Three notes on scheduling unit-length jobs with precedence constraints to minimize the total completion time

Tianyu Wang, Odile Bellenguez

Three notes on scheduling unit-length jobs with precedence constraints to minimize the total completion time

Tianyu Wang[1] · **Odile Bellenguez**[2]

Abstract
In this paper, we provide three notes on scheduling unit-length jobs with precedence constraints to minimize the total completion time. First, we propose an exact algorithm for in-trees, of which the complexity depends mainly on the graph height, i.e., the length of the longest chain of the precedence graph. We show that this work improves the algorithm in the literature both theoretically and experimentally. Second, we close the open problem for level-orders by showing how it is polynomially solvable. Third, we prove that preemptive scheduling in-trees is strongly NP-hard with arbitrary number of machines, of which the complexity was also open.

Keywords Preemptive scheduling · In-tree · Level-orders · Precedence constraints · Complexity theory

1 Introduction

We consider the problems of scheduling a set $\mathbb{J}^\circ$ of n unit-length jobs on a fixed number m of identical parallel machines. The jobs are subject to precedence constraints described by a directed acyclic graph. We focus on the precedence graphs of *in-trees* (abbrv. i.t.), where each job has at most one direct successor (aka. *in-tree, in-forests*), or *level-orders* (abbrv. l.o.), where jobs on each level of a component precede all jobs on lower levels, see Fig. 1. We use C_j to denote the completion time of job j. The criterion of optimality is to minimize the total completion time $\sum C_j$. In the 3-field notation by Ronald et al. (1979), the problems are $Pm|p_j = 1,\text{i.t.}|\sum C_j$ and $Pm|p_j = 1,\text{l.o.}|\sum C_j$, where p_j represents the length of jobs. When m is arbitrary and preemption is allowed, the third problem considered is $P|p_j = 1,\text{i.t.,pmtn}|\sum C_j$.

The contribution of this paper lies in the establishment of new complexity results:

First, we propose a new algorithm for $Pm|p_j = 1,\text{i.t.}|\sum C_j$, which runs in $O(h^{m+1}n)$ time and $O(h^{m+1})$ space, where h is the length of the longest chain in the graph, called graph height. As can be seen, the complexity depends mainly on the graph height rather than the number of jobs n. This algorithm is both theoretically and experimentally faster than the existing algorithm in the literature, of which the complexity is $O(n^m)$. Specifically, for graphs with limited height, the algorithm is linear in time and constant in space. This shows that the problem is *fixed-parameter tractable* (Cygan et al. 2015) when it is parameterized by $h + m$.

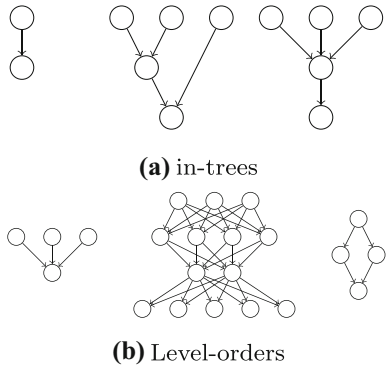
Second, we prove that the open problem $Pm|p_j = 1,\text{l.o.}|\sum C_j$ is polynomially solvable. On the other hand, we prove that $P|p_j = 1,\text{i.t.,pmtn}|\sum C_j$ is strongly NP-hard using a reduction from 3- PARTITION. This study updates the knowledge about the complexity of precedence constrained scheduling problems. It is worth pointing out that the problems with the total completion time criterion and preemption assumption are naturally more complicated than those with the makespan criterion and the non-preemptive assumption, which have been extensively studied in the literature. The common trick is trying to prove that minimizing total completion time and minimizing makespan are equivalent and that preemption is redundant in an optimal schedule. However, neither works for our problem: minimizing makespan is polynomial solvable and preemption does reduce total completion time.

