Annals of Operations Research (2020) 285:295–314
https://doi.org/10.1007/s10479-019-03172-8

S.I.: PROJECT MANAGEMENT AND SCHEDULING 2018

Scheduling equal length jobs with eligibility restrictions

Juntaek Hong[1] · **Kangbok Lee**[1] · **Michael L. Pinedo**[2]

Published online: 21 February 2019
© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract
We consider the problem of scheduling independent jobs on identical parallel machines to minimize the total completion time. Each job has a set of eligible machines and a given release date, and all jobs have equal processing times. For the problem with a fixed number of machines, we determine its computational complexity by providing a polynomial time dynamic programming algorithm. We also present two polynomial time approximation algorithms along with their worst case analyses. Experiments with randomly generated instances show that the proposed algorithms consistently generate schedules that are very close to optimal.

Keywords Parallel machine scheduling · Eligibility · Release date · Equal processing time jobs · Total completion time

1 Introduction

Scheduling problems with jobs that have equal processing times have always received a certain amount of attention in the past, due to theoretical interest as well as practical applications, see Lee et al. (2011). In many standardized systems in practice, jobs consistently have exactly the same processing times, e.g., transmission packets in data communication networks, Full Truck Loads (FTLs) in transportation by truck, and Twenty-foot Equivalent Units (TEUs) in container shipments. For more detailed information, see a survey paper by Kravchenko and Werner (2011). From a theoretical point of view scheduling problems with equal processing time jobs are also of interest. While many scheduling problems with jobs that have arbitrary processing times turn out to be NP-hard, their special cases with all jobs having equal processing times are considerably easier and often polynomial time solvable. Even though there has been extensive research in the time complexity of scheduling problems with equal processing times (see, for example, Baptiste et al. 2004), some cases remain still open from a computational complexity point of view.

✉ Kangbok Lee
kblee@postech.ac.kr

[1] Department of Industrial and Management Engineering, Pohang University of Science and Technology, Pohang, Gyeongbuk, Korea

[2] Department of Information, Operations and Management Sciences, Stern School of Business, New York University, New York, NY, USA

In most parallel machine scheduling problems, each job can be processed on any machine when the machine is available. However, recent research has focused on more general problems with jobs that cannot be processed on just any machine, but only on a specific subset of the machines. Such a constraint is referred to as an *eligibility constraint*, see Hwang et al. (2004). Different terminology has been proposed by other researchers, i.e, multi-purpose machines by Brucker et al. (1997), processing set restrictions by Glass and Mills (2006), etc. The reader is referred to a survey paper by Leung and Li (2008). This type of scheduling problem also serves as an important special case of unrelated parallel machine variants; when a job is not eligible for a certain machine, its processing time on that machine is regarded as ∞.

Scheduling with equal processing time jobs subject to eligibility constraints has not received much attention in the literature. Lee et al. (2011) considered a parallel machine scheduling problem to minimize the makespan with equal processing time jobs subject to eligibility constraints and developed a polynomial time algorithm. There have been several results on scheduling with unit processing time jobs subject to eligibility constraints, see Glass and Mills (2006) and Li (2006).

We consider a problem of scheduling n jobs on m identical parallel machines to minimize the total completion time. Let $J = \{1, \ldots, n\}$ be the set of jobs and $M = \{1, \ldots, m\}$ be the set of machines. Job j can be processed on a subset of the m machines, which is called the *eligible set* of job j and is denoted by M_j for $j \in J$. Job j also has a given release date r_j, and all jobs have an equal processing time p. We assume that both r_j and p are integers. According to the well-established $\alpha \mid \beta \mid \gamma$ notation for scheduling problems, as described by Pinedo (2016), this problem can be denoted by $P \mid M_j, r_j, p_j = p \mid \sum C_j$. When the number of machines is considered a fixed constant, the problem is denoted by $Pm \mid M_j, r_j, p_j = p \mid \sum C_j$.

In order to investigate the computational complexity of the problem $P \mid M_j, r_j, p_j = p \mid \sum C_j$, it is important to check the closely related problems, that are its immediate special and general cases. When the eligibility constraints (M_j) are relaxed, each job is eligible for processing on any one of the machines and the problem becomes $P \mid r_j, p_j = p \mid \sum C_j$. Simons (1983) proved that this relaxed problem can be solved in polynomial time, even with deadline constraints. Moreover, problems with more general objectives, i.e., $P \mid r_j, p_j = p \mid \sum w_j C_j$ and $P \mid r_j, p_j = p \mid \sum T_j$, can be solved in polynomial time as well, see Brucker and Kravchenko (2005, 2008), respectively. When the release dates of all jobs are zero, the problem $P \mid M_j, p_j = p \mid \sum C_j$ can be solved in polynomial time; it is known that a more general problem $R \mid\mid \sum C_j$ can be solved in $O(n^3)$ time through an assignment formulation, see Bruno et al. (1974) and Horn (1973). When processing times are arbitrary, the problem $P \mid M_j, r_j \mid \sum C_j$ is strongly NP-hard; Lenstra et al. (1977) proved that the much easier problem $1 \mid r_j \mid \sum C_j$ is already strongly NP-hard. When all jobs have unit processing times, problem $P \mid M_j, r_j, p_j = 1 \mid \sum C_j$ can be solved in polynomial time; Brucker et al. (1997) proved that the problem with a more general objective, i.e., problem $P \mid M_j, r_j, p_j = 1 \mid \sum w_j T_j$ can be solved in $O(n^3)$ time via an assignment formulation. However, the computational complexity of the problem $P \mid M_j, r_j, p_j = p \mid \sum C_j$ is not yet known.

Approximation algorithms were derived for some hard scheduling problems. An algorithm is said to be a δ-approximation algorithm for a minimization problem if, for any instance of the problem, the algorithm generates a solution of which the objective value is at most δ times the optimum. The δ is referred to as the approximation ratio, or worst case ratio, or worst case error bound, or approximation factor (Hochbaum 1996). In this paper the term *worst case performance ratio* is used for δ. Skutella (2001) proposed a 3/2-approximation

algorithm for $R \mid\mid \sum w_j C_j$ and a 2-approximation algorithm for $R \mid r_j \mid \sum w_j C_j$, both being a strongly polynomial time solvable algorithm. Recently, Im and Li (2017) proposed a better approximation algorithm for $R \mid r_j \mid \sum w_j C_j$ with a worst case performance ratio of 1.8786, and Li (2017) proposed a better approximation algorithm for $R \mid\mid \sum w_j C_j$ with a worst case performance ratio of $1.5 - c$ where c is $\frac{1}{6000}$. We propose a polynomial time approximation algorithm for $P \mid M_j, r_j, p_j = p \mid \sum C_j$ with a worst case performance ratio of 1.4.

In Sect. 2, we propose a dynamic programming algorithm for $Pm \mid M_j, r_j, p_j = p \mid \sum C_j$ and show that this algorithm runs in polynomial time when m is fixed. In Sect. 3, a mixed integer programming (MIP) model is proposed for generating an exact solution. Since the computational complexity of $P \mid M_j, r_j, p_j = p \mid \sum C_j$ is unknown, we provide two approximation algorithms along with their worst case analyses in Sect. 4. In Sect. 5, a practical algorithm is proposed and evaluated with randomly generated problem instances, and its experimental results are analyzed. Section 6 shows that the obtained results can be generalized to more general settings, and the concluding remarks are presented in Sect. 7.

2 Dynamic programming algorithm with fixed *m*

In this section, we develop a polynomial time dynamic programming (DP) algorithm for $Pm \mid M_j, r_j, p_j = p \mid \sum C_j$ with fixed m. Before developing the DP algorithm, we introduce additional notation and present two propositions that are useful for the development and analysis of the algorithm.

2.1 Preliminary

If $\mathcal{M} := \{M_j \mid j \in J\}$, then, $\mathcal{M} \subset 2^M \backslash \emptyset$. Let $k := |\mathcal{M}|$ and, without loss of generality, $\mathcal{M} := \{M^1, M^2, \ldots, M^k\}$. Note that if we consider arbitrary eligibility, $k \leq 2^m - 1$.

