}^{n} w_{[j]} \sum_{k=1}^{j} \sigma_{[k]} \Big/ m \geqslant \sum_{j=1}^{n} w_j \sum_{k=1}^{j} \sigma_k \Big/ m. \tag{25}$$

Furthermore, it is easy to check that

$$\sum w_j C_j(WSTP) \leqslant \sum_{j=1}^{n} w_j \cdot \sum_{k=1}^{j} \max_i \{p_{ik}\}. \tag{26}$$

From the assumption that

$$\min_{1 \leqslant i \leqslant m} \{p_{ij}\} \geqslant \frac{1}{a} \max_{1 \leqslant i \leqslant m} \{p_{ij}\},$$

we have

$$\max_{1 \leqslant i \leqslant m} \{p_{ij}\} + \frac{(m-1)\max_{1 \leqslant i \leqslant m}\{p_{ij}\}}{a} \leqslant \max_{1 \leqslant i \leqslant m} \{p_{ij}\} + (m-1) \min_{1 \leqslant i \leqslant m} \{p_{ij}\} \leqslant \sum_{i=1}^{m} p_{ij} = \sigma_j.$$

Or, equivalently,

$$\max_{1 \leqslant i \leqslant m} \{p_{ij}\} \leqslant \frac{a \cdot \sigma_j}{a + m - 1}.$$

Thus, from (26) we have

$$\sum w_j C_j(WSTP) \leqslant \frac{a}{a + m - 1} \sum_{j=1}^{n} w_j \sum_{k=1}^{j} \sigma_k. \tag{27}$$

The result follows from (25) and (27). □

It is easy to see that the above bound increases monotonically in m. When $m = 1$ it is 1, and when $m = 2$ it is $2a/(a + 1)$. When m goes to ∞, the bound goes to a.

We now have for *WSMP* the following result.

Theorem 9. *If*

$$\max_{1\leqslant i\leqslant m}\{p_{ij}\}\leqslant a\cdot\min_{1\leqslant i\leqslant m}\{p_{ij}\}$$

for $j=1,2,\ldots,n$, *then*

$$\frac{\sum w_jC_j(WSMP)}{\sum w_jC_j(OPT)}\leqslant a-\frac{a^2-a}{a+m-1}.$$

Proof. Again, we assume without loss of generality that

$$\frac{\varrho_1}{w_1}\leqslant\frac{\varrho_2}{w_2}\leqslant\cdots\leqslant\frac{\varrho_n}{w_n},$$

where $\varrho_j=\max_{1\leqslant i\leqslant m}\{p_{ij}\}$, $j=1,2,\ldots,n$.

It is clear that

$$\sum w_jC_j(WSMP)\leqslant\sum_{j=1}^{n}w_j\sum_{k=1}^{j}\varrho_k. \tag{28}$$

Now consider the order scheduled in the jth position of an optimal solution. Its completion time is

$$C_{[j]}(OPT)=\max_{1\leqslant i\leqslant m}\left\{\sum_{k=1}^{j}p_{i[k]}\right\}\geqslant\frac{\sum_{k=1}^{j}\sum_{i=1}^{m}p_{i[k]}}{m}. \tag{29}$$

For each $i=1,2,\ldots,m$ we have

$$\overset{(i)}{\min_{1\leqslant l\leqslant m}}\{p_{l[k]}\}\geqslant\min_{1\leqslant l\leqslant m}\{p_{l[k]}\}\geqslant\frac{\varrho_{[k]}}{a}. \tag{30}$$

The last "$\geqslant$" in (30) is due to the assumption. Therefore, from (29) we have

$$\begin{aligned}C_{[j]}(OPT)&\geqslant\frac{\sum_{k=1}^{j}\sum_{i=1}^{m}p_{i[k]}}{m}=\frac{\sum_{k=1}^{j}(\max_{1\leqslant i\leqslant m}\{p_{i[k]}\}+\sum_{i=1}^{m-1}\overset{(i)}{\min_{1\leqslant l\leqslant m}}\{p_{l[k]}\})}{m}\\&\geqslant\frac{\sum_{k=1}^{j}(\varrho_{[k]}+\sum_{i=1}^{m-1}\varrho_{[k]}/a)}{m}=\frac{a+m-1}{a\cdot m}\cdot\sum_{k=1}^{j}\varrho_{[k]}.\end{aligned}$$

It follows that

$$\sum w_jC_j(OPT)\geqslant\frac{a+m-1}{a\cdot m}\sum_{j=1}^{n}w_{[j]}\sum_{k=1}^{j}\varrho_{[k]}\geqslant\frac{a+m-1}{a\cdot m}\sum_{j=1}^{n}w_j\sum_{k=1}^{j}\varrho_k. \tag{31}$$

The result follows from (28) and (31). □

Theorem 10. *If*

$$\max_{1\leqslant i\leqslant m}\{p_{ij}\}\leqslant a\cdot\min_{1\leqslant i\leqslant m}\{p_{ij}\}$$

for each order $j=1,2,\ldots,n$, then

$$\frac{\sum w_jC_j(WECT)}{\sum w_jC_j(OPT)}\leqslant a-\frac{a^2-a}{a+m-1}.$$

Proof. The result follows immediately from Lemma 3 and (31) in Theorem 9. □

For the same performance bound in Theorems 8–10, if $a = 3$, then the performance bound becomes $3 - 6/(m+2)$. Note that when $a = 3$, these constraints are equivalent to

$$\frac{\max_{1\leqslant i\leqslant m}\{p_{ij}\} + \min_{1\leqslant i\leqslant m}\{p_{ij}\}}{2} \geqslant \max_{1\leqslant i\leqslant m}\{p_{ij}\} - \min_{1\leqslant i\leqslant m}\{p_{ij}\}, \quad j = 1, 2, \ldots, n.$$

This imposes that the average of the maximum and minimum processing times of an order is always no less than their difference, so that the processing times of the orders are "well-balanced".

4. LP-based approximation algorithms

In this section we allow orders to have different release dates. Preemptions are not allowed. However, unforced idleness of the machines is allowed. We present two approximation algorithms based on two different LP relaxations. We use the following notation: let $\widetilde{C}_1, \widetilde{C}_2, \ldots, \widetilde{C}_n$ denote the completion times in the schedule generated by an LP-based algorithm and let $C_1^*, C_2^*, \ldots, C_n^*$ denote the completion times in an optimal schedule.

4.1. Approximation algorithm based on completion times

Hall et al. [9] presented a 3-approximation LP-based algorithm for $1|r_j|\sum w_jC_j$. The algorithm was also considered by Chekuri and Khanna [5]. In this section, we shall extend this algorithm to solve $PDm|r_j|\sum w_jC_j$. Thus, $PDm\|\sum w_jC_j$ can also be solved as a special case.

Let $\mathscr{O} = \{1, 2, \ldots, n\}$ denote the set of all orders. For any subset $\mathscr{S} \subseteq \mathscr{O}$, let

$$\sigma_i(\mathscr{S}) = \sum_{j\in\mathscr{S}} p_{ij}, \quad \sigma_i^2(\mathscr{S}) = \sum_{j\in\mathscr{S}} p_{ij}^2, \quad i = 1, 2, \ldots, m.$$

The $PDm|r_j|\sum w_jC_j$ problem can be relaxed by the following linear program which we refer to in what follows as LP_1:

$$\text{minimize} \quad \sum_{j=1}^{n} w_jC_j$$

subject to

$$C_j \geqslant r_j + p_{ij}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n; \tag{32}$$

$$C_j \geqslant C_{ij}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n; \tag{33}$$

$$\sum_{j\in\mathscr{S}} p_{ij}C_{ij} \geqslant \frac{\sigma_i^2(\mathscr{S}) + (\sigma_i(\mathscr{S}))^2}{2}, \quad i = 1, \ldots, m, \quad \text{for each } \mathscr{S} \subseteq \mathscr{O}. \tag{34}$$

Constraint sets (32) and (33) are trivial. However, constraint set (34) needs some justification. Assume that $\mathscr{S} = \{1, 2, \ldots, |\mathscr{S}|\}$. It follows that for $j \in \mathscr{S}$,

$$C_{ij} \geqslant \sum_{k\leqslant j} p_{ik}, \quad i = 1, \ldots, m.$$

The inequality is due to the fact that there may be some idle time in the schedule because of the release dates. Thus,

$$p_{ij}C_{ij} \geqslant p_{ij}\sum_{k\leqslant j} p_{ik}.$$

Summing $p_{ij}C_{ij}$ over all $j \in \mathscr{S}$ and simple algebra results in (34).