✉ Tianyu Wang
tianyuterry.wang@utoronto.ca

Odile Bellenguez
odile.bellenguez@imt.atlantique.fr

[1] School of Economics and Management, Beihang University, Beijing, China

[2] IMT Atlantique, LS2N, La Chantrerie, 4 rue Alfred Kastler, 44307 Nantes, France



(a) in-trees

(b) Level-orders

Fig. 1 Precedence constraints graph

We overview related literature. Then, we separately study the three problems.

Table 1 Related computational complexity results, where "P","NP-h" and "?" stand for polynomial solvable, NP-hard and open problems

Pmtn	Obj	Prec	Result	
			arbitrary m	fixed m
Not allowed	$C_{\max}$	i.t.	P	P
		o.t.	P	P
		l.o.	NP-h	P
	$\sum C_j$	i.t.	NP-h	P*
		o.t.	P	P
		l.o.	NP-h	?*
Allowed	$C_{\max}$	i.t.	P	P
		o.t.	P	P
		l.o.	NP-h	?
	$\sum C_j$	i.t.	?*	?
		o.t.	P	P
		l.o.	NP-h	?

The three problems with "*" are considered in this paper

2 Literature review

The study of scheduling problems with precedence constraints can be traced back to the 60s. This paper is a part of the effort that clarifies the complexity of scheduling problems with precedence constraints. The computational results achieved in previous literature are summarized in Table 1. Prot Bellenguez-Morineau (2018) survey how the structure of precedence graph may affect the complexity. When the number of machines or the graph height is fixed, our study also falls in a stream of researches about parameterized complexity. Mnich van Bevern (2018) summarized previous results on the parameterized complexity of scheduling problems.

When preemption is not allowed, a great number of studies investigate minimizing the makespan $C_{\max}$. For example, a well-known linear algorithm by Hu (1961) is optimal for in-trees and its reverse graph, *out-tree* (abbrv. o.t.). Kubiak et al. (2009) define a class of graph called *divide-and-conquer* and propose a polynomial algorithm.

When the problem is parameterized by the width of the precedence graph, Bevern et al. (2016) prove that the problem with 2 machines is W[2]-hard. When it comes to total completion time, the algorithm by Hu (1961) is still optimal for out-tree, while Huo Leung (2006) illustrate that it is no longer optimal for in-trees. To solve $P|p_j = 1$,i.t.,pmtn$|\sum C_j$, Baptiste et al. (2004) propose an optimal algorithm in $O(n^m)$, which is polynomial for fixed m. These algorithms will be presented in details later. With an arbitrary m, the problem is recently closed by Wang and Bellenguez (2019a) by proving its NP-hardness.

When preemption is allowed, Muntz and Coffman (1970) gives a polynomial algorithm for $P|p_j = 1$,o.t.,pmtn$|C_{\max}$. Soper and Strusevich (2019) consider a special case of minimizing the makespan, where schedules have at most one preemption, separately for different numbers of machines. But for total completion time, little result has been achieved until (Brucker et al. 2001) solved $P|p_j = 1$,o.t.,pmtn$|\sum C_j$ in polynomial time. Coffman et al. (2003, 2012) study the existence of the ideal schedule, which minimizes both maximum and total completion time simultaneously, for preemptive and non-preemptive problems with in-trees precedence constraints. There exist also studies on bounds of structural parameters of preemptive schedules, such as the minimum number of preemptions, and a tight lower bound on shifts (Coffman et al. 2003; Chen et al. 2016).

Nevertheless, the problem $P|p_j = 1$,i.t.,pmtn$|\sum C_j$ remains open, to the best of our knowledge. Similarly, to minimize makespan for level-orders, i.e., $Pm|p_j = 1$,opposing-forest,pmtn$|C_{\max}$, is optimally solved in polynomial time by Dolev Warmuth (1984). Wang Bellenguez-Morineau (2019b) prove that with arbitrary number of machines, the problem is strongly NP-hard no matter for total completion time or for makespan. The complexity of problem $Pm|p_j = 1$,l.o.$|\sum C_j$ studied in this paper is still open.

