Journal of Combinatorial Optimization (2023) 45:98
https://doi.org/10.1007/s10878-023-01028-3

Exact algorithms for solving the constrained parallel-machine scheduling problems with divisible processing times and penalties

Jianping Li[1] · Runtao Xie[1] · Junran Lichen[2] · Guojun Hu[1] · Pengxiang Pan[1] · Ping Yang[1]

Accepted: 30 March 2023 / Published online: 19 April 2023

Abstract
In this paper, we address the constrained parallel-machine scheduling problem with divisible processing times and penalties (the CPS-DTP problem), which is a further generalization of the parallel-machine scheduling problem with divisible processing times (the PS-DT problem). Concretely, given a set M of m identical machines and a set J of n independent jobs, each job has a processing time and a penalty, the processing times of these n jobs are divisible, and we implement these n jobs under the requirement that each job in J must be either continuously executed on one machine with its processing time, or rejected with its penalty that we must pay for. We may consider three versions of the CPS-DTP problem, respectively. (1) The constrained parallel-machine scheduling problem with divisible processing times and total penalties (the CPS-DTTP problem) is asked to find a subset A of J and a schedule T for jobs in A to satisfy the aforementioned requirement, the objective is to minimize the makespan

✉ Jianping Li
jianping@ynu.edu.cn

Runtao Xie
xieruntao7@163.com

Junran Lichen
J.R.Lichen@buct.edu.cn

Guojun Hu
huguojun@mail.ynu.edu.cn

Pengxiang Pan
pengxiang@ynu.edu.cn

Ping Yang
1573395725@qq.com

[1] School of Mathematics and Statistics, Yunnan University, East Outer Ring South Road, University Town, Chenggong District, Kunming 650504, People's Republic of China

[2] School of Mathematics and Physics, Beijing University of Chemical Technology, No.15, North Third Ring East Road, Chaoyang District, Beijing 100029, People's Republic of China

of such a schedule T for jobs in A plus the summation of penalties paid for jobs not in A; (2) The constrained parallel-machine scheduling problem with divisible processing times and maximum penalty (the CPS-DTMP problem) is asked to find a subset A of J and a schedule T for jobs in A to satisfy the aforementioned requirement, the objective is to minimize the makespan of such a schedule T for jobs in A plus maximum penalty paid for jobs not in A; (3) The constrained parallel-machine scheduling problem with divisible processing times and bounded penalty (the CPS-DTBP problem) is asked to find a subset A of J and a schedule T for jobs in A to satisfy the aforementioned requirement and the summation of penalties paid for jobs not in A is no more than a fixed bound, the objective is to minimize the makespan of such a schedule T for jobs in A. As our main contributions, we design three exact algorithms to solve the CPS-DTTP problem, the CPS-DTMP problem and the CPS-DTBP problem, and these three algorithms run in time $O((n\log n+nm)C)$, $O(n^2\log n)$ and $O((n\log n+nm)\log C)$, respectively, where C is the optimal value of same instance for the PS-DT problem.

Keywords Combinatorial optimization · Constrained parallel-machine scheduling · Divisible processing times · Penalties · Exact algorithms

1 Introduction

The classical scheduling problem (Graham 1966; Lenstra et al. 1977) is one of fundamental and well-studied problems in combinatorial optimization, and it is modelled as follows. Given a set $M = \{a_1, a_2, \ldots, a_m\}$ of m identical machines and a set $J = \{b_1, b_2, \ldots, b_n\}$ of n independent jobs, where each job $b_j \in J$ has a processing time, it is asked to schedule these n jobs in J on machines in M such that each job $b_j \in J$ must be continuously executed only on one machine in M, the objective is to minimize the makespan, i.e., the total timespan required to execute the jobs in J. This scheduling problem has been widely and deeply considered in the literature, and it has a wide range of applications in various domains, such as business management, computer systems, transportation, aerospace, and medical and health. We use a notation $P||C_{\max}$ (Graham 1966; Lawler et al. 1993) to denote this scheduling problem.

The $P\,||\,C_{\max}$ problem is a strongly *NP*-hard problem (Graham 1966; Lenstra et al. 1977), and we have known that there are many approximation algorithms to solve this problem. Graham (1966) designed the list scheduling algorithm (the LS algorithm, for short) for solving the $P\,||\,C_{\max}$ problem, whose strategy is to arrange these jobs on machines and assign a job, which is then executed, on the fastest machine that is currently nearing completion, and this LS algorithm has an approximation ratio $2-\frac{1}{m}$. When the number of machines is either two or three, i.e., either $m = 2$ or $m = 3$, Faigle et al. (1989) proved that the LS algorithm (Graham 1966) is also an exact algorithm for solving the on-line version of the $P\,||\,C_{\max}$ problem in polynomial time $O(n\log m)$ that no improvement is possible. On the other hand, Graham (1969) presented another longest processing time algorithm (the LPT algorithm, for short) for solving the $P\,||\,C_{\max}$ problem, whose strategy is first to sort all the jobs in non-increasing order based on their processing times and secondly to execute all the jobs on the machines, and assigns the next job to be executed on the fastest machine that is

currently nearing completion. The LPT algorithm has an approximate ratio $\frac{4}{3} - \frac{1}{3m}$, and this LPT algorithm is significantly better than the LS algorithm. Having based on the correspondence between the bin packing problem and this problem, Coffman et al. (1978) presented the MULTIFIT approximation algorithm for solving the $P \mid\mid C_{\max}$ problem, and the same authors (Coffman et al. 1978) showed that the upper bound of the MULTIFIT algorithm does not exceed 1.22 in the worst case. Hochbaum and Shmoys (1987) proposed a first polynomial-time approximation scheme (PTAS, for short) for the $P \mid\mid C_{\max}$ problem, and this PTAS runs in time $O\left((\frac{n}{\epsilon})^{(\frac{1}{\epsilon^2})}\right)$ for each real number $\epsilon > 0$.

Nevertheless, when some processing times of jobs in a schedule are too longer than that of many practical situations, it may lead to need higher cost to execute all jobs on machines. In order to overcome this situation of excessive cost for all jobs, some researchers intend to give up a small amount of jobs, which have longer processing times, to bear smaller penalty costs (as a numerical value) instead of such "longer" jobs executed. Bartal et al. (2000) first proposed the parallel-machine scheduling problem with rejection penalties (the PS-P problem, for short), which is a generalization of the $P \mid\mid C_{\max}$ problem, and this new problem is modelled as follows. Given a set $M = \{a_1, a_2, \ldots, a_m\}$ of m identical machines and a set $J = \{b_1, b_2, \ldots, b_n\}$ of n independent jobs, each job $b_j \in J$ has a processing time p_j and a penalty e_j, and we implement these n jobs under the requirement that each job in J must be either scheduled only on one machine once to be executed without interruption with its processing time, or rejected with its penalty that we must pay for. We are asked to find a subset $A \subseteq J$ and a schedule T for jobs in A such that each job $b_j \in A$ has to be executed continuously on a machine and that each job $b_j \in J \backslash A$ has to be rejected, the objective is to minimize the makespan of such a schedule T for jobs in A plus the summation of penalties paid for jobs not in A, i.e., $\min\{C_{\max}(A, T) + \sum_{b_j \in J \backslash A} e_j \mid A \subseteq J$ and T is a feasible schedule for jobs in $A\}$, where $C_{\max}(A, T)$ is the makespan of the schedule T for jobs in A, and we call such a schedule T for jobs in A to be feasible if each job in A scheduled only on one machine once to be executed without interruption with its processing time. Motivated by the notation $P||C_{\max}$, we use a notation $P \mid rej \mid C_{\max}(A, T) + \sum_{b_j \in J \backslash A} e_j$ to denote the PS-P problem.