It is clear that (34) generates an exponential number of constraints. For the one-machine case, Queyranne [15] has shown that such constraints can be separated polynomially so that the above linear program can be solved in polynomial time by a variant of ellipsoid method [8]. This is the key observation that the above linear program can be used as a relaxation for approximation algorithms.

In order to state an important lemma regarding the linear programming formulation, assume that $\overline{C}_1, \overline{C}_2, \ldots, \overline{C}_n$ is a solution to LP_1 and assume without loss of generality that $\overline{C}_1 \leqslant \overline{C}_2 \leqslant \cdots \leqslant \overline{C}_n$.

Lemma 11. *For each order* $j = 1, 2, \ldots, n$, *the following inequality holds*:

$$\overline{C}_j \geqslant \frac{1}{2} \max_i \left\{ \sum_{k=1}^{j} p_{ik} \right\}. \tag{35}$$

Proof. Let $\mathscr{S} = \{1, 2, \ldots, j\}$. Constraints (33) and (34) imply that

$$\max_i \left\{ \sum_{k=1}^{j} p_{ik} \cdot \overline{C}_k \right\} \geqslant \max_i \left\{ \frac{\sigma_i^2(\mathscr{S}) + (\sigma_i(\mathscr{S}))^2}{2} \right\} \geqslant \max_i \left\{ \frac{(\sigma_i(\mathscr{S}))^2}{2} \right\}. \tag{36}$$

Since $\overline{C}_k \leqslant \overline{C}_j$, for each $k = 1, 2, \ldots, j$ we have

$$\overline{C}_j \cdot \max_i \{\sigma_i(S)\} = \max_i \left\{ \sum_{k=1}^{j} p_{ik} \overline{C}_j \right\} \geqslant \max_i \left\{ \sum_{k=1}^{j} p_{ik} \overline{C}_k \right\} \geqslant \max_i \left\{ \frac{(\sigma_i(\mathscr{S}))^2}{2} \right\},$$

due to (36). Equivalently,

$$\overline{C}_j \geqslant \max_i \left\{ \frac{\sigma_i(S)}{2} \right\} = \frac{1}{2} \max_i \left\{ \sum_{k=1}^{j} p_{ik} \right\}.$$

The result follows. □

Now consider the following algorithm:

An LP-based algorithm using completion times (H_{LP_1})

Step 1: Solve LP_1 by the ellipsoid method; let the optimal solution be $\overline{C}_1, \overline{C}_2, \ldots, \overline{C}_n$.

Step 2: Schedule the orders in nondecreasing order of $\overline{C}_j$. Ties are broken arbitrarily. Insert an idle time when r_j is greater than the completion time of the $(j-1)$th order.

Assume without loss of generality that $\overline{C}_1 \leqslant \overline{C}_2 \leqslant \cdots \leqslant \overline{C}_n$. It is clear that

$$\sum_{j=1}^{n} w_j C_j^* \geqslant \sum_{j=1}^{n} w_j \overline{C}_j. \tag{37}$$

In what follows, we shall analyze the performance guarantee of this algorithm for $PDm \| \sum w_j C_j$ and $PDm | r_j | \sum w_j C_j$.

Theorem 12. H_{LP_1} *is a* 2*-approximation algorithm for* $PDm \| \sum w_j C_j$.

Proof. Since $r_j = 0$ for all $j = 1, 2, \ldots, n$, there is no idle time in the schedule generated by H_{LP1}. For $\mathscr{S} = \{1, 2, \ldots, j\}$, we have

$$\widetilde{C}_j = \max_i \left\{ \sum_{k=1}^{j} p_{ik} \right\} \leqslant 2\overline{C}_j,$$

due to (35). Thus,

$$\sum_{j}^{n} w_j \widetilde{C}_j \leqslant 2 \sum_{j}^{n} w_j \overline{C}_j. \tag{38}$$

By (37) and (38), we have

$$\frac{\sum_{j=1}^{n} w_j \widetilde{C}_j}{\sum_{j=1}^{n} w_j C_j^*} \leqslant 2.$$

The result follows. □

Theorem 13. *H_{LP_1} is a 3-approximation algorithm for $PDm|r_j|\sum w_j C_j$.*

Proof. Let $\mathscr{S} = \{1, 2, \ldots, j\}$; let $r_{\max}(\mathscr{S}) = \max_{j\in\mathscr{S}}\{r_j\}$. Clearly, there is no idle time between $r_{\max}(\mathscr{S})$ and $\widetilde{C}_j$. It is easy to see that

$$\widetilde{C}_j \leqslant r_{\max}(\mathscr{S}) + \max_i\{\sigma_i(\mathscr{S})\}.$$

By (32) and the assumption that $\overline{C}_1 \leqslant \overline{C}_2 \leqslant \cdots \leqslant \overline{C}_n$, we have that $r_{\max}(\mathscr{S}) \leqslant \overline{C}_j$. Thus,

$$\widetilde{C}_j \leqslant \overline{C}_j + \max_i\{\sigma_i(\mathscr{S})\} \leqslant 3\overline{C}_j, \tag{39}$$

due to Lemma 11. Therefore, by (37) and (39),

$$\frac{\sum_{j=1}^{n} w_j \widetilde{C}_j}{\sum_{j=1}^{n} w_j C_j^*} \leqslant 3.$$

The result follows. □

4.2. Approximation algorithm based on time intervals

Inspired by the time interval indexed linear programming formulation for $R|r_j|\sum w_j C_j$ by Hall et al. [9], Wang and Cheng [20] presented a $\frac{16}{3}$-approximation algorithm for $PDm\|\sum w_j C_j$. In what follows, we present an extension of Wang and Cheng's algorithm for $PDm|r_j|\sum w_j C_j$.

Given $\lambda > 1$, we divide the time horizon of potential completion times into the following intervals:

$$[1, 1], (1, \lambda], (\lambda, \lambda^2], \ldots, (\lambda^{L-1}, \lambda^L],$$

where L is the smallest integer such that

$$\lambda^L \geqslant \max_{1\leqslant j\leqslant n}\{r_j\} + \max_{1\leqslant i\leqslant m}\left\{\sum_{j=1}^{n} p_{ij}\right\}.$$

For convenience, let

$$t_0 = 1 \quad \text{and} \quad t_l = \lambda^{l-1}, \quad l = 1, \ldots, L.$$

Thus, the lth interval runs from t_{l-1} to t_l, $l = 1, 2, \ldots, L$. Let the decision variable x_{jl} be

$$x_{jl} = \begin{cases} 1 & \text{if order } j \text{ is scheduled to complete within the interval } (t_{l-1}, t_l]; \\ 0 & \text{otherwise.} \end{cases}$$

Consider the following linear programming relaxation which we refer to as LP_2:

$$\text{minimize} \quad \sum_{j=1}^{n} w_j \sum_{l=1}^{L} t_{l-1} x_{jl}$$

$$\text{subject to} \quad \sum_{l=1}^{L} x_{jl} = 1, \quad j = 1, \ldots, n; \tag{40}$$

$$\sum_{k=1}^{l} \sum_{j=1}^{n} p_{ij} x_{jk} = \sum_{j=1}^{n} p_{ij} \sum_{k=1}^{l} x_{jk} \leqslant t_l, \quad i = 1, \ldots, m, \ \ l = 1, \ldots, L; \tag{41}$$

$$x_{jl} = 0 \quad \text{if } t_l < r_j + p_{ij}, \ \ j = 1, \ldots, n, \ \ l = 1, \ldots, L; \tag{42}$$

$$x_{jl} \geqslant 0, \quad j = 1, \ldots, n, \ \ l = 1, \ldots, L. \tag{43}$$

Now consider the following algorithm:

An LP-based algorithm using time intervals (H_{LP_2})

Step 1: Given λ, solve LP_2 and let the optimal solution be $\bar{x}_{jl}$, $j = 1, \ldots, n, l = 1, \ldots, L$.