Proposition 1 *There exists an optimal schedule in which jobs that have the same eligible set and that are assigned to the same machine are scheduled in ERD (Earliest Release Date first) order.*

Proof By exchange argument. □

From Proposition 1, we define a partition of the set of jobs as follows:

$$J^e = \{j \mid M_j = M^e\} \quad \text{for } e = 1, \ldots, k.$$

Let $n^e := |J^e|$ for $e = 1, \ldots, k$. Thus, $\sum_{e=1}^{k} n^e = n$. We assume that jobs in J^e are sorted in ERD order. Let $r^e(j)$ denote the release date of the j-th job in J^e for $j = 1, \ldots, n^e$ and $e = 1, \ldots, k$.

Proposition 2 *There exists an optimal schedule in which the completion time of each job is of the form $r_j + a \cdot p$ for some $j \in J$ and $a \in \{1, \ldots, n\}$.*

Proof If not, we can move the job forward and find a better schedule. □

From Proposition 2, we define the following set of candidates for the completion times of the jobs.

$$\Lambda = \left\{t \mid t = r_j + a \cdot p, \text{ for } j \in J, a \in \{1, \ldots, n\}\right\}$$

Thus, $|\Lambda| \leq n^2$.

2.2 Dynamic programming algorithm

Now we are ready to design a dynamic programming algorithm for the problem. We consider a partial schedule with the first b^e jobs from J^e for $e = 1, \ldots, k$ in which the latest completion time of machine i is t_i for $t_i \in \Lambda$ and $i = 1, \ldots, m$.

Let $\alpha_i := (t_i; b_i^1, b_i^2, \ldots, b_i^e, \ldots, b_i^k)$ be the state of machine i where b_i^e jobs among the first b^e jobs from J^e are scheduled on machine i for $e = 1, \ldots, k$ and where $\sum_{i=1}^m b_i^e = b^e$. Note that if $i \notin M^e$, then $b_i^e = 0$. We know that the following restriction suffices:

$$t_i \in \Lambda$$
$$b_i^e \in \{0, 1, \ldots, n^e\}$$

The state of a partial schedule is a collection of states of all machines and is denoted by $\alpha := (\alpha_1, \alpha_2, \ldots, \alpha_m)$. Let $V(\alpha)$ be the total completion time of the current partial schedule.

We propose a dynamic programming formulation for the problem by describing the boundary conditions, the recursive relationship, and the optimal value function.

Boundary conditions

- $V(\alpha^0) = 0$ where $\alpha^0 := (\alpha_1^0, \alpha_2^0, \ldots, \alpha_m^0)$ and $\alpha_i^0 := (0; 0, \ldots, 0)$ for $i \in \{1, \ldots, m\}$.
- $V(\alpha) = \infty$ if there exists an i such that $t_i < 0$ or if there exist an e and an i such that $b_i^e < 0$.

Recursive relationship

- $V(\alpha) = \min \left\{ V\left(\hat{\alpha}(h, f)\right) + t_h \;\middle|\; f \in \{1, \ldots, k\}, h \in M^f, r^f(b^f) \leq t_h - p, t_h \in \Lambda \right\}$

where $\hat{\alpha}(h, f) = (\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_m)$, $b^f = \sum_{i=1}^m b_i^f$, and

$$\hat{\alpha}_i = \alpha_i \text{ for } i \neq h$$
$$\hat{\alpha}_h = (\hat{t}_h; \hat{b}_h^1, \ldots, \hat{b}_h^k)$$

and

$$\hat{b}_h^e = b_h^e \quad \text{for } e \neq f$$
$$\hat{b}_h^f = b_h^f - 1$$
$$\hat{t}_h \leq t_h - p \quad \text{and} \quad \hat{t}_h \in \Lambda.$$

If there does not exist a pair of (h, f) for $f \in \{1, \ldots, k\}$, $h \in M^f$ satisfying $r^f(b^f) \leq t_h - p$, then $V(\alpha)$ is defined to be ∞.

Optimal value function

- $\min\{V(\alpha) \mid b^e = n^e \text{ for } e \in \{1, \ldots, k\}\}$.

Now we consider the time complexity of the proposed DP algorithm. The number of possible states is bounded by

$$\left(n^2 \times \prod_{e=1}^{k} (1 + n^e)\right)^m \leq O\left((n^2 n^k)^m\right) = O\left(n^{(2+k)m}\right).$$

In order to compute the value of a state, by the recursive relationship, we have to try all h, f and $\hat{t}_h$ and find the minimum; this has to be done kmn^2 times. Thus, the required computation is bounded by

$$kmn^2 \times O\left(n^{(2+k)m}\right) = O\left(kmn^{2+(2+k)m}\right).$$

Since for an arbitrary eligibility case $k \leq 2^m - 1$, the time complexity is

$$O\left(m2^m n^{2+(1+2^m)m}\right).$$

For fixed m, the running time of the proposed DP algorithm is polynomial and $Pm \mid M_j, r_j, p_j = p \mid \sum C_j$ can, therefore, be solved in polynomial time.

3 MIP formulation

Although we proposed in the previous section a DP algorithm that has a polynomial running time for fixed m, its time complexity is so high that the DP cannot be used in practice. Therefore, we propose a MIP formulation for generating an exact solution.

There are several types of MIP formulations for scheduling problems. In the existing scheduling literature, the fundamental differences between the various types of formulations lie in the choice of decision variables, see Keha et al. (2009). The job-machine assignment decisions are expressed by binary variables. The job sequence on a given machine is also expressed by other binary variables. For the job sequence: (i) predecessor variables between two jobs, (ii) immediate predecessor variables between two jobs, or (iii) position variables within the sequence are frequently used. Unlike aforementioned variables, time-indexed variables are also used. Since the model with time-indexed variables is known to have a strong linear programming relaxation, it can solve small size problems very efficiently. On the other hand, its problem size is pseudo-polynomial and it may not work well for large size problems. However, for the problem being considered here, by Proposition 2, we can use a time-indexed formulation with a polynomial number of variables and constraints.

Recall that $\Lambda = \left\{t \mid t = r_j + a \cdot p \text{ for } j \in J, a \in \{1, \ldots, n\}\right\}$. Let λ_i be the i-th smallest element in Λ, i.e., $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_q\}$ such that $\lambda_1 < \lambda_2 < \ldots < \lambda_q$. Note that $|\Lambda| = q$ is bounded by n^2(i.e., $q \leq n^2$). We define the following index set that is useful for describing the formulation.

$$\Gamma^r = \{h \in \{1, \ldots, q\} \mid \lambda_r - p < \lambda_h \leq \lambda_r\}$$

for $r \in \{1, \ldots, q\}$. Thus, $|\Gamma^r| \leq n$.

We define the following decision variables: $x_{irj} = 1$ if job j is scheduled on machine i and its completion time is λ_r, and $x_{irj} = 0$ otherwise, for $i \in M, r \in \{1, \ldots, q\}, j \in J$.

The proposed mathematical programming formulation is as follows:

$$\text{Minimize} \quad \sum_{i \in M} \sum_{r=1}^{q} \sum_{j \in J} \lambda_r x_{irj} \tag{1}$$

$$\text{Subject to} \quad \sum_{i \in M} \sum_{r=1}^{q} x_{irj} = 1 \qquad \text{for } j \in J \tag{2}$$

$$x_{irj} + \sum_{h \in \Gamma^r} \sum_{\ell \in J \setminus \{j\}} x_{ih\ell} \leq 1 \qquad \text{for } i \in M, r \in \{1, \ldots, q\}, j \in J \tag{3}$$

$$x_{irj} = 0 \qquad \text{for } i \notin M_j, r \in \{1, \ldots, q\}, j \in J \tag{4}$$

$$x_{irj} = 0 \qquad \text{for } i \in M, \lambda_r < r_j + p \tag{5}$$

$$x_{irj} \in \{0, 1\} \qquad \text{for } i \in M, r \in \{1, \ldots, q\}, j \in J \tag{6}$$

The objective function (1) describes the total completion time. Constraint set (2) implies that each job must be scheduled exactly once on one machine. Constraint set (3) implies that, when job j is finished at time λ_r on machine i, job j occupies the time period $(\lambda_r - p, \lambda_r]$ of machine i so that no other job can be completed at time λ_h for $h \in \Gamma^r$ on machine i. Constraint set (4) means that each job can only be scheduled on one of its eligible machines. Constraint set (5) implies that each job can only start at its release date or later. Constraint set (6) means domain constraints. The number of variables and the number of constraints are $O(mn^3)$ since $q \leq n^2$.