For the PS-P problem, Bartal et al. (2000) designed a fully polynomial-time approximation scheme (FPTAS, for short) for the fixed integer m and a PTAS for the arbitrary integer m, respectively. In addition, using the strategy to reject the jobs, each of whose penalty is not greater than the value of its processing time divided by the number of machines, and executing the LS algorithm (Graham 1966) on all other jobs with the shortest processing time, and then choosing the best one among all solutions, the same authors (Bartal et al. 2000) provided a $\left(2 - \frac{1}{m}\right)$-approximation algorithm for solving the PS-P problem, and for the arbitrary integer m, this algorithm runs in time $O(n \log n)$. On the other hand, He and Min (2000) presented the best possible on-line algorithms for solving the on-line version of the PS-P problem on two or three machines when the value of speed ratio is certain, and the same authors (He and Min 2000) showed the fact that this algorithm is optimal when the number of machines is two and the speed ratio $s \geq (\sqrt{5} + 1)/2$, or the number of machines is three and the speed ratio $s \geq 2$. Furthermore, presenting an up-to-date survey of such results in this

field, Shabtay et al. (2013) presented a survey to offer a unified framework for offline scheduling with rejection, and they highlighted the close connection between scheduling with rejection and other fields of research such as scheduling with controllable processing times and scheduling with due date assignments.

Zhang et al. (2009) considered the parallel-machine scheduling problem with bounded penalty (the PS-BP problem, for short), which is another generalization of the PS-P problem, and it is modelled as follows. Given a set $M = \{a_1, a_2, \ldots, a_m\}$ of m identical machines, and a set $J = \{b_1, b_2, \ldots, b_n\}$ of n independent jobs, each job $b_j \in J$ has a processing time p_j and a penalty e_j. Whenever a job in J is operated in a schedule, this job must be either scheduled only on one machine once to be executed without interruption, or rejected with its penalty that we must pay for. We are asked to find a subset $A \subseteq J$ and a schedule T for jobs in A such that each job $b_j \in A$ has to be continuously executed on one machine and each job $b_j \in J \backslash A$ has to be rejected, where the summation of penalties paid for jobs in $J \backslash A$ is no more than B, i.e., $\sum_{b_j \in J \backslash A} e_j \leq B$, the objective is to minimize the makespan of a schedule T for jobs in A. Motivated by the notation $P||C_{\max}$, we use a notation $P \mid \sum_{b_j \in J \backslash A} e_j \leq B \mid C_{\max}(A, T)$ to denote the PS-BP problem.

When the number of machines in M is fixed, Zhang et al. (2009) presented a pseudo-polynomial-time dynamic programming algorithm and an FPTAS for solving the PS-BP problem, respectively. Li et al. (2015) presented a 2-approximation algorithm in strongly polynomial time and a PTAS for solving the PS-BP problem, and for the case where m is a fixed constant, the same authors (Li et al. 2015) designed an FPTAS for solving the PS-BP problem, which improved previous best running time from $O\left(\frac{n^{m+2}}{\epsilon^m}\right)$ (Zhang et al. 2009) to $O\left(\frac{1}{\epsilon^{2m+3}} + mn^2\right)$.

We have a heavy impression on the fact that Coffman et al. (1978) first considered the bin packing problem with divisible item sizes (the BP-DS problem, for short), and divisible item sizes are of interest because they arise naturally in certain applications, such as memory allocation in computer systems, where device capacities and block sizes are commonly restricted to powers of 2 (Knuth 1997). The same authors (Coffman et al. 1978) showed that the first fit decreasing algorithm (the FFD algorithm) (Simchi-Levi 1994) optimally solves the BP-DS problem in time $O(n \log n)$, and they presented other exact algorithms in polynomial time to solve related bin packing problems with divisible item sizes. One interesting fact is that Coffman et al. (1987) simultaneously considered in the same paper the parallel-machine scheduling problem with divisible processing times (the PS-DT problem, for short), which is modelled as follows. Given a set $M = \{a_1, a_2, \ldots, a_m\}$ of m identical machines and a set $J = \{b_1, b_2, \ldots, b_n\}$ of n independent jobs, each job $b_j \in J$ has a processing time p_j, where processing times of jobs in J are divisible, i.e., either $p_i \mid p_j$ or $p_j \mid p_i$, for any two distinct jobs b_i and b_j in J, we are asked to find a schedule T for jobs in J such that each job $b_j \in J$ has to be executed without interruption only on one machine, the objective is to minimize the makespan of such a schedule T for jobs in J. And Coffman et al. (1987) showed that the LPT algorithm, which is designed by Graham (1969) for solving the $P \mid\mid C_{\max}$ problem, also optimally solves the PS-DT problem, whose complexity is still the time $O(n \log n)$. Motivated by the notation $P||C_{\max}$, we use a notation $P \mid (p_i, p_j) = \min\{p_i, p_j\} \mid C_{\max}$ to denote the PS-DT problem.

Zheng et al. (2018) considered the parallel-machine scheduling problem with penalties under constraints (the PS-PC problem, for short), which is modelled as follows. Given a set $M = \{a_1, a_2, \ldots, a_m\}$ of m identical machines, and a set $J = \{b_1, b_2, \ldots, b_n\}$ of n independent jobs, each job $b_j \in J$ has a processing time p_j and a penalty e_j, where processing times of jobs are divisible, i.e., either $p_i \mid p_j$ or $p_j \mid p_i$, and in addition, we have $e_i/p_i \geq e_j/p_j$ whenever $p_i \geq p_j$, it is asked for us to find a subset $A \subseteq J$ and a schedule T for jobs in A such that each job $b_j \in A$ is continuously scheduled only on one machine and that each job $b_j \in J\backslash A$ is rejected, the objective is to minimize the makespan of such a schedule T for jobs in A plus the summation of penalties paid for jobs not in A, i.e., $\min\{C_{\max}(A, T) + \sum_{b_j \in J\backslash A} e_j \mid A \subseteq J$ and T is a feasible schedule for jobs in $A\}$. Zheng et al. (2018) designed an exact algorithm to solve the PS-PC problem in time $O(n^2 \log n)$. Motivated by the notation $P||C_{\max}$, we use a notation $P \mid (p_i, p_j) = \min\{p_i, p_j\}, e_i/p_i \geq e_j/p_j \mid C_{\max}(A, T) + \sum_{b_j \in J\backslash A} e_j$ to denote the PS-PC problem.

Motivated by parallel-machine scheduling, divisible processing times, penalties and other related aforementioned problems, we address the constrained parallel-machine scheduling problem with divisible processing times and penalties (the CPS-DTP problem, for short), which is a further generalization of the PS-DT problem and other related problems. Concretely, given a set $M = \{a_1, a_2, \ldots, a_m\}$ of m identical machines and a set $J = \{b_1, b_2, \ldots, b_n\}$ of n independent jobs, each job $b_j \in J$ has a processing time $p_j \in \mathbb{Z}^+$ and a penalty $e_j \in \mathbb{R}^+$, where processing times of jobs are divisible, i.e., either $p_i \mid p_j$ or $p_j \mid p_i$, for each pair b_i, b_j in J, it is asked for us to find a subset $A \subseteq J$ and a schedule T for jobs in A to satisfy the requirement that each job $b_j \in A$ is scheduled only on one machine and each job $b_j \in J\backslash A$ is rejected. We consider the following three versions of the CPS-DTP problem, having three different objectives, respectively, i.e.,

(1) The constrained parallel-machine scheduling problem with divisible processing times and total penalties (the CPS-DTTP problem, for short) is asked to find a subset $A \subseteq J$ and a schedule T for jobs in A to satisfy the aforementioned requirement, the objective is to minimize the makespan of such a schedule T for jobs in A plus the summation of penalties paid for jobs not in A, i.e., $\min\{C_{\max}(A, T) + \sum_{b_j \in J\backslash A} e_j \mid A \subseteq J$ and T is a feasible schedule for jobs in $A\}$.
(2) The constrained parallel-machine scheduling problem with divisible processing times and maximum penalty (the CPS-DTMP problem, for short) is asked to find a subset $A \subseteq J$ and a schedule T for jobs in A to satisfy the aforementioned requirement, the objective is to minimize the makespan of such a schedule for jobs in A plus the maximum penalty paid for jobs not in A, i.e., $\min\{C_{\max}(A, T) + e_{\max}(J\backslash A) \mid A \subseteq J$ and T is a feasible schedule for jobs in $A\}$, where $e_{\max}(J\backslash A) = \max\{e_j \mid b_j \in J\backslash A\}$.
(3) The constrained parallel-machine scheduling problem with divisible processing times and bounded penalty (the CPS-DTBP problem, for short) is asked to find a subset $A \subseteq J$ and a schedule T for jobs in A to satisfy the aforementioned requirement and the summation of penalties paid for jobs not in A is no more than

the bound B, i.e., $\sum_{b_j \in J \setminus A} e_j \leq B$, the objective is to minimize the makespan of such a schedule for jobs in A, i.e., $\min\{C_{\max}(A, T) \mid A \subseteq J$ and T is a feasible schedule for jobs in $A\}$.