Step 2: Let $\overline{C}_j = \sum_{l=1}^{L} t_{l-1} \bar{x}_{jl}, \ \ j = 1, \ldots, n$.

Step 3: Schedule the jobs in nondecreasing order of $\overline{C}_j$. Ties are broken arbitrarily. Insert idle time when r_j is greater than the completion time of the $(j-1)$th order.

Again, assume without loss of generality that $\overline{C}_1 \leqslant \overline{C}_2 \leqslant \cdots \leqslant \overline{C}_n$. In what follows, we shall analyze the performance guarantee of the above algorithm for $PDm|r_j| \sum w_j C_j$.

Lemma 14. *The optimal value of LP_2 is a lower bound for the minimum cost of $PDm|r_j| \sum w_j C_j$. That is*

$$\sum_{j=1}^{n} w_j C_j^* \geqslant \sum_{j=1}^{n} w_j \overline{C}_j. \tag{44}$$

Proof. Consider an optimal schedule π^* for $PDm|r_j| \sum w_j C_j$. We construct a solution π to LP_2 by setting $x_{jl} = 1$ if order j completes within the lth interval. Clearly, π is feasible to LP_2, that is, constraints (40)–(43) are all satisfied. Schedule π can never be better than the optimal solution to LP_2. On the other hand, the objective cost of π^* is larger than that of π, since the completion time of order j is at least t_{l-1}. It follows that the objective cost of π^* is no better than the optimal cost of LP_2. □

Theorem 15 (*Wang and Cheng, [20]*). *Given $\lambda = 2$, H_{LP_2} is a $\frac{16}{3}$-approximation algorithm for $PDm\| \sum w_j C_j$.*

Theorem 16. *Given $\lambda = 2$, H_{LP_2} is a $\frac{19}{3}$-approximation algorithm for $PDm|r_j| \sum w_j C_j$.*

proof. Following the same argument as the one in Wang and Cheng [20], it can be shown that

$$\max_i \left\{ \sum_{k=1}^{j} p_{ik} \right\} \leqslant \frac{16}{3} \overline{C}_j l; \tag{45}$$

$$\widetilde{C}_j \leqslant \max_{1 \leqslant k \leqslant j} \{r_k\} + \max_i \left\{ \sum_{k=1}^{j} p_{ik} \right\} \leqslant \overline{C}_j + \frac{16}{3} \overline{C}_j \leqslant \frac{19}{3} \overline{C}_j. \tag{46}$$

Therefore,

$$\sum_{j=1}^{n} w_j \widetilde{C}_j \leqslant \frac{19}{3} \sum_{j=1}^{n} w_j \overline{C}_j. \tag{47}$$

Thus, by (44) and (47), the result follows. □

For LP_2, it would be of interest to investigate if a smaller λ leads to a better performance. In the next section, we present an empirical analysis.

5. Empirical analysis of the algorithms

Since the priority rules have only been formulated for $PDm\|\sum w_jC_j$, we focus our experiments on the problem with all release dates equal to zero.

5.1. Generation of problem instances

For each problem size with $n = 20, 50, 100, 200$ orders and $m = 2, 5, 10, 20$ machines, 30 instances are randomly generated using a factor called *order diversity*. The order diversity k is used to characterize the number of product types each order requires. The following three cases of order diversity are considered:

$k = 2$: In problem instances 1–6 each order requests two different product types. We denote this group as G_1.

$k = m$: In problem instances 7–24 each order requests the maximum number of different product types, namely m, i.e., the number of machines. However, these 18 instances are grouped into the following three subgroups:
For instances 7–12 in group G_2, the processing times of each order have no additional constraints.

For instances 13–18 in group G_3, the processing times of each order j are subject to the constraint

$$\max_{1\leqslant i\leqslant m}\{p_{ij}\}\leqslant 3\min_{1\leqslant i\leqslant m}\{p_{ij}\}.$$

For instances 19–24 in group G_4, the processing times of each order j are subject to the constraints

$$\sum_{i=1}^{m} p_{ij}\Big/ m\geqslant \max_i\{p_{ij}\}-\min_i\{p_{ij}\}.$$

$k = r$: In problem instances 25–30 in group G_5 each order requests a random number (r) of different product types; r is randomly generated from the uniform distribution $[1, m]$.

When the number of product types, l, for each order j is determined, l machines are chosen randomly. For each machine i that is selected, an integer processing time p_{ij} is generated from the uniform distribution $[1, 100]$. Note that for instances 13–24, the processing times of each order are generated in such a way that they satisfy the additional requirements. In addition to the generation of processing times, for each order j, a weight is randomly generated from the uniform distribution $[1, 10]$. In total, $4 \times 4 \times 30 = 480$ instances are generated.

5.2. Experimental results and analysis

The algorithms are implemented in C++. We used the GLPK 4.8 [13] callable library to solve the linear programs in the H_{LP_2} algorithm. The running environment is based on the Windows 2000 operating system; the PC used was a notebook computer (Pentium III 900 MHz plus 384 MB RAM). It should be noted that the time-interval LP-based algorithm needs a significant amount of virtual memory. For example, for a problem instance with $n = 200$ and $m = 20$, when we let $\lambda = 2^{1/4}$, the total memory usage for the time-interval LP-based algorithm could reach 1 GB. Because of this we set the total file paging size for the hard disk equal to 1152 MB.

In what follows, we study the performance of the algorithms in terms of two aspects: the frequencies at which they are the best, and comparisons of their average costs and average running times. To reduce the table size, we define the following table fields to represent the nine heuristics:

H_1: *WSTP*, H_2: *WSMP*, H_3: *WSMC*,

H_4: *WSPL*, H_5: *WECT*, LP_1: H_{LP_1},

LP_2': H_{LP_2} with $\lambda = 2^{1/4}$, LP_2'': H_{LP_2} with $\lambda = 2^{1/2}$, LP_2''': H_{LP_2} with $\lambda = 2$.