4 Approximation algorithms

In this section, we propose two approximation algorithms that run in polynomial time. For convenience of explanation, we first propose a simple approximation algorithm, namely Algorithm MR (Modified Release dates), and provide a worst case analysis. Then, we present a more elaborate approximation algorithm, namely Algorithm IMR (Iterative Modified Release dates), along with its worst case analysis.

4.1 Algorithm MR

Let I be an instance of the problem and let $\sigma(I)$ be an optimal schedule of I. Let $z(\sigma)$ be the objective value under schedule σ. Thus, $z(\sigma(I))$ is the objective function value of the optimal schedule of I. We consider now two problem instances that are variations of I:

- I_L: r_j is redefined as $\left\lfloor \frac{r_j}{p} \right\rfloor \times p$ for $j \in J$
- I_U: r_j is redefined as $\left\lceil \frac{r_j}{p} \right\rceil \times p$ for $j \in J$

Since the release dates of I_L and I_U are integer multiples of p, by scaling, these can be regarded as problem instances of $P \mid M_j, r_j, p_j = 1 \mid \sum C_j$ with r_j being a nonnegative integer. Recall that $P \mid M_j, r_j, p_j = 1 \mid \sum C_j$ can be solved through an assignment problem formulation in polynomial time.

The solution of I_L may violate the release dates constraints but its objective function value serves as a lower bound for the optimal objective function value of the original problem instance I. Let $\sigma(I_L)$ be an optimal schedule for problem instance I_L and $\sigma'(I_L)$ be a feasible schedule for the original problem by delaying jobs in schedule $\sigma(I_L)$ minimally.

The solution of I_U satisfies all the constraints and thus its objective function value serves as an upper bound for the optimal objective function value of the original problem instance I. Let $\sigma(I_U)$ be an optimal schedule for problem instance I_U and $\sigma'(I_U)$ be a feasible schedule for the original problem instance by minimally moving up jobs from schedule $\sigma(I_U)$.

We have now the following relationships among the objective function values of the different schedules:

$$z(\sigma(I_L)) \leq z(\sigma(I)) \leq z(\sigma'(I_L)) \tag{7}$$

$$z(\sigma(I)) \leq z(\sigma'(I_U)) \leq z(\sigma(I_U)) \tag{8}$$

Algorithm MR
Choose of $\sigma'(I_L)$ and $\sigma'(I_U)$ the better schedule.

Theorem 1 *Algorithm MR is a* (5/3)*-approximation algorithm for* $P \mid M_j, r_j, p_j = p \mid \sum C_j$ *that runs in polynomial time; its worst case performance ratio is tight.*

Proof Let σ^{MR} denote the schedule generated by Algorithm MR. By the definition of σ^{MR},

$$z(\sigma^{MR}) = \min\left\{z(\sigma'(I_L)), z(\sigma'(I_U))\right\}. \tag{9}$$

We derive two upper bounds for $z(\sigma^{MR}) - z(\sigma(I))$ and one lower bound for the optimal objective function value, $z(\sigma(I))$.

Note that $\lceil \frac{r_j}{p} \rceil \times p \leq \lfloor \frac{r_j}{p} \rfloor \times p + p$. So, if the optimal solution of I_L is delayed by p time units it becomes a feasible schedule for I_U. Thus, $z(\sigma(I_U)) \leq z(\sigma(I_L)) + np$. By (7)–(9), we have

$$z(\sigma^{MR}) - z(\sigma(I)) \leq np. \tag{10}$$

Let J' be the set of jobs that start in schedule $\sigma'(I_L)$ immediately at their release dates and let J'' be $J \backslash J'$. Since jobs in J'' cannot start at their release dates in schedule $\sigma'(I_L)$, there must exist at least one scheduled job right before them in schedule $\sigma'(I_L)$.

When schedule $\sigma'(I_L)$ is constructed from schedule $\sigma(I_L)$, job $j \in J'$ is delayed by $(r_j \mod p)$ and job $j \in J''$ is delayed by at most p. Thus, we have

$$z(\sigma'(I_L)) - z(\sigma(I_L)) \leq \sum_{j \in J'} (r_j \mod p) + \sum_{j \in J''} p.$$

By the above inequality, (7) and (9), we have

$$z(\sigma^{MR}) - z(\sigma(I)) \leq \sum_{j \in J'} (r_j \mod p) + \sum_{j \in J''} p. \tag{11}$$

By (7) and the fact that the completion time of each job in J' is at least p and the completion time of each job in J'' is at least $2p$ in schedule $\sigma(I_L)$, we have

$$\begin{aligned} z(\sigma(I)) &\geq z(\sigma(I_L)) \\ &\geq \sum_{j \in J'} p + \sum_{j \in J''} 2p \\ &= np + \sum_{j \in J''} p. \end{aligned} \tag{12}$$

Since the completion time of job j in an optimal schedule is at least $r_j + p$, we have

$$z(\sigma(I)) \geq np + \sum_{j \in J} r_j. \tag{13}$$

By (12) and (13), we have

$$z(\sigma(I)) \geq np + \frac{1}{2}\left(\sum_{j \in J} r_j + \sum_{j \in J''} p\right). \tag{14}$$

By (10), (11), (14), and the fact that $\sum_{j \in J} r_j \geq \sum_{j \in J'} (r_j \mod p)$,

$$\frac{z(\sigma^{MR})}{z(\sigma(I))} \leq 1 + \frac{\min\left\{np, \sum_{j \in J'} (r_j \mod p) + \sum_{j \in J''} p\right\}}{np + \frac{1}{2}\left(\sum_{j \in J} r_j + \sum_{j \in J''} p\right)} \leq \frac{5}{3}.$$

Therefore, Algorithm MR is a (5/3)-approximation algorithm for $P \mid M_j, r_j, p_j = p \mid \sum C_j$.

It takes polynomial time to generate schedules $\sigma(I_L)$ and $\sigma(I_U)$ through an assignment formulation and it takes linear time to modify them to get $\sigma'(I_L)$ and $\sigma'(I_U)$. Thus, Algorithm MR is a polynomial time algorithm.

In order to show the tightness of the obtained worst case performance ratio, the following problem instance and their schedules are presented. Consider two jobs with $r_1 = 1, M_1 = \{1\}$, and $r_2 = p - 1, M_2 = \{1\}$. Since both jobs are only eligible to machine 1, we only need to consider their starting times.

In problem instance I_L, the modified release dates of both jobs are zero. In $\sigma(I_L)$, job 2 can be scheduled earlier. The completion times of jobs 2 and 1 in $\sigma'(I_L)$ are $2p - 1$ and $3p - 1$, respectively, and $z(\sigma'(I_L)) = 5p - 2$. Similarly, with problem instance I_U, the modified release dates of both jobs are p. In $\sigma(I_U)$, job 2 can be scheduled earlier, and $z(\sigma'(I_U)) = 5p - 2$. However, in the optimal schedule, job 1 is scheduled earlier and the optimal objective function value is $z(\sigma(I)) = (p + 1) + (2p + 1) = 3p + 2$.

$$\lim_{p\to\infty} \frac{\min\{z(\sigma'(I_L)), z(\sigma'(I_U))\}}{z(\sigma(I))} = \frac{5}{3}.$$

Therefore, the proposed worst case performance ratio of Algorithm MR is tight. □

This worst case performance ratio is not great, but the algorithm works very well in practice. Moreover, it is asymptotically optimal as the number of jobs increases. By (10), the difference between $z(\sigma^{MR})$ and $z(\sigma(I))$ is bounded by np which is a linear function of n. However, both $z(\sigma(I_U))$ and $z(\sigma(I))$ are $o\left((\frac{n}{m})^2\right) \times p$. Thus,

$$\lim_{n\to\infty} \frac{z(\sigma^{MR})}{z(\sigma(I))} \leq 1 + \lim_{n\to\infty} \frac{z(\sigma(I_U)) - z(\sigma(I))}{z(\sigma(I))} = 1.$$

4.2 Algorithm IMR

In Algorithm MR, the release dates in the original problem instance I are adjusted in order to be integer multiples of p. In the following algorithm, we consider more general cases where the release dates are modified to have the same remainder after division by p.