For convenience as some aforementioned notations, we may denote the CPS-DTTP problem by $P \mid (p_i, p_j) = \min\{p_i, p_j\}, rej \mid C_{\max}(A, T) + \sum_{b_j \in J \setminus A} e_j$, the CPS-DTMP problem by $P \mid (p_i, p_j) = \min\{p_i, p_j\}, rej \mid C_{\max}(A, T) + \max\{e_j \mid b_j \in J \setminus A\}$ and the CPS-DTBP problem by $P \mid (p_i, p_j) = \min\{p_i, p_j\}, \sum_{b_j \in J \setminus A} e_j \leq B \mid C_{\max}(A, T)$, respectively.

To the best of our knowledge, these three aforementioned problems have not been considered in the literature, and we hope that there are more important applications for these three problems in theories and in practices. We intend to design an exact algorithm to solve the CPS-DTTP problem in pseudo-polynomial time, an exact algorithm to solve the CPS-DTMP problem in strongly polynomial time and an exact algorithm to solve the CPS-DTBP problem in polynomial time, respectively.

The remainder of this paper is organized into the following sections. In Sect. 2, we provide some terminologies, notations and fundamental lemmas to ensure the correctness of our exact algorithms. In Sect. 3, we design an exact algorithm to optimally solve the CPS-DTTP problem, and this algorithm runs in time $O((n \log n + nm)C)$, where C is the optimal value of the same instance for the PS-DT problem. In Sect. 4, we provide an exact algorithms to optimally solve the CPS-DTMP problem, and that algorithm runs in time $O(n^2 \log n)$. In Sect. 5, we present an exact algorithms to optimally solve the CPS-DTBP problem, and that algorithm runs in time $O((n \log n + nm) \log C)$, where C is defined as mentioned above. In Sect. 6, we summarize our conclusions and put forward some interesting related problems that will be worth to consider in the future research.

2 Terminologies and key lemmas

In this section, we present some terminologies, notations and fundamental lemmas in order to show our exact algorithms to optimally solve the CPS-DTTP problem, the CPS-DTMP problem and the CPS-DTBP, respectively. In addition, according to the requirements of computer science for input data, we may assume that the weight functions are all positive integer functions. On the other hand, readers can find other necessary materials in these references (Bernhard and Vygen 2008; Pinedo 2012; Potts and Strusevich 2009; Schrijver 2003).

In order to present exact algorithms to optimally solve our three problems, for convenience, we need to describe the parallel-machine scheduling problem with divisible processing times (the PS-DT problem, for short) and the multiple knapsack problem with divisible item sizes (the MKP-DS problem, for short), where the first is a special version of the $P \mid\mid C_{\max}$ problem (Graham 1966) and the second is a special version of the multiple knapsack problem (Kellerer et al. 2004), respectively.

From now on, we present some following related problems and key lemmas. At first, we describe the parallel-machine scheduling problem with divisible processing times (the PS-DT problem) in the following way.

Problem 1 (the PS-DT problem (Coffman et al. 1987)) *Given a set* $M = \{a_1, a_2, \ldots, a_m\}$ *of m identical machines and a set* $J = \{b_1, b_2, \ldots, b_n\}$ *of n independent jobs, each job* $b_j \in J$ *having a processing time* $p_j \in \mathbb{Z}^+$*, where these processing times are divisible, i.e., either* $p_i \mid p_j$ *or* $p_j \mid p_i$ *for any two different jobs* b_i *and* b_j *in* J*, it is asked to schedule jobs in* J *on machines such that each job* b_j *is only continuously processed on one machine, the objective is to minimize the makespan, i.e., the total timespan required to execute jobs in* J*.*

For convenience, according to the content encountered in the sequel, we use $I = (M, J; p, e)$ to denote an instance of either the CPS-DTTP problem or the CPS-DTMP problem, and we sometimes use $I = (M, J; p, e; B)$ to denote an instance of the CPS-DTBP problem, respectively. From either an instance $I = (M, J; p, e)$ or an instance $I = (M, J; p, e; B)$ as mentioned above, we can construct an instance $\tau(I) = (M, J; p)$ of the PS-DT problem with a processing time function $p : J \to \mathbb{Z}^+$. In addition, given a subset $A \subseteq J$ and a schedule T for jobs in A, we denote $C_{\max}(A, T)$ to be the makespan of such a schedule T for jobs in A, i.e., the timespan required to execute jobs in A, respective to this schedule T, and $e(J \backslash A) = \sum_{b_j \in J \backslash A} e_j$, respectively. And we call a schedule T for jobs in A to be feasible if each job in A scheduled only on one machine once to be executed without interruption with its processing time. Furthermore, we remind $e(\phi) = 0$, $e_{\max}(\phi) = 0$ and $e_{\max}(J') = \max\{e_i \mid b_i \in J' \subseteq J\}$.

On the other hand, Graham (1969) presented the longest processing time algorithm (the LPT algorithm) for solving the $P \mid\mid C_{\max}$ problem. In addition, Coffman et al. (1987) considered the bin packing problem with divisible item sizes (the BP-DS problem), and they showed in the same paper that the LPT algorithm (Graham 1969) also optimally solves the PS-DT problem in polynomial time, which is restated in the following

Lemma 1 (Coffman et al. 1987) *The LPT algorithm in* (Graham 1969)*, also denoted by the algorithm* $\mathcal{A}_{PS-DT}$*, optimally solves the PS-DT problem in time* $O(n \log n)$*, where n is the number of jobs with divisible processing times.*

Secondly, we describe the multiple knapsack problem with divisible item sizes (the MKP-DS problem) in the following way.

Problem 2 (The MKP-DS problem (Detti 2009)) *Given a set N of m knapsacks with same capacity L and a set* $X = \{x_1, \ldots, x_n\}$ *of n items, each item* $x_j \in X$ *has a size* $s_j \in \mathbb{Z}^+$ *and a profit* $v_j \in \mathbb{R}^+$*, where these n item sizes are divisible, i.e., either* $s_i \mid s_j$ *or* $s_j \mid s_i$ *for any two distinct items* x_i *and* x_j *in X, it is asked to find a subset* $X' \subseteq X$ *and a scheme T for items in* X' *such that the items in* X' *are all packed into m knapsacks under the constraints that the summation of sizes of items in each knapsack does not exceed the capacity L, the objective is to maximize the value of profits of assigned items in these m knapsacks, i.e.,* $\max\{\sum_{x_i \in X'} v_i \mid X' \subseteq X$ *is chosen to satisfy the aforementioned constraints*$\}$.

In the similar way, we use $I = (N, X; s, v; L)$ to denote an instance of the MKP-DS problem. In addition, we denote $v(X') = \sum_{x_j \in X'} v_j$ for a subset $X' \subseteq X$.

Detti (2009) designed an exact algorithm to solve the MKP-DS problem in strongly polynomial time, whose structural descriptions in details can be found in (Detti 2009), and we may provide the related conclusion in the following way.

Lemma 2 (Detti 2009) *There is an exact combinatorial algorithm, denoted by the algorithm $\mathcal{A}_{MKP-DS}$, to optimally solve the MKP-DS problem, and this algorithm runs in time $O(n \log n + nm)$, where m is the number of knapsacks with same capacity L and n is the number of items with divisible sizes, respectively.*

At present, we intend to transfer an instance of either the CPS-DTTP problem or the CPS-DTBP problem into an instance of the MKP-DS problem in the following ways. Given an instance $I = (M, J; p, e)$ of the CPS-DTMP problem or an instance $I = (M, J, p, e; B)$ the CPS-DTBP problem and a fixed integer k, we can construct an instance $\rho_k(I) = (N, X; s, v; k)$ of the MKP-DS problem, i.e., N is the set of m knapsacks with same capacity k, each item $x_j \in X$ has its size $s_j \in \mathbb{Z}^+$ and its value $v_j \in \mathbb{R}^+$, where $s_j = p_j$ and $v_j = e_j$ for each job $b_j \in J$. In addition, whenever we use the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009) on this instance $\rho_k(I)$ of the MKP-DS problem, we can obtain the following result.