Table 1
The frequency that each algorithm performs the best for instances of group G_1

n	m	H_1	H_2	H_3	H_4	H_5	H_6	LP_1	LP'_2	LP''_2	LP'''_2
20	2	0	0	0	0	2	2	2	2	0	0
	5	0	0	0	0	1	1	3	1	1	0
	10	0	0	0	0	2	2	2	1	1	0
	20	0	0	0	0	2	2	3	1	0	0
50	2	0	0	0	0	1	1	3	2	0	0
	5	0	0	0	0	2	2	1	2	1	0
	10	0	0	0	0	1	1	2	2	1	0
	20	0	0	0	0	2	2	2	1	0	1
100	2	0	0	0	0	0	0	0	6	0	0
	5	0	0	0	0	0	0	1	5	0	0
	10	0	0	0	0	1	1	0	5	0	0
	20	0	0	0	0	0	0	1	5	0	0
200	2	0	0	0	0	0	0	0	6	0	0
	5	0	0	0	0	0	0	0	6	0	0
	10	0	0	0	0	0	0	0	6	0	0
	20	0	0	0	0	0	0	0	6	0	0

Table 2
The frequency that each algorithm performs the best for instances of group G_2

n	m	H_1	H_2	H_3	H_4	H_5	H_6	LP_1	LP'_2	LP''_2	LP'''_2
20	2	0	0	0	0	1	1	3	2	0	0
	5	0	0	0	0	2	2	1	3	0	0
	10	0	0	0	0	3	3	2	1	0	0
	20	0	0	0	0	2	2	2	1	0	1
50	2	0	0	0	0	1	1	2	3	0	0
	5	0	0	0	0	2	2	1	2	1	0
	10	0	0	0	0	2	2	1	3	0	0
	20	0	0	0	0	1	1	2	3	0	0
100	2	0	0	0	0	0	0	2	4	0	0
	5	0	0	0	0	0	0	1	5	0	0
	10	0	0	0	0	1	1	0	5	0	0
	20	0	0	0	0	2	2	1	3	0	0
200	2	0	0	0	0	0	0	1	5	0	0
	5	0	0	0	0	0	0	0	6	0	0
	10	0	0	0	0	0	0	0	6	0	0
	20	0	0	0	0	0	0	0	6	0	0

In addition to the above fields, we use H_6 to represent the best schedule obtained by the five greedy heuristics $H_1, H_2, \ldots, H_5$. The reason that we take H_6 here is that, the five greedy heuristics run very fast and are easy to implement. Therefore, in practice, we can always take H_6 to gain over an individual greedy heuristic.

The data in these fields are the number of instances (out of six for each combination of n and m) for which the corresponding algorithms produce the best solutions. Tables 1–5 show the frequencies of each algorithm producing the best solution.

Table 1 shows that for the instances of group G_1 with $k = 2$, LP'_2, LP_1, and *WECT* (which is the only contributor to H_6) have the best performance. A close study of Table 1 reveals that for instances with small n, LP_1 is better than both LP'_2 and *WECT*, and *WECT* is better than LP'_2. However, when n is large, LP'_2 is significantly better than all other

Table 3
The frequency that each algorithm performs best for instances of group G_3

n	m	H_1	H_2	H_3	H_4	H_5	H_6	LP_1	LP'_2	LP''_2	LP'''_2
20	2	0	1	1	0	0	2	2	2	0	0
	5	0	1	0	1	2	4	0	2	0	0
	10	0	0	0	0	3	3	1	2	0	0
	20	1	0	0	0	2	3	1	1	1	0
50	2	0	0	0	0	3	3	2	1	0	0
	5	0	0	0	0	1	1	2	3	0	0
	10	0	0	0	0	3	3	1	3	0	0
	20	0	0	0	0	1	1	2	3	0	0
100	2	0	0	0	0	6	6	0	0	0	0
	5	0	0	0	0	2	2	1	3	0	0
	10	0	0	0	0	3	3	1	2	0	0
	20	0	0	0	0	3	3	0	3	0	0
200	2	0	0	0	0	6	6	0	0	0	0
	5	0	0	0	0	4	4	1	1	0	0
	10	0	0	0	0	4	4	0	2	0	0
	20	0	0	0	0	5	5	0	1	0	0

Table 4
The frequency that each algorithm performs the best for instances of group G_4

n	m	H_1	H_2	H_3	H_4	H_5	H_6	LP_1	LP'_2	LP''_2	LP'''_2
20	2	0	1	1	0	2	4	0	1	0	1
	5	0	0	0	0	3	3	2	0	1	0
	10	0	0	0	0	3	3	1	2	0	0
	20	0	0	0	0	4	4	1	1	0	0
50	2	0	0	0	0	3	3	2	1	0	0
	5	0	0	0	0	3	3	1	2	0	0
	10	0	0	0	0	4	4	0	2	0	0
	20	0	0	0	0	3	3	1	2	0	0
100	2	0	0	0	0	3	3	1	2	0	0
	5	0	0	0	0	1	1	2	3	0	0
	10	0	0	0	0	5	5	0	1	0	0
	20	0	0	0	0	5	5	0	1	0	0
200	2	0	0	0	0	5	5	0	1	0	0
	5	0	0	0	0	6	6	0	0	0	0
	10	0	0	0	0	3	3	1	2	0	0
	20	0	0	0	0	5	5	0	1	0	0

algorithms. The table also reveals a counter-intuitive finding that LP_1 does not perform better than the other algorithms all the time, even though it has the best known worst-case performance bound (note that the performance ratios of LP'_2 and LP''_2 are unknown yet). The subsequent tables will also exhibit this finding.

Table 2 shows that for the instances of group G_2, for which $k = m$ but there are no additional constraints on the properties of processing times, LP'_2, LP_1, and *WECT* (which is again the only contributor to H_6) have the best performance. For small n, it is hard to tell which one of the three algorithms is the best. However, for large n, LP'_2 performs better than LP_1, which in turn performs better than *WECT*.

Table 3 shows that the best algorithm for G_3 is H_6 (with *WECT* being its biggest contributor). LP'_2 is second best, and LP_1 third best. When n is small, *WSTP*, *WSMP*, *WSMC*, and *WSPL* may occasionally beat the other algorithms and

Table 5
The frequency that each algorithm performs best for instances of group G_5

n	m	H_1	H_2	H_3	H_4	H_5	H_6	LP_1	LP'_2	LP''_2	LP'''_2
20	2	0	0	0	0	1	1	3	1	1	0
	5	0	0	0	0	0	0	2	2	1	0
	10	0	0	0	0	1	1	2	2	0	1
	20	0	0	0	0	2	2	2	2	0	0
50	2	0	0	0	0	0	0	2	4	0	0
	5	0	0	0	0	0	0	3	3	0	0
	10	0	0	0	0	0	0	2	3	1	0
	20	0	0	0	0	1	1	3	2	0	0
100	2	0	0	0	0	0	0	1	5	0	0
	5	0	0	0	0	0	0	2	4	0	0
	10	0	0	0	0	0	0	1	5	0	0
	20	0	0	0	0	0	0	0	6	0	0
200	2	0	0	0	0	0	0	0	6	0	0
	5	0	0	0	0	0	0	1	5	0	0
	10	0	0	0	0	0	0	1	5	0	0
	20	0	0	0	0	0	0	0	6	0	0

becomes a contributor to H_6. However, when n becomes large, the tendency is that *WECT* beats all other algorithms. Table 4 shows similar findings for G_4. These results are consistent with our theoretical analysis that the priority rules exhibit a better performance with additional constraints on the processing times, except that the results for LP_1 are somehow counter-intuitive.

Table 5 shows that for the instances of group G_5, for which $k = r$, the two best algorithms are LP'_2 and LP_1. For small n, LP_1 slightly outperforms LP'_2. In addition, for occasional cases of small n, *WECT* (which is the only contributor to H_6), LP''_2, and LP'''_2 may beat the other algorithms. However, for large n, LP'_2 becomes significantly better than all other algorithms.

From these tables, the comparison among LP'_2, LP''_2, and LP'''_2 shows that smaller λ leads to better performance of the H_{LP_2} algorithm, even though there are some exceptions.

Now let us investigate the performance of the algorithms in terms of comparisons of average costs and average running times. Tables 6–10 show the positive ratios of the algorithms that are higher than the average costs of the best algorithms. In addition to the same field labels defined for Tables 1–5, we introduce five more fields:

T_1: The average running time (seconds) per instance for all five priority rules together.
T_2: The average running time (seconds) per instance for LP_1.
T_3: The average running time (seconds) per instance for LP'_2.
T_4: The average running time (seconds) per instance for LP''_2.
T_5: The average running time (seconds) per instance for LP'''_2.