Let I^l denote the problem instance that is the same as problem instance I except for r'_j; the modified release dates of job j are

$$r'_j = \left\lceil \frac{r_j - l}{p} \right\rceil \times p + l$$

for $l \in L$ where $L := \{l \in \mathbb{Z} \mid 0 \leq l < p\}$. Thus, I^0 is equivalent to I_U and I^l can also be solved optimally through an assignment problem formulation similar to the case of I_U. Let $\sigma(I^l)$ denote an optimal schedule of instance I^l and let $\sigma'(I^l)$ denote a feasible schedule for the original problem by minimally moving up the jobs from schedule $\sigma(I^l)$.

Algorithm IMR
Choose the best schedule among $\sigma'(I_L)$ and $\sigma'(I^l)$ for $l \in L$.

Theorem 2 *The worst case performance ratio of Algorithm IMR for* $P \mid M_j, r_j, p_j = p \mid \sum C_j$ *is at most 1.4.*

Proof Let σ^{IMR} denote the schedule generated by Algorithm IMR. By the definition of σ^{IMR},

$$z(\sigma^{IMR}) = \min\left\{z(\sigma'(I_L)), \min_{l\in L}\left\{z(\sigma'(I^l))\right\}\right\}. \tag{15}$$

□

We define a schedule $\sigma^l(I)$ as a schedule from which an optimal schedule for instance I, $\sigma(I)$, can be derived by minimally delaying the jobs such that the starting time of each job is of the form of $a \cdot p + l$ for some nonnegative integer a.

Let $d(l)$ be the sum of the differences between the completion times of all jobs in $\sigma(I)$ and $\sigma^l(I)$. Thus, $d(l) := z(\sigma^l(I)) - z(\sigma(I))$.

Lemma 1

$$\min_{l\in L} d(l) \le n(p-1)/2$$

Proof of Lemma 1 Suppose Lemma 1 is not true, then $d(l) > n(p-1)/2$ for all $l \in L$. Let $C_j(\sigma)$ be the completion time of job j in schedule σ, and let $n_l = |\{j \in J \mid C_j(\sigma(I)) \bmod p = l\}|$ for $l = 0, \ldots, p-1$. Then

$$\begin{array}{rrrrcrl}
d(0): & 0n_0+ & (p-1)n_1+ & (p-2)n_2+ & \cdots & + & 1n_{p-1} > n(p-1)/2 \\
d(1): & 1n_0+ & 0n_1+ & (p-1)n_2+ & \cdots & + & 2n_{p-1} > n(p-1)/2 \\
d(2): & 2n_0+ & 1n_1+ & 0n_2+ & \cdots & + & 3n_{p-1} > n(p-1)/2 \\
 & & \cdots & & & & \\
d(p-1): & (p-1)n_0+ & (p-2)n_1+ & (p-3)n_2+ & \cdots & + & 0n_{p-1} > n(p-1)/2
\end{array}$$

Summing the left hand sides yields

$$\sum_{l=0}^{p-1} l(n_0 + n_1 + \cdots + n_{p-1}) = np(p-1)/2$$

while the summation of the right hand sides yields also $np(p-1)/2$. Since a strict inequality must hold, it is a contradiction. Thus, Lemma 1 is true. □

By Lemma 1,

$$\min_{l\in L} z(\sigma^l(I)) - z(\sigma(I)) \le n(p-1)/2$$

Lemma 2 *$\sigma^l(I)$ is a feasible schedule for I^l.*

Proof of Lemma 2 The job assignment in $\sigma^l(I)$ is the same as the job assignment in $\sigma(I)$. Since each job in $\sigma(I)$ satisfies its release date constraint, the job in $\sigma^l(I)$ also satisfies its release date constraint in the problem instance I^l. Thus, $\sigma^l(I)$ is a feasible schedule for I^l. □

From Lemma 2 it follows that

$$z(\sigma(I^l)) \le z(\sigma^l(I))$$

From Lemma 1 and the above inequality, it follows that

$$\min_{l\in L}\left\{z(\sigma(I^l))\right\} - z(\sigma(I)) \le n(p-1)/2$$

From the above inequality, (15), and the fact that $z(\sigma'(I^l)) \leq z(\sigma(I^l))$, it follows that

$$z(\sigma^{IMR}) - z(\sigma(I)) \leq n(p-1)/2 \tag{16}$$

From (16), (11), (14), and the fact that $\sum_{j\in J} r_j \geq \sum_{j\in J'}(r_j \mod p)$ and $z(\sigma^{IMR}) \leq z(\sigma^{MR})$, it follows that

$$\begin{aligned}\frac{z(\sigma^{IMR})}{z(\sigma(I))} &\leq 1 + \frac{\min\left\{\frac{n(p-1)}{2},\ \sum_{j\in J'}(r_j \mod p) + \sum_{j\in J''} p\right\}}{np + \frac{1}{2}\left(\sum_{j\in J} r_j + \sum_{j\in J''} p\right)} \\ &\leq 1 + \frac{\min\left\{\frac{1}{2}np,\ \sum_{j\in J} r_j + \sum_{j\in J''} p\right\}}{np + \frac{1}{2}\left(\sum_{j\in J} r_j + \sum_{j\in J''} p\right)}\end{aligned}$$

Let $np = A$ and $\sum_{j\in J} r_j + \sum_{j\in J''} p = B$, then

$$\frac{z(\sigma^{IMR})}{z(\sigma(I))} \leq 1 + \frac{\min\{\frac{A}{2},\ B\}}{A + \frac{B}{2}} \leq 1 + \frac{\frac{4}{5}\cdot\frac{A}{2} + \frac{1}{5}\cdot B}{A + \frac{B}{2}} = \frac{7}{5}.$$

□

We consider $l_1 \in L\backslash\{r_j \mod p \mid j \in J\}$, then we can define $l_2 = \arg\min_l\{(l_1 - l) \mod p \mid l \in \{r_j \mod p \mid j \in J\}\}$. Let r'_j and r''_j denote the modified release dates of job j in I^{l_1} and I^{l_2}, respectively. Then $r''_j = r'_j - \Delta$ where $\Delta = (l_1 - l_2) \mod p$. Since problem instances I^{l_1} and I^{l_2} are the same except that the release dates of the jobs in I^{l_1} are constantly larger than the release dates of the jobs in I^{l_2}, $\sigma(I^{l_2})$ is the shifted schedule of $\sigma(I^{l_1})$. More precisely,

$$C_j\left(\sigma(I^{l_2})\right) = C_j\left(\sigma(I^{l_1})\right) - \Delta \ \ \forall j \in J$$

If we minimally move jobs up in both schedules $\sigma(I^{l_1})$ and $\sigma(I^{l_2})$, we can obtain the very same feasible schedule from both, and thus

$$\sum_{j\in J} C_j\left(\sigma'(I^{l_2})\right) = \sum_{j\in J} C_j\left(\sigma'(I^{l_1})\right).$$

Therefore, in Algorithm IMR, it suffices to choose $l \in L := \{r_j \mod p \mid j \in J\}$. Then $|L| \leq \min\{n, p\}$ and thus Algorithm IMR runs in polynomial time.