Lemma 3 (Detti 2009) *Given an instance $\rho_k(I)$ of the MKP-DS problem, the algorithm $\mathcal{A}_{MKP-DS}$ can determine a subset $X' \subseteq X$, where the items in which can be packed into m knapsacks under the constraints that the summation of sizes of items in each knapsack does not exceed the capacity k, such that $v(X')$ is maximized, and this algorithm runs in time $O(n \log n + nm)$, where n is the number of jobs with divisible processing times and m the number of these knapsacks with same capacity k, respectively.*

Using Lemma 3, we can indeed determine the following result.

Lemma 4 *Given a set $M = \{a_1, a_2, \ldots, a_m\}$ of m identical machines and a set $J = \{b_1, b_2, \ldots, b_n\}$ of n independent jobs, each job $b_j \in J$ has a processing time $p_j \in \mathbb{Z}^+$ and a penalty $e_j \in \mathbb{R}^+$, where these processing times are divisible, i.e., either $p_i \mid p_j$ or $p_j \mid p_i$ for any two different jobs b_i and b_j in J, and for any positive integer k, when we execute the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009) on the instance $\rho_k(I)$, we can determine a subset J' ($\subseteq J$), where the jobs in $J' = \{b_{j_1}, b_{j_2}, \ldots, b_{j_t}\}$ constructed from the item subset $X' = \{x_{j_1}, x_{j_2}, \ldots, x_{j_t}\}$, produced by the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009), such that $e(J')$ is maximized, and this algorithm runs in time $O(n \log n + nm)$.*

For convenience, using Lemma 4, we may treat two sets X and J as the coincide set, i.e., we may assume that the set X and the set J are both same set, implying that we may treat the subset $X' = \{x_{j_1}, x_{j_2}, \ldots, x_{j_t}\}$ as the subset $J' = \{b_{j_1}, b_{j_2}, \ldots, b_{j_t}\}$, which are produced in Lemma 4, and X' and J' are the same in the sequel sections.

3 An exact algorithm to solve the CPS-DTTP problem

In this section, we consider the constrained parallel-machine scheduling problem with divisible processing times and total penalties (the CPS-DTTP problem). Concretely,

given an instance $I = (M, J; p, e)$ of the CPS-DTTP problem, we are asked to find a subset A ($\subseteq J$) and a scheme T for jobs in A to satisfy the requirements in two parts, i.e., (1) each job in A is continuously executed on one machine with its processing time, and (2) each job not in A is rejected with its penalty that we must pay for, the objective is to minimize the value $h(A) = C_{\max}(A, T) + e(J\backslash A)$.

The strategy of our algorithm to solve the CPS-DTTP problem is presented in the following ways.

(1) Given an instance $I = (M, J; p, e)$ of the CPS-DTTP problem, we construct an instance $\tau(I) = (M, J; p)$ of the PS-DP problem, and then executing the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969) on this instance $\tau(I)$, we may find an optimal solution for the instance $\tau(I)$ of the PS-DP problem, having its optimal value C;
(2) Given an instance $I = (M, J; p, e)$ of the CPS-DTTP problem and a fixed integer k, where $k = 1, 2, \ldots, C$, we construct an instance $\rho_k(I) = (N, X; s, v; k)$ of the MKP-DS problem, and then executing the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009) on that instance $\rho_k(I)$, we can find a schedule T_k for jobs in a subset $A_k \subseteq J$, where each job in A_k can be executed on one of m machines to satisfy the aforementioned constraints, such that $e(A_k)$ is maximized, equivalently that $e(J\backslash A_k)$ is minimized;
(3) Choose the best one among all feasible schedules in (2), having the minimum value of a makespan chosen plus the summation of penalties of all rejected jobs.

We design a following algorithm, denoted by the algorithm $\mathcal{A}_{CPS-DTTP}$, to solve the CPS-DTTP problem in the following ways.

Algorithm: $\mathcal{A}_{CPS-DTTP}$
INPUT: An instance $I = (M, J; p, e)$ of the CPS-DTTP problem;
OUTPUT: A subset $A_{k_0} \subseteq J$, a schedule T_{k_0} for jobs in A_{k_0} and the value $h(A_{k_0}) = C_{\max}(A_{k_0}, T_{k_0}) + e(J\backslash A_{k_0})$.
Begin
Step 1 Given an instance $I = (M, J; p, e)$ of the CPS-DTTP problem, we construct an instance $\tau(I) = (M, J; p)$ of the PS-DT problem, and then using the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969) on this instance $\tau(I)$, we can determine an optimal solution and its optimal value C for the PS-DT problem;
Step 2 Denote $A_0 = \phi$, $T_0 = (\phi, \ldots, \phi; \phi)$, $C_{\max}(A_0, T_0) = 0$, and $e(J\backslash A_0) = \sum_{b_j \in J} e_j$;
Step 3 For $k = 1$ to C do:
3.1 Using the instance $I = (M, J; p, e)$ of the CPS-DTTP problem and a fixed integer k, we construct an instance $\rho_k(I) = (N, Y; s, v; k)$ of the MKP-DS problem as mentioned above;
3.2 Executing the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009) on the instance $\tau_k(I)$ of the MKP-DS problem, we can determine a schedule $T_k = (S_{1_k}, S_{2_k}, \ldots, S_{m_k}; A_k)$, where $A_k = \cup_{i=1}^{m} S_{i_k}$ and jobs in S_{i_k} are all executed on the machine a_i under the constraints that the completion of each machine a_i does not exceed the fixed integer k, such that the value $e(A_k)$ is maximized, equivalently, that $e(J\backslash A_k)$ is minimized;
3.3 Denote $C_{\max}(A_k, T_k) = k$;
Step 4 Find a subset $A_{k_0} \in \{A_0, A_1, A_2, \ldots, A_C\}$ and a scheme $T_{k_0} \in \{T_0, T_1, T_2, \ldots, T_C\}$, satisfying the following $C_{\max}(A_{k_0}, T_{k_0}) + e(J\backslash A_{k_0}) =$

$\min\{C_{\max}(A_k, T_k) + e(J\backslash A_k) \mid k = 0, 1, 2, \ldots, C\}$, and denote $h(A_{k_0}) = C_{\max}(A_{k_0}, T_{k_0}) + e(J\backslash A_{k_0})$;
Step 5 Output "the subset A_{k_0} ($\subseteq J$), the schedule T_{k_0} for jobs in A_{k_0} and the value $h(A_{k_0}) = C_{\max}(A_{k_0}, T_{k_0}) + e(J\backslash A_{k_0})$".
End

Using the algorithm $\mathcal{A}_{CPS-DTTP}$, we can determine the following result.

Theorem 1 *The algorithm $\mathcal{A}_{CPS-DTTP}$ is an exact algorithm to optimally solve the CPS-DTTP problem, and this algorithm runs in pseudo-polynomial time $O((n \log n + nm)C)$, where n is the number of jobs, m is the number of machines for an instance $I = (M, J; p, e)$ of the CPS-DTTP problem, and C is the optimal value for the instance $\tau(I) = (M, J; p)$ of the PS-DT problem.*

Proof Given an instance $I = (M, J; p, e)$ of the CPS-DTTP problem, we may suppose that there is an optimal solution for the instance I, i.e., there is an optimal schedule $T^* = (S_1^*, S_2^*, \ldots, S_m^*; A^*)$ for jobs in a subset A^* ($\subseteq J$) with the optimal value $h(A^*) = C_{\max}(A^*, T^*) + e(J\backslash A^*)$, where $A^* = \cup_{i=1}^m S_i^*$, $C_{\max}(A^*, T^*) = \max\{\sum_{b_j \in S_i^*} p_j \mid i = 1, 2, \ldots, m\}$, $e(J\backslash A^*) = \sum_{b_j \in J\backslash A^*} e_j$ and the completion time of jobs in each set S_i^* is not beyond $C_{\max}(A^*, T^*)$. And for the same instance I, the algorithm $\mathcal{A}_{CPS-DTTP}$ can produce a schedule $T_{k_0} = (S_{1_{k_0}}, S_{2_{k_0}}, \ldots, S_{m_{k_0}}; A_{k_0})$ for jobs in A_{k_0} with the output value $h(A_{k_0}) = C_{\max}(A_{k_0}, T_{k_0}) + e(J\backslash A_{k_0})$, where $A_{k_0} = \cup_{i=1}^m S_{i_{k_0}}$. We shall prove $h(A^*) = h(A_{k_0})$ in the sequel, where C is the optimal value of the instance $\tau(I) = (M, J; p)$ for the PS-DT problem and $k_0 \in \{0, 1, 2, \ldots, C\}$.