Preliminaries

Before we start with any problem, we first clarify the terminology throughout this paper. A job is *final* if it has no successor. A job is *initial* if it has no predecessor. Specifically, a job is *final in* a job set when this job does not have any successor in this job set, and that the job itself is in this job set. A schedule is *non-idle* if all machines are busy all the time before the end of the schedule; a schedule is active

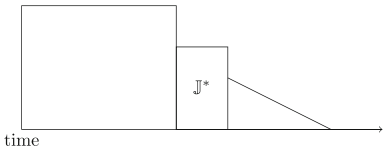


Fig. 2 Profile of an optimal schedule

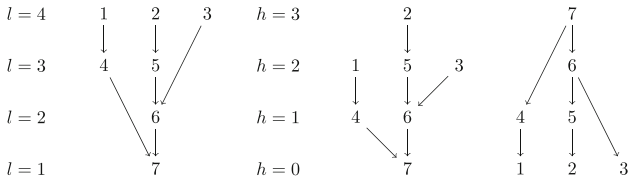


Fig. 3 Level, height and height of reverse graph

if it is not possible to construct another schedule, through changes in the order of processing on the machines, with at least one operation finished earlier and no operation finished later (Pinedo 2016, page 24, Definition 2.3.3). As the three problems discussed in this paper have the same objective function, total completion time, in the remainder of the text, unless otherwise stated, *optimal* stands for *optimal for total completion time*. Also, the propositions and lemmas in each section yield only for the specific scheduling problem discussed in that section.

3 $Pm|p_j = 1,\text{i.t.}|\sum C_j$

Obviously, in-trees with at most m initial jobs can be optimally scheduled in linear time. This trivial case will not be discussed. We assume that there are always more than m initial jobs. We briefly sketch the work of Hu (1961) and Baptiste et al. (2004).

Hu (1961) define the *height* of job j, denoted by $h(j)$, as the length of a longest chain from j to its final successor(s). Hu's algorithm, which iteratively executes the initial jobs with the largest height, is proved optimal for makespan.

On the other hand, Baptiste et al. (2004) show that the Gantt chart of an optimal schedule always takes the shape shown in Fig. 2: The first slot with less than m jobs, denoted by $\mathbb{J}^*$, separates the schedule into 2 parts. An optimal schedule can be *restored* from the given $\mathbb{J}^*$: The non-idle sub-schedule of the part of jobs executed before $\mathbb{J}^*$ can be obtained by Hu's algorithm, and any active sub-schedule is optimal for the part of jobs executed after $\mathbb{J}^*$, which are the successors of $\mathbb{J}^*$. One job set is called an *optimal set* if it restores an optimal schedule. To obtain an optimal set, Baptiste et al. (2004) propose to check all sets of at most $m-1$ jobs. That way, solving $Pm|p_j = 1,\text{i.t.}|\sum C_j$ consists of two steps: (1) traverse all candidate set for $\mathbb{J}^*$; (2) Restore a schedule from a given $\mathbb{J}^*$. Their complexities are, respectively, $O(n^{m-1})$ and $O(n)$. Accordingly, in Sect. 3.1, we study another way to restore the schedule. Strictly speaking, it checks whether there exists a non-idle sub-schedule for the first parts in $O(h)$. In 3.2, we show the theoretical basis of reducing the search space from $O(n^{m-1})$ to $O(h^m)$.

3.1 Lowest-level-last scheduling

Instead of Hu's algorithm, we introduce another method, which checks the existence of the non-idle schedule in lower complexity.

We first define the *level* of a job j, denoted by $l(j)$. It is another way to describe the position of a job in the precedence graph: $h+1$ minus the length of the longest chain from an initial job to j, remind that h is the graph height. See Fig. 3. To avoid ambiguity, we specify that the terms *high* and *low* describe $l(j)$, not $h(j)$. We define the *total levels* of a job set $\mathbb{J}$ as $\sum_{j\in\mathbb{J}} l(j)$.