5 Experimental results

5.1 A practical algorithm

In Algorithm MR, we modified the release dates to make them integer multiples of p and solve the modified problems through assignment problem formulations. Through the modifications, we bypass the complexity coming from the release dates and focus on eligibility. However, as the worst case example of Theorem 1 in Sect. 4.1 shows, simplifying the release dates may lead to inefficient schedules because no distinction can be made between jobs that have the same modified release date. Algorithm IMR has to deal with that same issue as well. To overcome this issue in a practical manner, we propose in Sect. 5.1.1 a perturbation of cost coefficients used in the assignments for Algorithm MR and Algorithm IMR. In addition, we

consider in Sect. 5.1.2 a simple greedy algorithm that focuses more on release dates than on eligibility. By combining these algorithms, we propose in Sect. 5.1.3 a more practical algorithm, namely Algorithm GIMR (Generalized IMR).

5.1.1 A perturbation for assignment problem

We revise hereby only the objective of the assignment formulations for the two approximation algorithms that solve the modified instances I_L and I^l, since the constraints are the same as those in the original assignment problems.

Recall that decision variable x_{irj} is 1 if job j is scheduled on machine i with completion time λ_r. Thus, the objective function of the assignment problem is $\sum_{i\in M}\sum_{r=1}^{q}\sum_{j\in J}\lambda_r x_{irj}$. For the modified instance I^l with the modified release dates r_j, a candidate for completion time λ_r is defined as

$$\lambda_r \in \left\{ap + l \mid a = 1, \ldots, \max\left(\frac{r_j - l}{p}\right) + n\right\}.$$

For the modified instance I_L with the modified release dates r_j,

$$\lambda_r \in \left\{ap \mid a = 0, \ldots, \max\left(\frac{r_j}{p}\right) + n\right\}.$$

In the revised formulation, the objective in the assignment problem formulation is

$$\text{Minimize} \sum_{i\in M}\sum_{r=1}^{q}\sum_{j\in J} c_{irj}x_{irj}$$

where only the coefficients are different from those in the objective (1) of the MIP formulation in Sect. 3. The purpose of using cost c_{irj} rather than λ_r is to avoid any inefficiency emerging from the differences between the original instance I and the modified instances I_L and I^l, without affecting the original objective.

In the revised formulation, the primary objective of the assignment problem formulation is the same as the original objective, to minimize the total completion time. The second objective encourages jobs with earlier original release dates r_j^o to be scheduled earlier among those jobs with the same r_j in the modified instance. This perturbation is achieved by adding $\lambda_r\epsilon(r_j - r_j^o)$ to the assignment coefficient with a small positive ϵ. In this way we can avoid the inefficiency shown in the worst case example of Theorem 1. The third objective is to encourage jobs with a smaller eligible set size $|M_j|$ to be scheduled earlier among those jobs with the same r_j^o; this is achieved by adding $-\lambda_r\epsilon\frac{|M_j|}{m}$ to the assignment coefficient. We assume, without loss of generality, that every job is eligible to at least one machine. Thus $\frac{1}{m} \le \frac{|M_j|}{m} \le 1$.

Therefore, the assignment cost c_{irj} is defined as follows:

$$c_{irj} = \begin{cases} \infty & \text{if } \lambda_r < r_j + p \text{ or } i \notin M_j \\ \lambda_r\left(1 + \epsilon(r_j - r_j^o - \frac{|M_j|}{m})\right) & \text{otherwise} \end{cases}$$

where $\epsilon = \frac{1}{(p+1)(\lambda_q+1)}$.

Now, let the schedule generated by Algorithm IMR with the perturbation idea using c_{irj} be denoted by $\sigma^{IMR'}$.

Table 1 Information of jobs for the proof of Theorem 3

Group	Job index	M_j	r_j
0	$j = 1, \ldots, F_1$	$\{j, j + F_1\}$	0
1	$j = F_1 + 1, \ldots, F_2$	$\{j - 2F_1 + F_2, j - F_1\}$	1
$1'$	$j = 2^l + 1$	$\{2^{l-1} + 1, \ldots, 2^l\} \bigcup \{2^l + 1\}$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
g	$j = F_g + 1, \ldots, F_{g+1}$	$\{j - 2F_g + F_{g+1}, j - F_g\}$	g
g'	$j = 2^l + g$	$\{2^{l-g} + 1, \ldots, 2^l\} \bigcup \{2^l + g\}$	g
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$l - 1$	$j = F_{l-1} + 1 = 2^l - 1$	$\{1, 2\}$	$l - 1$
$(l - 1)'$	$j = 2^l + l - 1$	$\{2 + 1, \ldots, 2^l\} \bigcup \{2^l + l - 1\}$	$l - 1$
l	$j = 2^l$	$\{1\}$	l
l'	$j = 2^l + l$	$\{2, \ldots, 2^l\} \bigcup \{2^l + l\}$	l

5.1.2 Algorithm Greedy

A greedy algorithm is proposed to prioritize release dates more than eligibility by selecting jobs according to the Earliest Release Date (ERD) rule and using eligibility for tie breaking. The procedure is as follows:

Algorithm Greedy

1. Choose the job with the earliest release date among the jobs still to be scheduled. If a tie occurs, choose the job with the smallest eligible set size.
2. Choose the machine with the earliest available starting time among the eligible machines of the chosen job. If a tie occurs, choose the machine that has the smallest number of jobs still to be scheduled and eligible to that machine.
3. Schedule the chosen job to the chosen machine as early as possible.
4. Repeat 1–3 until all jobs are scheduled.

Algorithm Greedy is not as good as the proposed approximation algorithms such as IMR in a theoretical sense. The following theorem shows a lower bound of the worst case performance ratio of Algorithm Greedy.

Theorem 3 *The worst case performance ratio of Algorithm Greedy for* $P \mid M_j, r_j, p_j = p \mid \sum C_j$ *is at least 2.*

Proof Consider a problem instance of $2^l + l$ jobs and $2^l + l$ machines with the information of jobs given as in Table 1, where $F_x = \sum_{i=1}^{x} 2^{l-i}$.

Algorithm Greedy will choose jobs in the following group order.

$$\text{Group } 0 \to 1 \to 1' \to 2 \to 2' \to \cdots$$

According to Algorithm Greedy, the eligible machines of each job in Group g have the same priority because they have the same earliest available time and the same number of unscheduled jobs eligible to each machine. Thus, the schedule in Fig. 1a is possible by Algorithm Greedy. However, the optimal schedule is shown in Fig. 1b. □

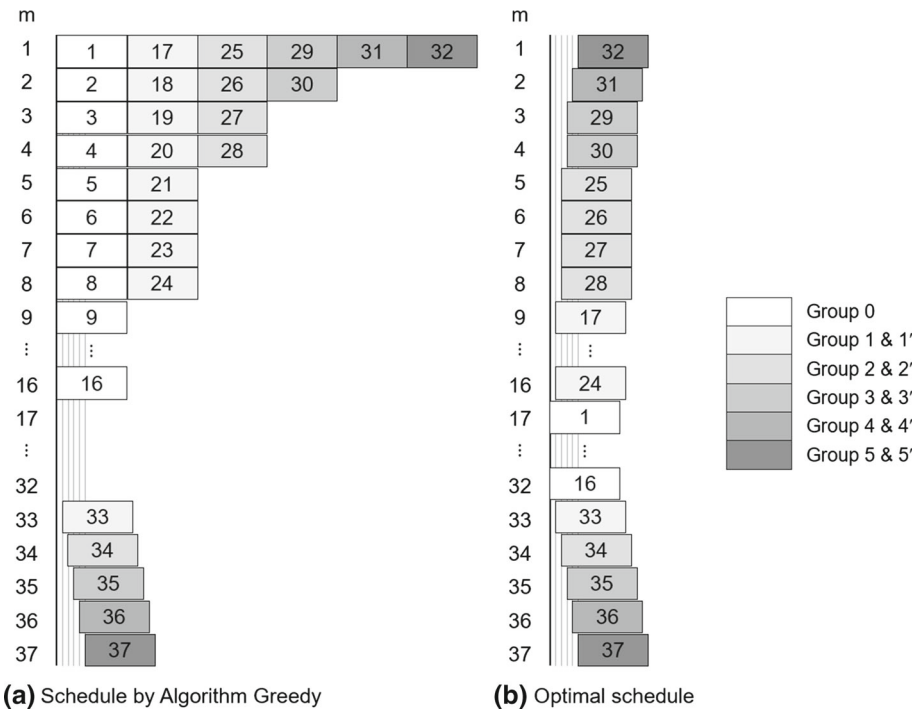

Fig. 1 The worst schedule by Algorithm Greedy and an optimal schedule when $l = 5$