Since we execute these n independent jobs in J on the m machines, where the processing times of these n independent jobs are divisible, using the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969), i.e., the LPT algorithm (seeing Lemma 1), we can obtain the minimum makespan for these jobs in J is C, this implies that the makespan of the jobs in the optimal set A^* ($\subseteq J$) is no more than C, in other words, the value $C_{\max}(A^*, T^*) \in \{0, 1, 2, \ldots, C\}$. For convenience, we denote $k^* = C_{\max}(A^*, T^*)$. According to Lemma 4, we obtain the fact that the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009) at Step 2 can produce a subset $A_{k^*} \subseteq J$, satisfying

(1) the jobs in A_{k^*} are all executed on the m machines under the constraints that the completion time of each machine in M does not exceed the value k^*, and
(2) the summation of profits in the instance $\rho_{k^*}(I) = (N, Y; s, v; k^*)$ (i.e., penalties in the instance $I = (M, J; p, e)$) of jobs in A_{k^*} is maximized among all subsets of J under the constraints in (1), i.e., $e(A_{k^*}) = \max\{e(A) \mid$ the jobs in a subset A ($\subseteq J$) are all executed on m machines and the makespan of jobs in A is no more than the value $k^*\}$.

At present, we obtain $e(A_{k^*}) \geq e(A)$ for each subset A ($\subseteq J$), where jobs in A are executed on m machines such that the completion time of these machines does not exceed the fixed integer k^*. In particular, we have $e(A_{k^*}) \geq e(A^*)$, where we have $A^* \subseteq J$.

Since $e(J) = e(A_{k^*}) + e(J\backslash A_{k^*}) = e(A^*) + e(J\backslash A^*)$, using the result $e(A_{k^*}) \geq e(A^*)$, we can have $e(J\backslash A_{k^*}) = e(J) - e(A_{k^*}) \leq e(J) - e(A^*) = e(J\backslash A^*)$. Then,

we can obtain the following

$$\begin{aligned} h(A_{k_0}) &= C_{\max}(A_{k_0}, T_{k_0}) + e(J\backslash A_{k_0}) \\ &= \min\{C_{\max}(A_k, T_k) + e(J\backslash A_k) \mid k = 0, 1, 2, \ldots, C\} \\ &\le k^* + e(J\backslash A_{k^*}) \\ &\le k^* + e(J\backslash A^*) \\ &= C_{\max}(A^*, T^*) + e(J\backslash A^*) \\ &= h(A^*) \end{aligned}$$

where the second equality comes from Step 4 of the algorithm $\mathcal{A}_{CPS-DTTP}$, the third inequality comes from the fact $0 \le k^* \le C$, the fourth inequality comes from the fact $e(J\backslash A_{k^*}) \le e(J\backslash A^*)$ and the fifth equality comes from the definition $k^* = C_{\max}(A^*, T^*)$.

By the minimality of the optimal solution $A^* \subseteq J$ for an instance I of the CPS-DTTP problem, we have that $C_{\max}(A_{k_0}, T_{k_0}) + e(J\backslash A_{k_0}) = C_{\max}(A^*, T^*) + e(J\backslash A^*)$, i.e., $h(A^*) = h(A_{k_0})$, this implies that the schedule T_{k_0} for jobs in A_{k_0} ($\subseteq J$) produced by the algorithm $\mathcal{A}_{CPS-DTTP}$ is also an optimal solution for an instance I of the CPS-DTTP problem. This shows that the algorithm $\mathcal{A}_{CPS-DTTP}$ indeed correctly solves the CPS-DTTP problem.

The complexity of the algorithm $\mathcal{A}_{CPS-DTTP}$ can be determined as follows. (1) By using the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969), Step 1 needs at most time $O(n \log n)$ to compute the optimal value C of the instance $\tau(I) = (M, J; p)$ for the PS-DT problem; (2) Step 2 needs at most time $O(n)$ to compute $e(J)$; (3) For each fixed integer $k \in \{1, 2, \ldots, C\}$, the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009) needs time $O(n \log n + nm)$ to find a subset $A_k \subseteq J$ such that all jobs in A_k can be executed on m machines under the aforementioned constraints and that $e(A_k)$ is maximized, implying that Step 3 needs at most time $O((n \log n + nm)C)$ to execute C iterations; (4) Step 4 needs time $O(C)$ to find a subset $A_{k_0} \in \{A_0, A_1, A_2, \ldots, A_C\}$ and a scheme $T_{k_0} \in \{T_0, T_1, T_2, \ldots, T_C\}$, having minimum value $h(A_{k_0}) = C_{\max}(A_{k_0}, T_{k_0}) + e(J\backslash A_{k_0})$. Thus, the running time of the algorithm $\mathcal{A}_{CPS-DTTP}$ is in total $O((n \log n + nm)C)$, where C is the optimal value for the instance $\tau(I) = (M, J; p)$ of the PS-DT problem.

This completes the proof of the theorem. □

4 An exact algorithm to solve the CPS-DTMP problem

In this section, we consider the constrained parallel-machine scheduling problem with divisible processing times and maximum penalty (the CPS-DTMP problem). Specifically, given an instance $I = (M, J; p, e)$ of the CPS-DTMP problem, we are asked to determine a subset A ($\subseteq J$) and a scheme T for jobs in A to satisfy the requirements in two parts, i.e., (1) each job in A is continuously executed on one machine with its processing time, and (2) each job not in A is rejected with its penalty that we must pay for, the objective is to minimize the value $f(A) = C_{\max}(A, T) + e_{\max}(J\backslash A)$, where $e_{\max}(J\backslash A) = \max\{e_j \mid b_j \in J\backslash A\}$.

We observe the following facts. In general, given an instance $I = (M, J; p, e)$ of the CPS-DTMP problem, if A^* ($\subseteq J$) is an optimal solution to this instance I, then we can obtain the optimal value $f(A^*) = C_{\max}(A^*, T^*) + e_{\max}(J \backslash A^*)$. According to the definition of $e_{\max}(J \backslash A^*)$, this value $e_{\max}(J \backslash A^*)$ is exactly one of these n values e_j, where $j = 1, 2, \ldots, n$. For convenience, we may denote this value $e_{\max}(J \backslash A^*)$ as e^*, where $e^* \in \{e_1, e_2, \ldots, e_n\}$, such that $C_{\max}(A^*, T^*) + e^*$ is minimized among all feasible solutions to the instance I. In order to determine an optimal solution A^* ($\subseteq J$) to the instance I with the minimum value $C_{\max}(A^*, T^*) + e^*$, we may enumerate all possibilities to find such a value e^* from the set $\{e_1, e_2, \ldots, e_n\}$, where the value $C_{\max}(A^*, T^*)$ is obtained by solving some related instance $\tau(I)$, constructed from the instance I, of the PS-DT problem.

Motivated by the aforementioned facts, and executing the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969) on an instance $\tau(I)$ of the PS-DP problem, we use the following strategy to design our algorithm for solving the CPS-DTMP problem.

(1) For convenience, construct a new 'dummy' job b_0 with its processing time $p_0 = 0$ and penalty $e_0 = 0$;
(2) For each $j = 0, 1, 2, \ldots, n$, choose this job b_j and firmly reject it, and construct a set $A_j = \{b_t \in J \mid e_t > e_j\}$. In addition, using the set A_j, construct a new instance of the CPS-DTMP problem and an instance of the PS-DT problem, respectively, then implement suitable algorithms to produce an optimal schedule and the value of makespan on these instances;
(3) Choose the best one among these $n + 1$ feasible schedules, having the minimum value of makespan for all accepted jobs plus maximum penalty of all rejected jobs.

We design a following algorithm, denoted by the algorithm $\mathcal{A}_{CPS-DTMP}$, to solve the CPS-DTMP problem is described in the following ways.