The lowest-level-last rule, in one word, iteratively fills the *last* time slot with the m *lowest* final jobs from the unscheduled job set. In Fig 3, we see that the lowest level of the original graph means the largest height of the reverse graph. This rule is equivalent to Hu's algorithm performed on the reverse graph. As the minimum makespans of the original graph and the reversed graph are equal, this rule produces a non-idle schedule too, if there exists one. More importantly, we have

Proposition 1 *The existence of a non-idle schedule can be checked in* $O(h)$.

Proof We only discuss the case where n is an integer multiple of m. We perform lowest-level-last for h iterations, which is in $O(h)$ as m is fixed. Obviously, if any machine idles anytime during these iterations, then there cannot be any non-idle schedule. We now prove that, without the need of further check, one can immediately claim that there exists a non-idle schedule if there is no idleness in the last h time slots after h iterations.

Note that the precedence graph may consist of more than one components with different heights. If there are less than m jobs on level $l = 1$, then all these jobs are scheduled in the last time slot by the first iteration and removed from the unscheduled job set. Similarly, each iteration removes one lowest level with less than m jobs. As there are at most $h-1$ levels which contain less than m jobs, before the hth iteration, there must be one iteration, at which the lowest level contains at least m jobs. Afterwards, see Fig. 4 for an example, before all jobs are on the same level, there will *always* be more than m final jobs on the lowest two levels. This means that the schedule is non-idle.

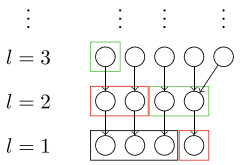


Fig. 4 Non-idle schedule checking. $m = 3$, the lowest final jobs in the last (second last, third last) slot are in black (red, green) rectangle. They are all on the lowest or second lowest levels

In a non-idle schedule produced by lowest-level-last, we observe the following lemma for later use:

Lemma 1 *Let $\mathbb{J}_k$ be the jobs executed in the first k slots. For any two jobs j_1, j_2 final in $\mathbb{J}_k$ such that $l(j_1) < l(j_2)$, we have $C_{j_1} \geq C_{j_2}$.*

3.2 Further analysis: lowest optimal schedule

We call the optimal sets with minimum total levels among all optimal sets the *lowest optimal sets*. Then, the optimal schedules restored from these optimal sets are called the *lowest optimal schedules*. We use lowest-level-last instead of Hu's algorithm to restore the non-idle sub-schedule. In this section, we analyze the properties of the lowest optimal schedules for the theoretical basis of the algorithm.

We first take a deeper look of one lowest optimal schedule, as shown in Fig. 5. We denote the set of jobs in the previous slot of $\mathbb{J}^*$ by $\mathbb{J}^{'}$, denote the jobs before $\mathbb{J}^{'}$ by $\mathbb{J}^1$ and denote the jobs after $\mathbb{J}^*$ by $\mathbb{J}^2$. We partition $\mathbb{J}^*$ into $\mathbb{A}^*$ and $\mathbb{B}^*$: each job in $\mathbb{B}^*$ has at least one predecessor in $\mathbb{J}^{'}$, and jobs in $\mathbb{A}^*$ do not. Reversely, we denote the predecessors of $\mathbb{B}^*$ in $\mathbb{J}^{'}$ by $\mathbb{B}^{'}$, and then $\mathbb{A}^{'} := \mathbb{J}^{'} \setminus \mathbb{B}^{'}$. When there is an ambiguity, we use $(\mathbb{J}^*)_{\mathcal{S}}$ to represent the set $\mathbb{J}^*$ of the specific lowest optimal schedule $\mathcal{S}$ and similar notations apply to other job sets.

Rather than directly searching an optimal set, as the first step, we search the $\mathbb{J}^* \cup \mathbb{A}^{'}$ of a lowest optimal schedule because its structural properties (Proposition 2) can be exploited in an algorithm. To obtain $\mathbb{J}^* \cup \mathbb{A}^{'}$, we search for the jobs in $\mathbb{J}^* \cup \mathbb{A}^{'}$ one by one from low to high. We sort the jobs by levels, such that $\mathbb{J}^* \cup \mathbb{A}^{'} = \{j_1, j_2, \ldots, j_k\}$. Suppose the first i jobs $\{j_1, j_2, \ldots, j_i\}$, where $i < k$, are already known. Then the key problem is where to find the *next lowest job* j_{i+1}, i.e., $i+1$th lowest job.