Let σ^{Greedy} denote the schedule generated by Algorithm Greedy. The performance ratio of this problem instance is

$$\frac{z(\sigma^{Greedy})}{z(\sigma(I))} = \frac{\sum_{x=0}^{l-1}\left\{2^{l-x-1}(1+x)p\right\} + (l+1)p + \sum_{x=1}^{l}\{p+x\}}{\sum_{x=0}^{l-1}\left\{2^{l-x-1}(p+x)\right\} + p + l + \sum_{x=1}^{l}\{p+x\}}$$
$$= \frac{2^{l+1}p - p + lp + \frac{l(l+1)}{2}}{2^l p + 2^l + p + lp + \frac{l(l+1)}{2}}.$$

Therefore,

$$\lim_{l\to\infty}\lim_{p\to\infty}\frac{z(\sigma^{Greedy})}{z(\sigma(I))} = 2.$$

It completes the proof. □

5.1.3 Algorithm GIMR

For practical applications, we consider a combination of two algorithms. Algorithm IMR′ is similar to Algorithm IMR which consolidates release dates of the jobs and considers the eligibility with consolidated release dates, except for the perturbations of the assignment problem coefficients. Thus, it is expected that the schedule generated by the algorithm will be a reasonable one. On the other hand, Algorithm Greedy pays much more attention to release dates than to eligibility. Since the two algorithms may behave in a complementary manner,

Table 2 Values used for parameters

Parameters	Values
m	{2, 3, 4, 8, 16}
n/m	{2, 3, 4, 5, 6, 8}
p	{2, 3, 4, 8, 16}
dr	{0.25, 0.5, 0.75, 1.0}
dM	{0.4, 0.6, 0.8}

Table 3 Experiment results

Algorithm	IMR′	Greedy	GIMR
Average objective ratio	1.0061	1.0063	1.0015
Maximum objective ratio	1.0900	1.1864	1.0615

we design a practical algorithm, Algorithm Generalized IMR (GIMR), for the experiments as follows.

Algorithm GIMR
Choose between $\sigma^{IMR'}$ and σ^{Greedy} the better schedule.

5.2 Experimental settings

For the experiments, we randomly generate problem instances. Five parameters are chosen to investigate whether they have an impact on the performance of the proposed algorithms and are used in the random generation of problem instances: the number of machines (m), the ratio of the numbers of jobs to machines (n/m), the equal processing time of all jobs (p), the density of release dates (dr), and the density of eligibility(dM). The randomly generated release dates follow a discrete uniform distribution of $[0, dr \cdot \frac{np}{m})$. Note that $\frac{np}{m}$ is the optimal makespan when all jobs are eligible to all machines and all can start at time zero. With a small dr value, all release dates are close to zero, while with a large dr value, all release dates are spread out uniformly. For a randomly generated eligible set, let each machine have a probability to be included in the eligible set of each job with a probability of dM, while every job is eligible to at least one machine. Thus, $dM \approx E\left[\frac{|M_j|}{m}\right]$.

The values of the parameters for the experiments are as described in Table 2.

For each setting of the parameters, 20 problem instances are created, and the total number of generated instances in all experiments is 36,000.

The experiments are conducted on a desktop computer with OS of Fedora 28 server(Linux kernel 4.18.9), CPU of AMD Ryzen 5 2600X, and RAM of 16 GB 3200 MHz. The algorithms are implemented in Python version 3.6.6. The optimization solver Gurobi version 8.0.1 is utilized.

5.3 Results

The performance of the algorithm is measured by the objective ratio of $z(\sigma^A)$ to $z(\sigma(I))$ where σ^A is the schedule generated by Algorithm A. The value $z(\sigma(I))$ can be obtained by solving the MIP formulation in Sect. 3 allowing sufficient running time. An overview of the results is presented in Table 3.

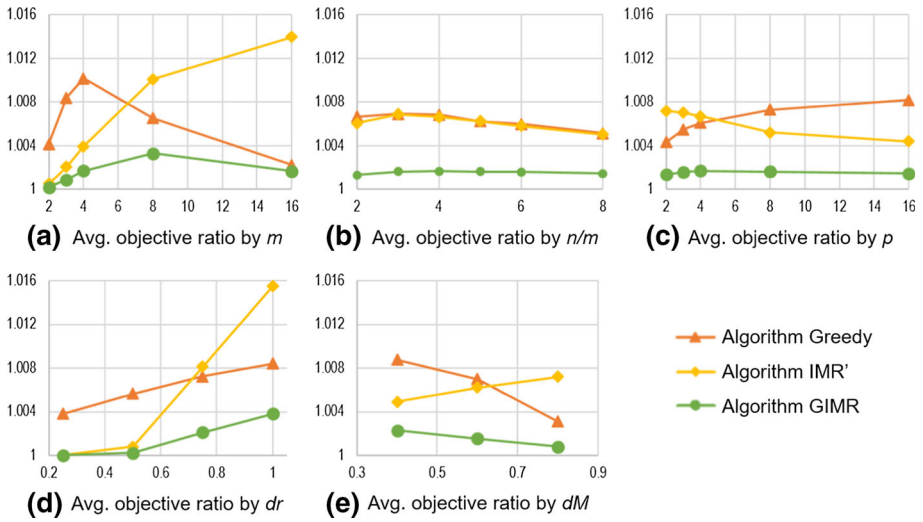

Fig. 2 Average objective ratio as a function of parameters

As for the average objective ratio, Algorithm IMR′ and Algorithm Greedy perform comparably. However, as for the maximum objective ratio, Algorithm IMR′ performs in a more stable manner. Interestingly, both the average and maximum objective ratios of Algorithm GIMR are much smaller than those of each algorithm, respectively.

5.3.1 Experiment result of GIMR

Figure 2a shows that Algorithm Greedy works relatively well for large m while Algorithm IMR′ works well for small m. Thus, when $m = 8$, the performance of Algorithm GIMR is at its worst. Figure 2b demonstrates that the difference between the performances of Algorithm IMR and Algorithm Greedy does not depend on the parameter n/m. However, choosing the better schedule based on the results of the two algorithms greatly decreases the average objective ratio. Figure 2c shows that the performance of Algorithm Greedy gets worse and that of Algorithm IMR′ gets better as p increases. However, the performance of Algorithm GIMR is quite stable and does not depend on p. Figure 2d shows that the average objective ratios of both algorithms increase as the value of parameter dr increases and the performance of Algorithm GIMR gets worse with a larger dr. Figure 2e shows the performance of Algorithm Greedy gets better and that of Algorithm IMR′ gets worse as dM increases. Overall, the performance of Algorithm GIMR gets better as dM increases. Granted, the problem with a larger dM is closer to the problem without eligibility constraints.

From Fig. 2, we can say that there are complementary relationships between Algorithm IMR′ and Algorithm Greedy with respect to all parameters even though each one of the parameters has a different impact on the performance of Algorithm GIMR.