Algorithm: $\mathcal{A}_{CPS-DTMP}$
INPUT: An instance $I = (M, J; p, e)$ of the CPS-DTMP problem;
OUTPUT: A subset $A_{j_0} \subseteq J$, a schedule T_{j_0} for jobs in A_{j_0} and the value $f(A_{j_0}) = C_{\max}(A_{j_0}, T_{j_0}) + e_{j_0}$.
Begin
Step 1 Construct a new 'dummy' job b_0 with its processing time $p_0 = 0$ and penalty $e_0 = 0$;
Step 2 For $j = 0$ to n do:
2.1 Firmly reject the job b_j, and construct a set $A_j = \{b_t \in J \mid e_t > e_j\}$;
2.2 Using an instance I, construct another instance $I_j = (M, A_j; p, e)$ of the CPS-DTMP problem;
2.3 Using this instance I_j of the CPS-DTMP problem, construct an instance $\tau(I_j) = (M, A_j; p)$ of the PS-DT problem mentioned in Section 2;
2.4 Implementing the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969) on this instance $\tau(I_j) = (M, A_j; p)$ of the PS-DT problem, find a schedule T_j for the subset A_j such that the makespan $C_{\max}(A_j, T_j)$ for all jobs in A_j is minimized;
2.5 Denote $f(A_j) = C_{\max}(A_j, T_j) + e_j$;
Step 3 Find a subset $A_{j_0} \in \{A_0, A_1, A_2, \ldots, A_n\}$ and a scheme $T_{j_0} \in \{T_0, T_1, T_2, \ldots, T_n\}$, satisfying the following

$$f(A_{j_0}) = \min\{f(A_j) \mid j = 0, 1, 2, \ldots, n\};$$

Step 4 Output "the subset A_{j_0}, the schedule T_{j_0} for jobs in A_{j_0} and the value $f(A_{j_0})$".
End

Using the algorithm $\mathcal{A}_{CPS-DTMP}$, we can determine the following result.

Theorem 2 *The algorithm $\mathcal{A}_{CPS-DTMP}$ is an exact algorithm for optimally solving the CPS-DTMP problem, and this algorithm runs in time $O(n^2 \log n)$, where n is the number of jobs.*

Proof Given an instance $I = (M, J; p, e)$ of the CPS-DTMP problem, we may assume that there is an optimal solution for the instance I of the CPS-DTMP problem, i.e., there is a subset A^* ($\subseteq J$) and a schedule T^* for jobs in A^* with a value $f(A^*) = C_{\max}(A^*, T^*) + e_{\max}(J\backslash A^*)$ such that $f(A^*)$ is minimized among all values in $\{C_{\max}(A, T) + e_{\max}(J\backslash A) \mid A \subseteq J$ and T is a feasible schedule for jobs in $A\}$, where the jobs in each subset A of J can be scheduled on these m machines, implying that $f(A^*) = C_{\max}(A^*, T^*) + e_{\max}(J\backslash A^*) = \min\{C_{\max}(A, T) + e_{\max}(J\backslash A) \mid A$ is a subset (of J) and T is a feasible schedule for jobs in $A\}$. For convenience, if $A^* \neq J$, we denote $e_{j^*} = e_{\max}(J\backslash A^*)$, where $b_{j^*} \in J\backslash A^*$ and $e_t \leq e_{j^*}$ for each job $b_t \in J\backslash A^*$. And we remind to denote $e_{\max}(\phi) = 0$.

For the same instance I, by using the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969) (seeing Step 2 of the algorithm $\mathcal{A}_{CPS-DTMP}$), the algorithm $\mathcal{A}_{CPS-DTMP}$ produces a subset A_{j_0} ($\subseteq J$) and the output value $f(A_{j_0})$ such that $f(A_{j_0}) = C_{\max}(A_{j_0}, T_{j_0}) + e_{\max}(J\backslash A_{j_0}) = \min\{C_{\max}(A_j, T_j) + e_{\max}(J\backslash A_j) \mid j = 0, 1, 2, \ldots, n\}$, where the jobs in A_j ($\subseteq J$) are all executed on m machines such that $C_{\max}(A_j, T_j)$ is minimized among all schedules for jobs in A_j (for each $j = 0, 1, 2, \ldots, n$). In particular, we assume a 'dummy' job b_0 whose rejection cost is less than the penalty of jobs in J, i.e., $e_0 < e_j$ for each job b_j in J, we obtain the fact $A_0 = \{b_t \in J \mid e_t > e_0\} = J$ when $j = 0$, i.e., no jobs in J is rejected in this case, and the jobs in the subset A_0 ($= J$) are all executed on m machines in this case. We shall prove $f(A^*) = f(A_{j_0})$ in the sequel.

Without loss of generality, we may assume that b_{j^*} is the job with the largest penalty in the subset $J\backslash A^*$, satisfying $A_{j^*} = \{b_t \mid e_t > e_{j^*}\} = A^*$. Using the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969) on the set A_{j^*}, using Lemma 1, we can schedule each job $b_j \in A_{j^*}$ on one machine in M such that the makespan for all jobs in A_{j^*} is minimized, i.e., $C_{\max}(A_{j^*}, T_{j^*}) = \min\{C_{\max}(A_{j^*}, T) \mid T$ is a feasible schedule for jobs in $A_{j^*}\}$. This shows that $C_{\max}(A_{j^*}, T_{j^*}) \leq C_{\max}(A^*, T^*)$ whenever T^* is a feasible schedule for jobs in A_{j^*}.

Now, we obtain the following

$$\begin{aligned} f(A_{j_0}) &= C_{\max}(A_{j_0}, T_{j_0}) + e_{\max}(J\backslash A_{j_0}) \\ &= \min\left\{C_{\max}(A_j, T_j) + e_{\max}(J\backslash A_j) \mid j = 0, 1, 2, \ldots, n\right\} \\ &\leq C_{\max}(A_{j^*}, T_{j^*}) + e_{\max}(J\backslash A^*) \\ &\leq C_{\max}(A^*, T^*) + e_{\max}(J\backslash A^*) \\ &= f(A^*) \end{aligned}$$

where the third equality comes from the facts $0 \leq j^* \leq n$, and the fourth inequality comes from the fact $C_{\max}(A_{j^*}, T_{j^*}) \leq C_{\max}(A^*, T^*)$.

Thus, by the minimality of the optimal solution for an instance I of the CPS-DTMP problem, we have $C_{\max}(A_{j_0}, T_{j_0}) + e_{\max}(J \backslash A_{j_0}) = C_{\max}(A^*, T^*) + e_{\max}(J \backslash A^*)$, i.e. $f(A_{j_0}) = f(A^*)$, implying that the subset A_{j_0} with the value $f(A_{j_0}) = C_{\max}(A_{j_0}, T_{j_0}) + e_{\max}(J \backslash A_{j_0})$ produced by the algorithm $\mathcal{A}_{CPS-DTMP}$ is also an optimal solution for an instance I of the CPS-DTMP problem. This shows that the algorithm $\mathcal{A}_{CPS-DTMP}$ indeed works correctly to solve the CPS-DTMP problem.

The complexity of the algorithm $\mathcal{A}_{CPS-DTMP}$ can be determined as follows. (1) Step 1 needs at most time $O(n)$ to be executed; (2) For each $j \in \{0, 1, 2, \ldots, n\}$, the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969) needs time $O(n \log n)$ to schedule all jobs in each subset A_j on these m machines such that $C_{\max}(A_j, T_j)$ is minimized, implying that Step 2 needs at most time $O(n^2 \log n)$ to execute $n + 1$ iterations; (3) Step 3 needs at most time $O(n)$ to find a subset $A_{j_0} \in \{A_0, A_1, A_2, \ldots, A_n\}$ and a scheme $T_{j_0} \in \{T_0, T_1, T_2, \ldots, T_n\}$, having minimum value $f(A_{j_0})$. Thus, the running time of the algorithm $\mathcal{A}_{CPS-DTMP}$ is in total $O(n^2 \log n)$.

This completes the proof of the theorem. □

5 An exact algorithm to solve the CPS-DTBP problem

In this section, we consider the constrained parallel-machine scheduling problem with divisible processing times and bounded penalty (the CPS-DTBP problem). Specifically, given an instance $I = (M, J; p, e; B)$, we hope to determine a subset A ($\subseteq J$) and a scheme T for jobs in A to satisfy the constraint that the summation of penalties paid for the jobs not in A is no more than a bound B, i.e., $\sum_{b_j \in J \backslash A} e_j \leq B$, the objective is to minimize the makespan $C_{\max}(A, T)$.