The entries in the columns of the above four fields are simply the average running time in seconds per instance, as defined above. Now focus on the columns corresponding to the algorithms. For these columns, a "–" indicates that an algorithm produces the minimum average objective cost. Note that each row has one and only one "–" for each combination of n and m. In each row, the entries other than "–" are computed as

$$\frac{\text{The average cost of corresponding algorithm} - \text{The minimum average cost}}{\text{The minimum average cost}} \times 100.$$

In order to rank the performance of the algorithms, we use in what follows the notation $A \prec B$ to indicate that algorithm A performs better than algorithm B. Furthermore, if algorithm A is comparable to algorithm B, we write $A \sim B$.

First of all, Tables 6–10 show that, for each n, the percentage of each priority rule (except for *WECT*) tends to increase when m increases. This is consistent with our previous theoretical analysis which indicates that the performance of each priority rule becomes worse when m becomes larger. In contrast, neither H_{LP_1} nor H_{LP_2} exhibit such a relationship

Table 6
Comparison of the algorithms for group G_1 by percentages that the average costs are higher than those of the best ones and by average running times

n	m	H_1	H_2	H_3	H_4	H_5	H_6	LP_1	LP'_2	LP''_2	LP'''_2	T_1	T_2	T_3	T_4	T_5
20	2	5.40	6.06	3.28	5.39	0.39	0.38	0.21	–	0.30	2.62	0.00	0.91	1.34	0.23	0.04
	5	8.51	10.8	5.14	22.4	0.66	–	0.48	2.69	2.55	2.98	0.2	2.72	4.24	1.05	0.37
	10	8.40	8.84	3.95	17.1	2.72	–	2.81	3.46	3.54	3.56	0.07	5.23	7.38	1.29	0.23
	20	18.8	17.2	15.2	25.5	2.7	2.57	–	12.8	12.8	12.9	0.07	12.9	20.8	2.63	0.46
50	2	4.85	5.84	3.82	7.56	1.09	0.93	–	0.45	1.41	2.41	0.06	7.63	9.20	1.88	0.61
	5	12.2	14.7	8.54	30.7	–	–	5.17	4.59	4.91	7.64	0.06	19.1	28.7	5.33	1.26
	10	6.11	7.74	2.79	25.0	7.59	–	2.27	0.21	0.51	1.35	0.09	31.5	58.0	10.0	2.03
	20	11.7	14.1	8.94	22.9	1.25	–	1.0	8.42	8.29	8.96	0.07	70.8	124	20.7	3.64
100	2	3.11	4.79	3.94	8.73	1.35	1.35	1.08	–	1.04	4.81	0.11	37.2	48.8	11.2	3.78
	5	6.59	8.09	4.35	26.7	3.68	2.77	0.45	–	0.92	4.49	0.20	80.3	121	24.5	6.73
	10	8.28	11.8	4.55	30.0	4.44	2.24	0.72	–	0.68	3.11	0.25	191	251	45.2	10.4
	20	7.41	9.43	3.08	24.2	4.43	2.35	0.13	–	0.19	0.95	0.26	401	578	98.5	17.9
200	2	2.82	4.18	3.15	8.50	0.92	0.78	0.73	–	1.14	3.85	0.24	271	354	87.0	32.8
	5	6.31	8.12	3.86	26.6	4.89	3.64	1.39	–	1.11	4.69	0.28	607	687	145	42.4
	10	6.87	8.91	4.03	28.0	6.30	3.65	0.68	–	1.00	3.99	0.30	1152	1440	251	65.0
	20	9.01	9.59	3.77	27.5	11.3	3.77	0.27	–	0.48	2.68	0.26	3059	4368	482	94.7

Table 7
Comparison of the algorithms for group G_2 by percentages that the average costs are higher than those of the best ones and by average running times

n	m	H_1	H_2	H_3	H_4	H_5	H_6	LP_1	LP'_2	LP''_2	LP'''_2	T_1	T_2	T_3	T_4	T_5
20	2	3.94	5.01	3.67	6.72	1.53	1.48	–	0.11	0.53	2.66	0.00	0.92	1.37	0.26	0.04
	5	3.47	4.49	3.26	12.3	0.05	–	0.22	0.08	0.49	1.69	0.01	2.38	4.95	0.68	0.13
	10	4.25	5.06	2.83	8.24	0.12	–	0.24	0.36	0.49	1.97	0.05	6.05	18.1	1.52	0.30
	20	3.54	4.06	3.24	7.94	0.03	0.03	–	0.22	0.31	1.25	0.06	15.2	43.5	5.31	0.81
50	2	3.01	4.02	2.76	8.44	0.81	0.68	0.29	–	0.57	3.04	0.07	4.11	9.34	2.05	0.65
	5	6.78	7.65	4.76	11.5	0.02	0.02	0.62	–	0.56	2.55	0.07	28.8	30.1	5.24	1.18
	10	5.37	7.16	5.30	10.3	0.56	0.30	0.58	–	0.76	2.31	0.10	85.3	104	12.6	2.44
	20	4.42	4.41	3.71	13.3	0.16	0.14	0.21	–	0.68	1.87	0.16	362	518	36.8	6.42
100	2	2.90	4.63	3.77	7.20	1.44	1.25	0.35	–	1.16	3.46	0.07	32.3	48.6	11.7	3.80
	5	4.80	5.95	4.29	12.6	0.93	0.88	0.95	–	1.02	3.36	0.16	70.9	139	25.0	6.22
	10	5.64	7.44	4.94	13.5	0.40	0.33	0.74	–	1.34	2.71	0.29	329	503	54.5	12.0
	20	4.65	5.44	3.83	13.2	0.04	–	0.62	0.03	0.87	2.84	0.28	846	1994	167	26.9
200	2	1.87	3.44	2.74	7.66	0.93	0.80	0.86	–	0.98	3.67	0.28	129	313	85.0	30.6
	5	4.34	6.62	5.30	13.7	0.96	0.88	1.54	–	1.36	3.67	0.27	617	855	161	47.8
	10	5.17	6.76	5.29	13.9	0.71	0.63	1.03	–	1.18	3.13	0.25	2278	3222	358	80.3
	20	4.15	5.41	4.26	15.3	0.65	0.52	0.87	–	1.05	2.97	0.20	7390	15 865	1199	194

between their performance and the value of m. Secondly, these tables also show that a smaller λ leads to a better performance of H_{LP_2}. This is consistent with a previous finding from Tables 1–5.

Now consider the performance of the algorithms for each group of problem instances. Table 6 shows that, for group G_1, when $n \geqslant 100$, LP'_2 is overwhelmingly better than other algorithms. Although it is hard to fix a ranking of performance when n is small, for large n we see that

$$LP'_2 \prec LP_1 \prec LP''_2 \prec H_6 \prec WECT \prec LP'''_2 \sim WSMC \prec WSTP \prec WSMP \prec WSPL.$$

Table 7 shows that, for group G_2, *WECT* (of course, H_6 also) and LP_1 can beat LP'_2 when $n = 20$ (but it is hard to tell which one of the two algorithms is better.) However, when $n \geqslant 50$, LP'_2 is much better than all other algorithms. For