5.3.2 GIMR vs. MIP

As we discussed, the average objective ratio of GIMR is 1.0015 and it implies that Algorithm GIMR generates an optimal solution or a near optimal solution. On the other hand,

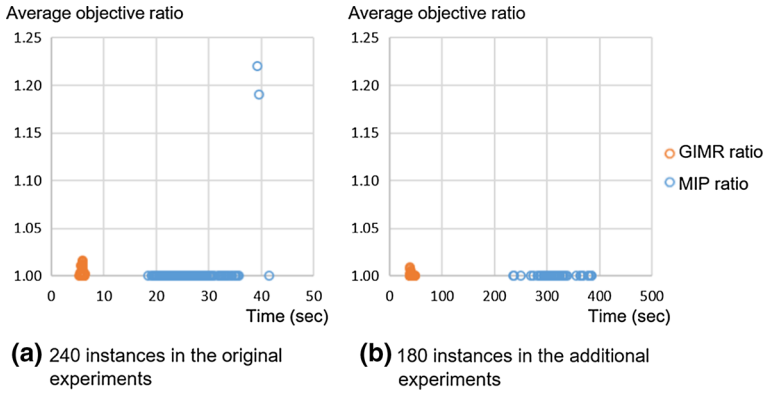

Fig. 3 Comparison of GIMR and MIP with respect to time and average objective ratio

in the experiments, the MIP formulation always optimally solves the problem with small size instances in a short time. However, when the problem size gets bigger, the required computation time of the MIP increases very rapidly. Thus, we now compare the performance measures and the computation times of the GIMR and the MIP with a timelimit.

We compare the results by GIMR and MIP for the largest 240 instances among the experiments conducted, with parameters $m = 8$, $n/m = 16$, $p = 16$, $dr \in \{0.25, 0.5, 0.75, 1.0\}$ and $dM \in \{0.4, 0.6, 0.8\}$. Figure 3a shows the objective ratios and the computation times of all 240 instances by Algorithm GIMR and the MIP. Algorithm GIMR takes 5.88 s on average with an average objective ratio of 1.0014. The running time is always less than 7 s and the objective ratio is no more than 1.02. Thus, GIMR seems to generate schedules in a robust manner. The MIP approach with a time limit of 30 s takes 25.24 s on average. The MIP finds optimal solutions for all but two of the 240 problem instances; the two instances take more than 30 s and have objective ratios greater than 1.19. Because of these bad objective ratios, the average objective ratio of the MIP approach with the time limit is 1.0017. Moreover, in spite of the 30 s time limit, the MIP finds optimal solutions for 29 instances in more than 30 s. Thus, Algorithm GIMR seems to be more robust with respect to the performance and the computation time.

In order to investigate this pattern more thoroughly, we conducted additional experiments with larger parameter settings. We generated 180 instances with parameters $m = 8$, $n/m = 32$, $p = 32$, $dr \in \{0.25, 0.5, 0.75\}$ and $dM \in \{0.4, 0.6, 0.8\}$. The current computational environment is not able to solve instances with $dr = 1$, because of a lack of memory; these instances are not considered. Figure 3b shows the objective ratios of all 180 instances by Algorithm GIMR and the MIP. Algorithm GIMR takes 41.4 s on average with an average objective ratio of 1.0004. The running time is always less than 60 s and the objective ratio is less than 1.01. The MIP is applied with a time limit of 300 s and it cannot find feasible solutions for 122 of the 180 instances. Thus, the instances with no solutions do not appear in Fig. 3b. The MIP approach finds feasible solutions only for 58(32.2%) instances, and all solutions found are optimal. The experiments with the larger problem instances also show that Algorithm GIMR works in a more robust manner.

6 Discussion

We consider three generalizations of the approaches described in the previous sections, namely the DP in Sect. 2, the MIP formulation in Sect. 3, and Algorithm IMR in Sect. 4.

6.1 Generalization of DP

The DP algorithm in Sect. 2 can be applied to a generalization of the problem with uniform machines, denoted by $Qm \mid M_j, r_j, p_j = p \mid \sum C_j$ after only a minor modification. In a uniform machine environment, machine i has a positive speed s_i and the processing requirement of each job on machine i is p/s_i for $i \in M$.

First, Proposition 2 can be modified for problem $Qm \mid M_j, r_j, p_j = p \mid \sum C_j$.

Proposition 3 *There exists an optimal schedule in which the completion time of each job scheduled on machine i is of the form $r_j + a \cdot p/s_i$, for some $j \in J$ and $a \in \{1, \ldots, n\}$.*

From Proposition 3, we define the following set of the candidates for the completion times of the jobs scheduled on machine i.

$$\Lambda_i = \left\{ t \mid t = r_j + a \cdot p/s_i, \quad \text{for } j \in J, \quad a \in \{1, \ldots, n\} \right\}$$

For a dynamic programming algorithm, we consider a partial schedule with the first b^e jobs from J^e for $e = 1, \ldots, k$ in which the latest completion time of machine i is t_i for $t_i \in \Lambda_i$ and $i = 1, \ldots, m$. The state of machine i, $\alpha_i := (t_i; b_i^1, b_i^2, \ldots, b_i^e, \ldots, b_i^k)$, is defined as before except that $t_i \in \Lambda$ is changed to $t_i \in \Lambda_i$. The state of a partial schedule is defined as a collection of states of all the machines as before.

The Boundary Conditions and the Optimal Value Function remain the same. The Recursive Relationship is changed to

$$V(\alpha) = \min \left\{ V\left(\hat{\alpha}(h, f)\right) + t_h \;\middle|\; f \in \{1, \ldots, k\}, h \in M^f, r^f(b^f) \le t_h - p/s_h, t_h \in \Lambda_h \right\}$$

along with the change of the definition of $\hat{\alpha}_h$. The condition on $\hat{t}_h$ in $\hat{\alpha}_h$ is also changed to

$$\hat{t}_h \le t_h - p/s_h \quad \text{and} \quad \hat{t}_h \in \Lambda_h.$$

The time complexity is the same as the one of $Pm \mid M_j, r_j, p_j = p \mid \sum C_j$. Thus, for fixed m, the running time of the modified DP algorithm is still polynomial. Therefore, $Qm \mid M_j, r_j, p_j = p \mid \sum C_j$ can be solved in polynomial time.

6.2 Generalization of the MIP formulation

The MIP formulation proposed in Sect. 3 can be extended to the problem of minimizing other objectives with job j having a nonnegative weight w_j and a due date d_j.

For the problem of minimizing the total weighted tardiness, the objective becomes

$$\text{Minimize} \quad \sum_{i \in M} \sum_{r=1}^{q} \sum_{j \in J} w_j \max\{\lambda_r - d_j, 0\} x_{irj}$$

while all the constraints remain the same. Since $w_j \max\{\lambda_r - d_j, 0\}$ can be computed in advance for given $r \in \{1, \ldots, q\}$ and $j \in J$, it does not increase the problem size.

For the problem minimizing the total weighted number of tardy jobs, the objective becomes

$$\text{Minimize} \quad \sum_{i\in M}\sum_{r=1}^{q}\sum_{j\in J} w_{rj}x_{irj}$$

where

$$w_{rj} = \begin{cases} w_j & \text{if } \lambda_r > d_j \\ 0 & \text{otherwise.} \end{cases}$$

6.3 Generalization of algorithm IMR

Algorithm IMR proposed in Sect. 4.2 can be applied to the problem of minimizing the total weighted completion time, denoted by $P \mid M_j, r_j, p_j = p \mid \sum w_j C_j$. We can define I_L and I^l for $l \in L$ from I in the same way. Since $P \mid M_j, r_j, p_j = 1 \mid \sum w_j C_j$ can be solved optimally in polynomial time (see Brucker et al. 1997), the problem of minimizing the weighted completion time with problem instances I_L and I^l for $l \in L$ can be solved in polynomial time as well.