We plan to combine the binary algorithm (Schrijver 2003) and other suitable algorithms (Graham 1969; Detti (2009) to solve the CPS-DTBP problem in the following ways. Given an instance $I = (M, J; p, e; B)$ of the CPS-DTBP problem and a fixed number k, we construct an instance $\tau(I)$ of the PS-DT problem and an instance $\rho_k(I)$ of the MKP-DS problem, and executing the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969) on this instance $\tau(I)$ of the PS-DT problem and then the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009) on that instance $\rho_k(I)$ of the MKP-DS problem, where the execution of the second algorithm may be executed many times respect to the fixed constant k, we can optimally solve the CPS-DTBP problem. The strategy of our algorithm is presented in the following ways.

(1) If the summation $e(J)$ of penalties paid for jobs in J is not greater than B, then jobs are all rejected, i.e., no job in J is executed on each of machines, and stop;
(2) If the summation $e(J)$ of penalties paid for jobs in J is greater than B, executing the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969) on this instance $\tau(I)$ of the PS-DT problem, we find an optimal makespan C for jobs in J, implying that the optimal value for the instance $I = (M, J; p, e; B)$ of the CPS-DTBP problem is between 1 and C. Since we may assume that, according to the requirements of computer science for input data, the two weight functions $p(\cdot)$ and $e(\cdot)$ are positive integer functions, we

intend to use the binary algorithm to find this optimal value. In details, given each possible number k (we may guess an optimal makespan), where $k = 1, 2, \ldots, C$, we use the binary search algorithm in (Schrijver 2003) to find a subset A_k, where the jobs in A_k can be executed on m machines to satisfy the completion time for jobs not in A_k less or equal to k, such that $e(A_k)$ is maximized, equivalently, that $e(J \backslash A_k)$ is minimized;

(3) Choose the best one among all feasible schedules, having the minimum of makespans to satisfy the constraint that the summation of penalties paid for jobs not in A_k is less or equal to B.

We design a following algorithm, denoted by the algorithm $\mathcal{A}_{CPS-DTBP}$, to solve the CPS-DTBP problem is described in the following ways.

Algorithm: $\mathcal{A}_{CPS-DTBP}$
INPUT: An instance $I = (M, J; p, e; B)$ of the CPS-DTBP problem;
OUTPUT: A subset $A_{k_0} \subseteq J$, a schedule T_{k_0} for jobs in A_{k_0} and the value k_0.
Begin
Step 1 Given an instance $I = (M, J; p, e; B)$ of the CPS-DTBP problem, we construct an instance $\tau(I) = (M, J; p)$ of the PS-DT problem mentioned in Section 2, and then executing the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969) on this instance $\tau(I)$, we find an optimal solution and its optimal value C for the instance $\tau(I) = (M, J; p)$;
Step 2 If $(e(J) \leq B)$ then
Output"the subset $A_0 = \phi$, and the makespan $k_0 = 0$", and STOP;
Else
Denote $B_L = 0$ and $B_U = C$;
Step 3 Denote $k_0 = \lceil (B_L + B_U)/2 \rceil$;
Step 4 We execute the following two steps
4.1 Given an instance I of the CPS-DTBP problem and the number k_0, we construct an instance $\rho_{k_0}(I) = (N, Y; s, v; k_0)$ of the MKP-DS problem as mentioned above;
4.2 Executing the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009) on this instance $\rho_{k_0}(I)$ of the MKP-DS problem, we find a subset A_{k_0} $(\subseteq A)$ and then schedule all jobs in A_{k_0} on these m machines, satisfying that the completion time (as the numerical values of sizes of jobs) for jobs not in A_{k_0} is less or equal to k_0, such that the value $e(A_{k_0})$ is maximized, equivalently that $e(J \backslash A_{k_0})$ is minimized;
Step 5 If $(e(J \backslash A_{k_0}) \leq B)$ then
Denote $B_U = k_0$;
Else
Denote $B_L = k_0$;
Step 6 If $(B_U - B_L > 1)$ then
Go to Step 3;
Step 7 Output "the subset A_{k_0}, the schedule T_{k_0} for jobs in A_{k_0} and the value $k_0 = B_U$".
End

Using the algorithm $\mathcal{A}_{CPS-DTBP}$, we can determine the following result.

Theorem 3 *The algorithm $\mathcal{A}_{CPS-DTBP}$ is an exact algorithm for optimally solving the CPS-DTBP problem, and this algorithm runs in polynomial time $O((n \log n + nm) \log C)$, where n is the number of jobs, m is the number of machines and C is the optimal value for the same instance of the PS-DT problem.*

Proof Given an instance $I = (M, J; p, e; B)$ of the CPS-DTBP problem, we may assume that there is an optimal solution, i.e., a subset A^* ($\subseteq J$) with value $k^* = C_{\max}(A^*, T^*)$ and a schedule T^* for jobs in A^*, satisfying the constraint $e(J \backslash A^*) \leq B$, such that the value k^* is minimized among all values in $\{C_{\max}(A, T) \mid A \subseteq J$ and T is a feasible schedule for jobs in $A\}$, where each of jobs in the subset A of J can be scheduled only on one of m machines, implying that $k^* = C_{\max}(A^*, T^*) = \min\{C_{\max}(A, T) \mid A$ is a subset (of J), and T is a feasible schedule for jobs in $A\}$. We obtain $k^* \geq 1$ in this case. For the same instance I, the algorithm $\mathcal{A}_{CPS-DTBP}$ produces a subset A_{k_0} with a value $k_0 = C_{\max}(A_{k_0}, T_{k_0})$ to satisfy the constraint $e(J \backslash A_{k_0}) \leq B$. We shall prove $k^* = k_0$ in the sequel.

If the algorithm $\mathcal{A}_{CPS-DTBP}$ produces the output solution $A_0 = \phi$ with the value $k_0 = 0$ at Step 1, we have the fact $e(J) \leq B$, implying that these n jobs in J are all rejected. In this case, it is easy for us to obtain $k^* = k_0$ ($= 0$) such that this solution $A_0 = \phi$ with the value $k_0 = 0$ is an optimal solution. In the following arguments, we may consider the case $e(J) > B$, equivalently, there is at least one job that is scheduled on one machine, implying that $A^* \neq \phi$ and $k^* \neq 0$.

For each $k = 1, 2, \ldots, C$, using the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009) at Step 4, we can determine a subset A_k and a schedule to process all jobs in A_k on m machines to satisfy the aforementioned constraint such that $e(A_k)$ is maximized, equivalently, $e(J \backslash A_k)$ is minimized. At Step 5, we determine whether $e(J \backslash A_k) \leq B$ or not, in details, we may assign the value k to be an upper bound B_U if $e(J \backslash A_k) \leq B$ and the value k to be a lower bound B_L otherwise. In other words, when k is an upper bound, we obtain $e(J \backslash A_k) \leq B$, and when k is a lower bound, we obtain $e(J \backslash A_k) > B$, i.e., when each feasible solution satisfies the aforementioned constraint, the value of this feasible solution is between this lower bound B_L and that upper bound B_U. By the assumption that penalties of job are all positive integers, we can obtain integer lower bound and integer upper bound at each recursion of Step 6. At this time, when the difference between an upper bound and a lower bound is exactly equal to 1 and k_0 is exactly that upper bound at present, we obtain the fact that this subset A_{k_0} satisfies the constraint $e(J \backslash A_{k_0}) \leq B$, and however, when an upper bound is $k_0 - 1$, we obtain this subset A_{k_0} to satisfy $e(J \backslash A_{k_0 - 1}) > B$, which show that this subset A_{k_0} is not a feasible solution for the CPS-DTBP problem. This shows that k_0 is the smallest positive integer satisfying the constraint $e(J \backslash A_k) \leq B$, indeed, $k_0 = \min\{k \in \{1, 2, \ldots, C\} \mid e(J \backslash A_k) \leq B\}$.