Table 8
Comparison of the algorithms for group G_3 by percentages that the average costs are higher than those of the best ones and by average running times

n	m	H_1	H_2	H_3	H_4	H_5	H_6	LP_1	LP'_2	LP''_2	LP'''_2	T_1	T_2	T_3	T_4	T_5
20	2	2.01	1.92	1.48	2.64	0.44	0.37	0.09	–	0.30	1.48	0.00	0.43	1.50	0.13	0.03
	5	1.98	1.66	1.55	3.57	0.05	0.03	0.24	–	0.25	1.01	0.00	2.41	3.70	0.30	0.05
	10	2.13	2.55	1.32	3.40	0.05	–	0.41	0.32	0.58	1.29	0.01	7.22	10.4	0.69	0.14
	20	1.59	1.83	1.41	4.11	0.02	–	0.2	0.23	0.36	1.37	0.01	16.1	24.4	1.43	0.30
50	2	1.32	1.94	1.46	3.06	–	–	0.11	0.23	0.44	3.76	0.01	5.45	8.51	1.12	0.38
	5	2.71	3.29	2.30	3.72	0.07	0.07	0.26	–	0.37	3.61	0.01	18.3	25.8	2.67	0.69
	10	2.57	2.90	2.19	3.62	0.13	0.06	0.49	–	0.62	2.26	0.01	45.2	60.3	3.79	0.95
	20	1.86	1.86	1.54	4.32	0.07	0.07	0.34	–	0.47	3.57	0.01	129	183	7.23	1.67
100	2	1.16	1.93	1.5	2.51	0.31	–	0.34	0.39	1.0	3.95	0.01	30.9	46.5	7.89	2.44
	5	2.21	2.71	2.13	3.80	0.21	0.20	0.74	–	0.95	4.04	0.02	78.1	101	14.7	3.69
	10	2.59	3.05	2.32	4.30	0.10	0.08	0.60	–	0.85	4.05	0.03	155	196	21.3	5.66
	20	1.98	2.27	1.85	4.03	0.03	0.02	1.76	–	0.79	4.04	0.04	313	783	53.2	7.96
200	2	0.93	1.62	1.29	2.41	0.03	–	0.68	0.76	0.94	3.86	0.05	199	284	84.13	26.0
	5	1.74	2.58	2.10	3.92	0.18	0.17	0.98	–	1.02	3.85	0.06	512	663	133	40.7
	10	2.04	2.68	2.18	4.13	0.15	0.13	0.78	–	0.93	3.42	0.08	1784	2363	204	53.8
	20	1.96	2.38	2.03	4.26	0.09	0.05	1.02	–	0.83	3.22	0.20	4003	10 941	674	101

Table 9
Comparison of the algorithms for group G_4 by percentages that the average costs are higher than those of the best ones and by average running times

n	m	H_1	H_2	H_3	H_4	H_5	H_6	LP_1	LP'_2	LP''_2	LP'''_2	T_1	T_2	T_3	T_4	T_5
20	2	1.58	1.20	0.98	1.85	0.04	–	0.15	0.06	0.19	0.40	0.00	0.53	1.39	0.17	0.03
	5	1.79	1.92	1.83	1.69	0.05	0.04	–	0.09	0.23	1.80	0.00	1.41	3.62	0.39	0.07
	10	1.62	1.97	1.46	4.29	0.01	–	0.32	0.21	0.41	2.86	0.00	4.92	15.43	1.15	0.21
	20	2.02	1.48	1.42	6.64	–	–	0.55	0.27	0.68	1.10	0.01	14.5	27.90	2.05	0.36
50	2	1.04	1.53	1.22	1.81	–	–	0.02	0.07	0.67	3.58	0.01	6.20	9.24	1.61	0.50
	5	1.99	2.56	2.26	3.33	–	–	0.48	0.02	0.59	2.96	0.01	17.7	23.65	3.17	0.80
	10	1.99	2.30	1.76	4.38	0.01	–	1.09	0.01	0.52	3.58	0.01	50.3	72.96	7.64	1.62
	20	1.59	2.20	1.89	6.25	–	–	0.52	0.12	0.73	3.00	0.02	109	231	15.41	3.09
100	2	0.76	1.27	1.11	1.23	0.01	–	0.48	0.02	0.97	4.64	0.01	29.5	43.95	8.76	3.66
	5	2.23	2.26	1.93	3.34	0.18	0.18	0.43	–	0.96	3.76	0.02	88.4	126	19.06	5.29
	10	2.08	2.41	2.11	3.41	–	–	0.75	0.10	0.82	3.75	0.03	209	339	31.20	7.92
	20	1.15	2.28	2.05	4.27	–	–	0.82	0.12	0.74	3.32	0.04	577	1127	98.07	16.16
200	2	0.76	1.22	1.09	1.43	–	–	0.94	0.13	1.03	4.74	0.05	196	288	76.67	26.10
	5	1.25	2.16	1.92	3.20	–	–	1.33	0.10	1.13	4.11	0.06	430	647	147	47.63
	10	1.60	2.37	2.08	4.57	0.04	0.04	0.77	–	0.87	4.08	0.08	981	2752	315	77.53
	20	1.29	2.27	2.05	4.84	–	–	0.71	0.03	0.92	3.92	0.22	3789	13 423	864	118

large n we see that

$$LP'_2 \prec H_6 \prec WECT \prec LP_1 \prec LP''_2 \prec LP'''_2 \prec WSTP \prec WSMC \prec WSMP \prec WSPL.$$

Note that the difference between *WECT* and LP'_2 is less than 1%, and H_6 is only slightly better than *WECT*. Thus, the performance of *WECT* is actually very close to that of LP'_2. However, the table shows that LP'_2 requires hours of running time for large instances, while *WECT* requires only milliseconds. As stated before, the LP'_2 algorithm requires virtual memory up to 1 GB. By contrast, *WECT* only requires several kilobytes of memory.

Table 8 shows that, for group G_3, the best two algorithms are *WECT* (H_6 is only slightly better than *WECT*) and LP'_2. *WECT* tends to perform better than others when n is small, while LP'_2 performs well when n is large. From the

Table 10
Comparison of the algorithms for group G_5 by percentages that the average costs are higher than those of the best ones and by average running times

n	m	H_1	H_2	H_3	H_4	H_5	H_6	LP_1	LP'_2	LP''_2	LP'''_2	T_1	T_2	T_3	T_4	T_5
20	2	5.08	4.06	2.47	12.93	0.14	0.05	–	0.11	0.32	1.37	0.00	0.34	1.05	0.22	0.04
	5	6.01	9.51	4.61	9.42	2.58	1.49	–	0.09	0.38	1.87	0.00	1.62	3.97	0.64	0.12
	10	5.39	7.54	5.04	16.19	0.39	0.31	0.03	–	0.23	1.71	0.07	4.70	9.93	1.52	0.31
	20	5.40	9.17	5.66	12.47	0.02	–	0.26	0.43	0.88	2.02	0.08	11.5	40.58	5.47	0.84
50	2	4.03	5.59	3.71	12.09	2.32	1.92	0.37	–	0.61	4.14	0.06	5.33	8.56	1.76	0.57
	5	7.05	9.78	7.47	15.84	2.49	2.41	0.20	–	0.49	2.81	0.07	19.1	27.17	4.59	1.08
	10	7.68	11.53	8.63	13.37	2.65	2.65	0.75	–	0.68	2.89	0.06	50.5	96.88	17.93	2.91
	20	9.03	15.40	9.31	18.17	1.86	1.86	0.09	–	0.65	2.63	0.26	178	324	47.02	7.04
100	2	3.01	5.02	3.39	15.01	2.12	1.46	1.28	–	1.09	5.14	0.10	15.9	44.18	10.59	3.56
	5	7.40	10.04	7.38	17.12	2.18	2.18	0.59	–	1.00	4.62	0.06	45.7	125	23.05	6.61
	10	7.99	13.35	10.72	15.86	2.40	2.40	0.83	–	0.73	3.46	0.28	186	373	65.16	12.25
	20	6.07	17.11	13.97	16.16	1.63	1.63	0.44	–	0.65	2.93	0.19	694	2594	250	42.53
200	2	2.97	4.99	3.35	17.24	2.24	1.97	0.91	–	1.15	5.14	0.25	139	289	78.82	27.19
	5	6.29	11.51	8.92	17.27	2.87	2.84	1.05	–	1.23	4.26	0.24	306	681	146	42.46
	10	6.67	14.58	11.49	16.93	2.63	2.63	0.83	–	1.01	3.67	0.29	1330	2806	336	77.67
	20	6.64	16.54	13.84	18.71	1.76	1.76	0.64	–	1.11	3.68	0.27	4507	16 300	1229	207

table, we see that

$$LP'_2 \prec H_6 \sim WECT \prec LP_1 \prec LP''_2 \prec WSTP \prec WSMC \prec WSMP \prec LP'''_2 \sim WSPL.$$

Note that the difference between *WECT* and LP'_2 is less than 0.5%. Thus, the performance of *WECT* is almost the same as that of LP'_2. However, to achieve such performance, LP'_2 requires more computational resources than *WECT*. It is also interesting to see that, *WSTP*, *WSMC*, and *WSMP* perform better than LP'''_2.