The following notation is needed for the more general algorithm and its analysis.

$$\begin{aligned} J_l &:= \{j \in J \mid C_j(\sigma(I)) \mod p = l\} \quad \text{for} \quad l \in L \\ z_w(\sigma) &:= \sum_{j\in J} w_j C_j(\sigma) \\ d_w(l) &:= z_w(\sigma^l(I)) - z_w(\sigma(I)) \\ W_l &:= \sum_{j\in J_l} w_j \\ W &:= \sum_{l=0}^{p-1} W_l = \sum_{j\in J} w_j \end{aligned}$$

The previous inequalities for the problem of minimizing the total completion time are modified as follows. By modifying (11) for the problem, we have

$$z_w(\sigma^{MR}) - z_w(\sigma(I)) \le \sum_{j\in J'} w_j(r_j \mod p) + \sum_{j\in J''} w_j p. \tag{17}$$

By modifying (14), we have

$$z_w(\sigma(I)) \ge Wp + \frac{1}{2}\left(\sum_{j\in J} w_j r_j + \sum_{j\in J''} w_j p\right) \tag{18}$$

In order to modify (16), we first need to modify Lemma 1 as follows.

Lemma 3

$$\min_{l\in L} d_w(l) \le W(p-1)/2$$

Proof of Lemma 3 Suppose Lemma 3 is not true, then $d_w(l) > W(p-1)/2$ for all $l \in L$. Then

$$\begin{aligned}
d_w(0): &\quad 0W_0+ &(p-1)W_1+ &(p-2)W_2+ &\cdots &+ &1W_{p-1} > W(p-1)/2\\
d_w(1): &\quad 1W_0+ &0W_1+ &(p-1)W_2+ &\cdots &+ &2W_{p-1} > W(p-1)/2\\
d_w(2): &\quad 2W_0+ &1W_1+ &0W_2+ &\cdots &+ &3W_{p-1} > W(p-1)/2\\
&&\cdots\\
d_w(p-1): &\quad (p-1)W_0+ &(p-2)W_1+ &(p-3)W_2+ &\cdots &+ &0W_{p-1} > W(p-1)/2
\end{aligned}$$

The summation of each side has the same value of $Wp(p-1)/2$ and it is a contradiction since a strict inequality must hold. Thus, the lemma is true. □

Using a similar argument as the one used to derive (16) and Lemma 3, we have

$$z_w(\sigma^{IMR}) - z_w(\sigma(I)) \le W(p-1)/2 \tag{19}$$

By (17), (18), (19) and the fact that $\sum_{j\in J} w_j r_j \ge \sum_{j\in J'} w_j (r_j \mod p)$ and $z_w(\sigma^{IMR}) \le z_w(\sigma^{MR})$,

$$\begin{aligned}
\frac{z_w(\sigma^{IMR})}{z_w(\sigma(I))} &\le 1 + \frac{\min\left\{W\frac{(p-1)}{2},\ \sum_{j\in J'} w_j (r_j \mod p) + \sum_{j\in J''} w_j p\right\}}{Wp + \frac{1}{2}\left(\sum_{j\in J} w_j r_j + \sum_{j\in J''} w_j p\right)}\\
&\le 1 + \frac{\min\left\{\frac{1}{2}Wp,\ \sum_{j\in J} w_j r_j + \sum_{j\in J''} w_j p\right\}}{Wp + \frac{1}{2}\left(\sum_{j\in J} w_j r_j + \sum_{j\in J''} w_j p\right)}.
\end{aligned}$$

If $A = Wp$ and $B = \sum_{j\in J} w_j r_j + \sum_{j\in J''} w_j p$, then

$$\frac{z_w(\sigma^{IMR})}{z_w(\sigma(I))} \le 1 + \frac{\min\{\frac{A}{2},\ B\}}{A + \frac{B}{2}} \le 1 + \frac{\frac{4}{5}\cdot\frac{A}{2} + \frac{1}{5}\cdot B}{A + \frac{B}{2}} = \frac{7}{5}.$$

Note that there are some known approximation algorithms for $R \mid r_j \mid w_j C_j$ which is more general than $P \mid M_j, r_j, p_j = p \mid w_j C_j$. $Rm \mid r_j \mid w_j C_j$ admits a Polynomial Time Approximation Scheme (PTAS) (Afrati et al. 1999) and $R \mid r_j \mid w_j C_j$ has a 2-approximation algorithm that uses a convex quadratic programming approach (Skutella 2001). Recently, Im and Li (2017) proposed a 1.8786-approximation algorithm based on linear programming and a rounding technique. Whereas those algorithms are complicated to implement, the proposed algorithm for $P \mid M_j, r_j, p_j = p \mid \sum w_j C_j$ is much easier to understand and to implement.

7 Conclusion

We consider a problem of scheduling jobs with equal processing times and different release dates subject to eligibility constraints to minimize the total completion time. Although the computational complexity of this problem remains open, a polynomial time DP algorithm is developed for the problem with a fixed number of machines. Two polynomial time approximation algorithms are presented along with their worst case analyses. The experiments with randomly generated instances show that the proposed algorithm performs very well in practice.

As a future research direction, it would be of interest to show whether or not the problem under consideration and the related problems are polynomial time solvable.

Acknowledgements This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government [grant number NRF-2017R1A2B4011486].

References

Afrati, F., Bampis, E., Chekuri, C., Karger, D., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., et al. (1999) Approximation schemes for minimizing average weighted completion time with release dates. In *40th annual symposium on foundations of computer science*, Piscataway: IEEE (pp. 32–43).

Baptiste, P., Brucker, P., Knust, S., & Timkovsky, V. G. (2004). Ten notes on equal-processing-time scheduling. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, *2*(2), 111–127.

Brucker, P., Jurisch, B., & Krämer, A. (1997). Complexity of scheduling problems with multi-purpose machines. *Annals of Operations Research*, *70*, 57–73.

Brucker, P., & Kravchenko, S. A. (2005). *Scheduling jobs with release times on parallel machines to minimize total tardiness*. Fachbereich Mathematik/Informatik: Universität Osnabrück.

Brucker, P., & Kravchenko, S. A. (2008). Scheduling jobs with equal processing times and time windows on identical parallel machines. *Journal of Scheduling*, *11*(4), 229–237.

Bruno, J., Coffman, E. G, Jr., & Sethi, R. (1974). Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, *17*(7), 382–387.

Glass, C. A., & Mills, H. (2006). Scheduling unit length jobs with parallel nested machine processing set restrictions. *Computers & Operations Research*, *33*(3), 620–638.

Hochbaum, D. S. (1996). *Approximation algorithms for NP-hard problems*. Berlin: PWS Publishing Co.

Horn, W. (1973). Minimizing average flow time with parallel machines. *Operations Research*, *21*(3), 846–847.

Hwang, H. C., Chang, S. Y., & Lee, K. (2004). Parallel machine scheduling under a grade of service provision. *Computers & Operations Research*, *31*(12), 2055–2061.

Im, S., & Li, S. (2017). Better unrelated machine scheduling for weighted completion time via random sets from non-uniform distributions. In *Proceedings of the 13th workshop on models and algorithms for planning and scheduling problems (MAPSP)*

Keha, A. B., Khowala, K., & Fowler, J. W. (2009). Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*, *56*(1), 357–367.

Kravchenko, S. A., & Werner, F. (2011). Parallel machine problems with equal processing times: a survey. *Journal of Scheduling*, *14*(5), 435–444.

Lee, K., Leung, J. Y. T., & Pinedo, M. L. (2011). Scheduling jobs with equal processing times subject to machine eligibility constraints. *Journal of Scheduling*, *14*(1), 27–38.

Lenstra, J. K., Kan, A. R., & Brucker, P. (1977). *Complexity of machine scheduling problems. Annals of Discrete Mathematics* (Vol. 1, pp. 343–362). Amsterdam: Elsevier.

Leung, J. Y. T., & Li, C. L. (2008). Scheduling with processing set restrictions: A survey. *International Journal of Production Economics*, *116*(2), 251–262.

Li, C. L. (2006). Scheduling unit-length jobs with machine eligibility restrictions. *European Journal of Operational Research*, *174*(2), 1325–1328.

Li, S. (2017). Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. In *IEEE 58th annual symposium on foundations of computer science (FOCS)*.

Pinedo, M. L. (2016). *Scheduling: Theory, algorithms, and systems*. Berlin: Springer.

Simons, B. (1983). Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Journal on Computing*, *12*(2), 294–299.

Skutella, M. (2001). Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of the ACM (JACM)*, *48*(2), 206–242.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.