When we use the algorithm $\mathcal{A}_{PS-DT}$ (Graham 1969) to execute the jobs in J on m machines at Step 1, we can obtain the optimal makespan C for the same instance of the PS-DT problem mentioned in Sect. 2, and it is easy for us to know the fact that the minimum makespan of jobs in the subset $A^* \subseteq J$ is no more than the value C, this shows that $C_{\max}(A^*, T^*) \in \{1, 2, \ldots, C\}$. Remind our assumption $k^* = C_{\max}(A^*, T^*)$. By using Lemma 4, the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009) at Step 4 produces a subset $A_{k^*} \subseteq J$ such that the jobs in A_{k^*} are all processed on m machines for a schedule and that the summation of penalties (these numerical values are treated

as profits for the knapsack problem) is the largest one among all subsets of J to satisfy the aforementioned constraint, implying that $e(A_{k^*}) \geq e(A^*)$, because A^* in an optimal solution is also a subset of J.

Now, we obtain the following

$$\begin{aligned} e(J\backslash A_{k^*}) &= e(J) - e(A_{k^*}) \\ &\leq e(J) - e(A^*) \\ &= e(J\backslash A^*) \\ &\leq B \end{aligned}$$

where the second inequality comes from the facts $e(A_{k^*}) \geq e(A^*)$, the fourth inequality comes from the fact that the optimal solution is also a feasible solution, implying that $e(J\backslash A_{k^*}) \leq B$.

Due to the minimality of k_0 in the algorithm $\mathcal{A}_{CPS-DTBP}$, i.e., $k_0 = \min\{k \in \{1, 2, \ldots, C\} \mid e(J\backslash A_k) \leq B\}$, we obtain $k_0 = C_{\max}(A_{k_0}, T_{k_0}) \leq k^* = C_{\max}(A^*, T^*)$. And by the minimality of the optimal solution k^* for an instance I of the CPS-DTBP problem, we have $k_0 = k^*$, implying that this subset A_{k_0} with the value k_0 produced by the algorithm $\mathcal{A}_{CPS-DTBP}$ is also an optimal solution for an instance I of the CPS-DTBP problem. This shows that the algorithm $\mathcal{A}_{CPS-DTBP}$ indeed works correctly to solve the CPS-DTBP problem.

The complexity of the algorithm $\mathcal{A}_{CPS-DTBP}$ can be determined as follows. (1) Step 1 needs at most time $O(n \log n)$ to compute C; (2) Step 2 needs at most time $O(n)$ to compute $e(J)$; (3) Step 3 needs at most time $O(1)$ to compute k_0; (4) Step 4 needs at most time $O(n \log n + nm)$ to find a subset A_k for some integer k by using the algorithm $\mathcal{A}_{MKP-DS}$ (Detti 2009); (5) Step 5 needs at most time $O(1)$ to determine the size relationship between $e(J\backslash A_k)$ and B, and assign k to B_U or B_L; (6) Step 6 determine the difference between B_U and B_L, and if the difference between B_U and B_L is exactly equal to 1, we determine the value k_0, and if this difference is greater than to 1, it returns to Step 3 and continues to use the binary search algorithm in (Schrijver 2003) to iterate, implying that Steps 3–6 have to be executed for at most $\log C$ iterations until the difference between B_U and B_L is exactly equal to 1. Thus, the running time of the algorithm $\mathcal{A}_{CPS-DTBP}$ is in total time $O((n \log n + nm) \log C)$.

This completes the proof of the theorem. □

6 Conclusion and further research

In this paper, we consider the three versions of the constrained parallel-machine scheduling problem with divisible job sizes and penalties (the CPS-DTP problem), i.e., the CPS-DTTP problem, the CPS-DTMP problem and the CPS-DTBP problem, respectively. We obtain the following three main results.

(1) We design an exact algorithm to optimally solve the CPS-DTTP problem, and this algorithm runs in time $O((n \log n + nm)C)$, where n is the number of jobs with divisible sizes, m is the number of machines for an instance $I = (M, J; p, e)$ of the

CPS-DTTP problem, and C is the optimal value for the instance $\tau(I) = (M, J; p)$ of the PS-DT problem;

(2) We provide an exact algorithm to optimally solve the CPS-DTMP problem, and this algorithm runs in time $O(n^2 \log n)$;

(3) We present an exact algorithm to optimally solve the CPS-DTBP problem, and that algorithm runs in time $O((n \log n + nm) \log C)$, where C is defined in (1).

In further research, we may consider other versions of the CPS-DTP problem. On the other hand, a challenging task is to design some exact algorithms in lower running times to optimally solve CPS-DTTP problem and/or other the constrained parallel-machine scheduling problems mentioned above.

Acknowledgements We are indeed grateful to the two anonymous reviewers for their insightful comments and for their suggested changes that improve the presentation greatly.

Author Contributions Jianping Li has proposed this problem and contributed to providing some ideas, methods, discussion and writing the final manuscript. Runtao Xie has finished design of algorithms, theoretical proofs and the original manuscript. Junran Lichen, Guojun Hu, Pengxiang Pan, and Ping Yang have contributed to providing some ideas, analysis, discussion and revision. All the authors have read and approved the final manuscript.

Funding These authors are supported by the National Natural Science Foundation of China [Nos.11861075,12101593]. Runtao Xie is also supported by the Postgraduate Research and Innovation Foundation of Yunnan University [No.KC-22221282], Junran Lichen is also supported by Fundamental Research Funds for the Central Universities [No.buctrc202219], and Jianping Li is also supported by Project of Yunling Scholars Training of Yunnan Province [No.K264202011820].

Data availability Enquiries about data availability should be directed to the authors.

Declarations

Conflict of interest The authors declare no conflict of interest. References

References

Bartal Y, Leonardi S, Marchetti-Spaccamela A, Sgall J, Stougie L (2000) Multiprocessor scheduling with rejection. SIAM J Discrete Math 13(1):64–78

Bernhard K, Vygen J (2008) Combinatorial optimization: theory and algorithms, 3rd edn. Springer, Berlin

Coffman EG Jr, Garey MR, Johnson DS (1978) An application of bin-packing to multiprocessor scheduling. SIAM J Comput 7(1):1–17

Coffman EG Jr, Garey MR, Johnson DS (1987) Bin packing with divisible item sizes. J Complex 3(4):406–428

Detti P (2009) A polynomial algorithm for the multiple knapsack problem with divisible item sizes. Inf Process Lett 109(11):582–584

Faigle U, Kern W, Turán G (1989) On the performance of on-line algorithms for partition problems. Acta Cybernet 9(2):107–119

Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell Syst Tech J 45(9):1563–1581

Graham RL (1969) Bounds on multiprocessing timing anomalies. SIAM J Appl Math 17(2):416–429

He Y, Min X (2000) On-line uniform machine scheduling with rejection. Computing 65:1–12

Hochbaum DS, Shmoys DB (1987) Using dual approximation algorithms for scheduling problems theoretical and practical results. J ACM JACM 34(1):144–162

Kellerer H, Pferschy U, Pisinger D, Kellerer H, Pferschy U, Pisinger D (2004) Multidimensional knapsack problems. Springer, Berlin

Knuth DE (1997) The art of computer programming: fundamental algorithms. Addison-Wesley, Boston

Lawler EL, Lenstra JK, Kan AHR, Shmoys DB (1993) Sequencing and scheduling: algorithms and complexity. Handb Oper Res Manag Sci 4:445–522

Lenstra JK, Kan AR, Brucker P (1977) Complexity of machine scheduling problems. Annals of discrete mathematics, vol 1. Elsevier, Amsterdam, pp 343–362

Li W, Li J, Zhang X, Chen Z (2015) Penalty cost constrained identical parallel machine scheduling problem. Theoret Comput Sci 607:181–192

Pinedo ML (2012) Scheduling, vol 29. Springer, Berlin

Potts CN, Strusevich VA (2009) Fifty years of scheduling: a survey of milestones. J Oper Res Soc 60:S41–S68

Schrijver A et al (2003) Combinatorial optimization: polyhedra and efficiency, vol 24. Springer, Berlin

Shabtay D, Gaspar N, Kaspi M (2013) A survey on offline scheduling with rejection. J Sched 16:3–28

Simchi-Levi D (1994) New worst-case results for the bin-packing problem. Naval Res Logist NRL 41(4):579–585

Zhang Y, Ren J, Wang C (2009) Scheduling with rejection to minimize the makespan. In: Combinatorial optimization and applications: third international conference, COCOA 2009, Huangshan, China, 10–12 June , 2009. Proceedings 3. Springer, pp. 411–420

Zheng S, Yue X, Chen Z (2018) Parallel machine scheduling with rejection under special conditions. In: Proceedings of the 8th international conference on communication and network security, pp 139–143