Table 9 shows that, for group G_4, *WECT* (which might be the only contributor to H_6) is the best. In details, the algorithms are ranked as follows:

$$H_6 \sim WECT \prec LP'_2 \prec LP_1 \prec LP''_2 \prec WSTP \prec WSMC \prec WSMP \prec LP'''_2 \sim WSPL.$$

Again, *WSTP* , *WSMC*, and *WSMP* perform better than LP'''_2.

Table 10 shows that, for group G_5, LP'_2 is the best. The algorithms are ranked as

$$LP'_2 \prec LP_1 \prec LP''_2 \prec H_6 \sim WECT \prec LP'''_2 \prec WSTP \prec WSMC \prec WSMP \prec WSPL.$$

Again, the results produced by *WECT* are quite close to those of the LP-based algorithms.

To observe the cost-effective performance of the H_{LP_2}, we compare for each group of instances in Table 11 the percentages that the average costs of LP''_2 and LP'''_2 are larger than that of LP'_2. From this table, we can see that the gap between the average cost of LP''_2 and that of LP'_2 is actually very small. For most cases, it is less than 1.0%; and the largest one is 1.3%. Therefore, the performance of LP''_2 is actually very close to that of LP'_2. However, from Tables 6 to 10, the average running time of LP'_2 is much more than that of LP''_2 (for some cases, it is more than 10 times). In addition, in the experiments, we noticed that the memory requirement of LP'_2 is twice that of LP''_2. Thus, in practice, if the use of H_{LP_2} is considered, it is recommended to choose $\lambda = \sqrt{2}$ in order to strike a balance between performance and the use of computational resources.

Summarizing the empirical analysis, among the individual algorithms, we recommend the use of either *WECT* or H_{LP_2} with $\lambda = \sqrt{2}$. It should be noted that, even though H_{LP_1} performs better than H_{LP_2} with $\lambda = \sqrt{2}$, and it does not take too much memory space, it runs very slowly. This is the main reason why we do not recommend the use of H_{LP_1} in practice. In an environment which requires a solution to be generated quickly with limited memory spaces, *WECT* is the most preferable. It is simple to implement, requires a small amount of memory, runs fast, and produces good results. However, since the five greedy heuristics run fast and are easy to implement, it always gains to run all of them

Table 11
The percentage that the average costs of LP_2^2 and LP_2^3 are larger than that of LP_2^1

n	m	G_1		G_2		G_3		G_4		G_5	
		LP_2''	LP_2'''	LP_2''	LP_2'''	LP_2''	LP_2'''	LP_2''	LP_2'''	LP_2''	LP_2'''
20	2	0.30	2.62	0.37	2.57	0.35	2.24	0.29	1.75	0.28	1.69
	5	−0.15	0.27	0.26	1.24	0.24	1.13	0.24	1.37	0.25	1.44
	10	0.07	0.09	0.12	1.33	0.17	1.17	0.18	1.65	0.19	1.62
	20	0.00	0.05	0.08	0.88	0.09	0.95	0.20	0.91	0.25	1.07
50	2	0.96	1.96	0.74	2.58	0.67	2.87	0.65	3.03	0.65	3.24
	5	0.31	2.92	0.50	2.63	0.45	3.00	0.49	2.95	0.49	2.92
	10	0.29	1.14	0.69	2.14	0.65	2.17	0.60	2.66	0.62	2.66
	20	−0.12	0.51	0.60	1.72	0.54	2.43	0.57	2.55	0.58	2.55
100	2	1.04	4.81	1.11	3.99	1.06	4.00	1.03	4.19	1.05	4.37
	5	0.92	4.49	0.99	3.62	0.97	3.80	0.97	3.78	0.98	3.96
	10	0.68	3.11	1.25	2.76	1.09	3.27	0.97	3.36	0.93	3.37
	20	0.19	0.95	0.78	2.64	0.78	3.22	0.71	3.17	0.71	3.11
200	2	1.14	3.85	1.04	3.74	1.01	3.77	0.98	4.02	1.02	4.24
	5	1.11	4.69	1.30	3.92	1.19	3.91	1.14	3.95	1.17	4.01
	10	1.00	3.99	1.16	3.25	1.07	3.33	1.00	3.60	1.01	3.59
	20	0.48	2.68	1.00	2.95	0.92	3.06	0.91	3.36	0.96	3.41

on an problem instance and take the best solution. This could be seen from the tables shown previously. To ensure this, we also compare in Table 12 the percentage that the costs of *WECT* with that of H_6. The table shows that, the solutions produced by H_6 are better than those of *WECT* for some problem instances, especially for those instances of group G_1 and those of $m = 2$.

6. Concluding remarks

In this paper, we have focused on several approximation algorithms for the customer order scheduling problem with the minimization of the total weighted completion time as objective. We focused on the design of approximation algorithms for this problem. The procedures include several priority rules as well as two LP-based algorithms. Although priority rules are easy to implement, our analysis showed that the performance guarantees vary according to the distribution properties of the processing times. By contrast, various linear programming relaxations uniformly provide tight lower bounds for approximation algorithms. However, different relaxations may result in approximation algorithm with very different performance guarantees. Fortunately, both LP-based algorithms we presented in this paper have a fixed ratio performance guarantee.

The experimental results revealed that H_{LP_1}, which has the best known worst-case performance bound, does not perform better than H_{LP_2} with $\lambda = 2^{1/4}$. This finding might imply that H_{LP_2} with $\lambda = 2^{1/4}$ has a performance bound tighter than 2. However, we could not show this in this paper.

Both the problems $PDm\|\sum w_jC_j$ and $PDm|r_j|\sum w_jC_j$ are strongly NP-hard. It is not known if there exists any polynomial time approximation scheme (PTAS) for these problems. Afrati et al. [1] have presented a PTAS for the problem $1|r_j|\sum w_jC_j$. It would be interesting to examine if their results shed any light on solving our problems. On the other hand, it would also be challenging to prove that the problems are APX-hard if such PTAS does not exist unless $P = NP$.

As extension of this work, it would be interesting to have further dedicated study on the problem with the presence of release dates, that is, $PDm|r_j|\sum w_jC_j$. For example, it would be worth to extend the five greedy heuristics to solve the problem, and to have an experimental analysis of these heuristics together with the LP-based algorithm H_{LP_1}, which we analyzed its performance bound but did not have experimental analysis. It would also be interesting to investigate if the online $(3+\varepsilon)$-approximation algorithm for $1|r_j|\sum w_jC_j$ due to Hall et al. [9] can be extended to solve the online version of our problem.

Table 12
The percentage that the costs of *WECT* are larger than the best costs among all five greedy heuristics

	n	20				50				100				200			
	m	2	5	10	20	2	5	10	20	2	5	10	20	2	5	10	20
G_1	1	0	0.94	8.17	8.32	0.14	1.8	2.61	7.18	0	0.41	6.61	8.16	0	1.58	8.72	10.6
	2	0.04	4.44	8.58	4.01	0	6.19	4.79	8.75	0	1.2	2.86	5.87	0	2.16	5.45	10.8
	3	0	2.18	4.66	5.79	0.75	1.4	3.78	11.2	0	1.06	5.8	14.3	0	3.82	5.6	9.44
	4	0	3.78	4.2	5.04	0	0	2.71	17.3	0	2.57	7.02	12.5	0	1.75	5.93	9.65
	5	0	0	6.7	4.71	0	0	6.4	11.6	0	3.7	4.78	7.7	0	1.14	4.73	10.6
	6	0	4.69	8.94	0	0	1.96	2.03	8.01	0	2.63	2.62	11.1	0.76	2.03	5.33	6.32
G_2	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	0	0.61	0	0	0	0.17	0	0	0
	9	0.4	0	0	0	0.69	0	0	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	0	0	0	0.21	0	0	0
	11	0	0	0	0	0	0	0	0	0	0	0	0	0.43	0	0	0
	12	0	0	0	0	0	0	0	0	0.78	0	0	0	0	0	0	0
G_3	13	0.84	0	0.25	0	1.54	0	0.67	0	0.27	0	0	0	0.22	0	0	0
	14	0	6.62	2.62	0	0	0	0	0	0	0	0	0	0.31	0	0	0
	15	3.49	0	0.43	0	1.2	0	0	0	0	0	0	0	0.63	0	0	0
	16	0	0	0	0	0.05	0	0	0	0	0	0	0	0	0	0	0
	17	0	2.1	0	7.96	0	2.57	0	0	2.78	0	0	0	0.8	0.26	0	0
	18	0	0	0	0	0	0	0	0	2.02	0	0	0	0	0	0	0
G_4	19	7.36	1.55	5.22	0	0.18	0	0	0	0.21	0	0	0	0	0	0	0
	20	1.31	0	0	5.88	0.73	0	4.27	0	0	0	0	0	1.45	0	0	0
	21	1.9	0	0	0	0	0	0	0	1.7	0	0	0	0	0	0	0
	22	1.78	0.73	0.86	0	0	0	0	0.01	0.32	0	0	0	2.95	0	0	0
	23	3.18	0	0	0	0.07	0	0	0	0	0	0	0	0	0.05	0	0
	24	8.86	2.85	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0
G_5	25	4.13	3.69	0	0.45	0	0	0	0	1.52	0	0	0	0	0	0	0
	26	0	0	0	0	2.13	0	0	0	0	0	0	0	0	0	0	0
	27	0.15	0	0	0	1.15	0	0	0	0	0.39	0	0	0	0	0	0
	28	0	0	3.15	1.31	0	0	0	0	0.88	0	0	0	0	0	0	0
	29	2.21	0.4	0	0	0	0	0	0	0	0	0	0	2.89	0	0	0
	30	0.67	0	0	0	1.39	0.46	0	0	0	0	0	0	0	0	0	0

There is another interesting issue that comes up when orders have different release dates. In this paper we allowed unforced idleness, i.e., the decision-maker is allowed to keep a machine idle in anticipation of an order with a high weight coming in, even though another (old) order may be waiting for its products. Clearly, this is a reasonable assumption in certain real world applications of our model, e.g., the equipment maintenance and repair application. However, in this paper, we assumed that a facility is *not* allowed to preproduce for an order that has not come in yet. In the manufacturing world, if it is known what the total quantity of product requested is (in current as well as in future orders), then the machines may be kept running producing all the different product types for orders that have not come in yet (i.e., unforced idleness is not allowed). In this case, when an order is released at a certain time, there may already be a sufficient number of the various different product types available that can be assigned to the new order, allowing for an immediate shipment. This way, the lead times of the orders may be reduced considerably. For the LP-based algorithm using completion time formulation, it is easy to modify the formulation in such a way that it meets this requirement. For example, we only need to change constraints (32) as follows:

$$C_j \geqslant \max\{r_j, p_{ij}\}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n.$$

We keep the other constraints unchanged. Step 2 of this algorithm has to be changed slightly, so that it is not required to insert idle time. Simple analysis shows that, under the new assumption, H_{LP_1} becomes a 2-approximation

algorithm for $PDm|r_j|\sum w_jC_j$. However, for the LP-based algorithm using time-interval formulation, it turns out that it is not so easy to incorporate the new assumption into either the LP formulation or the algorithm.

We believe that these types of assumptions are interesting for manufacturing and other settings. It would be of interest to consider such assumptions in future research.

Acknowledgments

The authors gratefully acknowledge the support by the National Science Foundation through grants DMI-0300156 and DMI-0245603. They also thank the anonymous referees for their comments which are valuable to improve the paper.

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, M. Sviridenko, Approximation schemes for minimizing average weighted completion time with release dates, The Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS'99), 1999, pp. 32–43.
[2] R.H. Ahmadi, U. Bagchi, Scheduling of Multi-job Customer Orders in Multi-machine Environments, ORSA/TIMS, Philadelphia, 1990.
[3] R.H. Ahmadi, U. Bagchi, Coordinated scheduling of customer orders, Working paper, John E. Anderson Graduate School of Management, University of California, Los Angeles, 1993.
[4] R.H. Ahmadi, U. Bagchi, T. Roemer, Coordinated scheduling of customer orders for quick response, Naval Res. Logist. 52 (2005) 493–512.
[5] C. Chekuri, S. Khanna, Approximation algorithms for minimizing average weighted completion time, in: J.Y.-T. Leung (Ed.), Handbook of Scheduling: Algorithms, Models, and Performance Analysis, CRC Press, Boca Raton, FL, USA, 2004.
[6] Z.-L. Chen, N.G. Hall, Supply chain scheduling: assembly systems, Working Paper, Department of Systems Engineering, University of Pennsylvania, 2001.
[8] M. Grötschel, L. Lovasz, A. Schrijver, Geometric Algorithms and Combinatorial Optimization, Springer, Berlin, 1993.
[9] L.A. Hall, A.S. Schulz, D.B. Shmoys, J. Wein, Scheduling to minimize average completion time: off-line and on-line approximation algorithms, Math. Oper. Res. 22 (1997) 513–544.
[10] J.Y.-T. Leung, H. Li, M.L. Pinedo, Order scheduling in an environment with dedicated resources in parallel, J. Sched. 8 (2005) 355–386.
[11] J.Y.-T. Leung, H. Li, M.L. Pinedo, Scheduling multiple product types with due date related objectives, Eur. J. Oper. Res. 168 (2006) 370–389.
[12] J.Y.-T. Leung, H. Li, M.L. Pinedo, S. Sriskandarajah, Open shops with jobs overlap—revisited, Eur. J. Oper. Res. 163 (2005) 569–571.
[13] A. Makhorin, GNU linear programming kit: reference manual (version 4.8), 2005.
[14] M.L. Pinedo, Scheduling: Theory, Algorithms, and Systems, second ed., Prentice-Hall, Upper Saddle River, NJ, 2002.
[15] M. Queranne, Structure of a simple scheduling polyhedron, Math. Program. 58 (1993) 263–285.
[16] T.A. Roemer, A note on the complexity of the concurrent open shop problem, J. Sched. 9 (2006) 389–396.
[17] W.E. Smith, Various optimizers for single stage production, Naval Res. Logist. Quart. 3 (1956) 59–66.
[18] C.S. Sung, S.H. Yoon, Minimizing total weighted completion time at a pre-assembly stage composed of two feeding machines, Internat. J. Production Econ. 54 (1998) 247–255.
[19] E. Wagneur, C. Sriskandarajah, Open shops with jobs overlap, Eur. J. Oper. Res. 71 (1993) 366–378.
[20] G. Wang, T.C.E. Cheng, Customer order scheduling to minimize total weighted completion time, Proceedings of the First Multidisciplinary Conference on Scheduling Theory and Applications, 2003, pp. 409–416.