For simplicity, we denote the first lowest jobs $\{j_1, j_2, \ldots, j_i\}$ by $\mathbb{L}$, and the next lowest job j_{i+1} by j. It is obvious that j cannot be any predecessor of $\mathbb{L}$. Thus, j should be chosen from a *pruned tree* of the precedence graph where $\mathbb{L}$ and its predecessors are removed. Moreover, we denote the set of jobs on the same level as j in this pruned tree by $\mathbb{S}^j$ and we update the level $l(j)$ of each job j. We have:

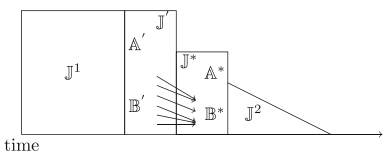


Fig. 5 Profile of a lowest optimal schedule

Proposition 2 *For any lowest optimal schedule $\mathcal{S}$,*

1. *$\forall j \in \mathbb{J}^* \cup \mathbb{A}^{'}$, if $|\mathbb{S}^j| \geq 2m$, then j is replaceable with any $j^{'} \in \mathbb{S}^j$. Formally, $\forall j^{'} \in \mathbb{S}^j$, there exists a lowest optimal schedule $\mathcal{S}^{'}$, of which $(\mathbb{J}^* \cup \mathbb{A}^{'})_{\mathcal{S}^{'}} = (\mathbb{J}^* \cup \mathbb{A}^{'})_{\mathcal{S}} \setminus \{j\} \cup \{j^{'}\}$.*
2. *if $|(\mathbb{J}^* \cup \mathbb{A}^{'})_{\mathcal{S}}| > m$, then there exists a lowest optimal schedule $\mathcal{S}^{'}$, such that $(\mathbb{J}^*)_{\mathcal{S}^{'}} \subseteq \mathbb{J}^m$, where set $\mathbb{J}^m$ is the set of the m lowest jobs of $(\mathbb{J}^* \cup \mathbb{A}^{'})_{\mathcal{S}}$.*

Remark The first statement narrows the scope of searching j from n to $2mh$ candidates: there are at most $2m$ candidates on each level. The second statement enables us to stop the search at the mth lowest jobs.

In the remainder of this section, we prove this proposition. If not specifically indicated, our discussion is dedicated to job j and job sets of $\mathcal{S}$ mentioned in Proposition 2. We start with several lemmas.

Lemma 2 *If $|\mathbb{S}^j| \geq 2m$, then $\mathbb{S}^j \cap \mathbb{J}^1 \neq \emptyset$.*

Proof Suppose $\mathbb{S}^j \cap \mathbb{J}^1 = \emptyset$. We can partition $\mathbb{S}^j$ into 3 parts: $\mathbb{S}^j_{\mathbb{J}^{'}} = \mathbb{S}^j \cap \mathbb{J}^{'}$, $\mathbb{S}^j_{\mathbb{J}^*} = \mathbb{S}^j \cap \mathbb{J}^*$ and $\mathbb{S}^j_{\mathbb{J}^2} = \mathbb{S}^j \cap \mathbb{J}^2$. As each job in $\mathbb{S}^j_{\mathbb{J}^2}$ must have a predecessor in $\mathbb{J}^*$, we denote these predecessors by $\mathbb{P}_{\mathbb{J}^*}$ and we have $|\mathbb{P}_{\mathbb{J}^*}| \geq |\mathbb{S}^j_{\mathbb{J}^2}|$. Considering $\mathbb{S}^j_{\mathbb{J}^*}$ and $\mathbb{P}_{\mathbb{J}^*}$ are not on the same level,

$$\begin{aligned} |\mathbb{S}^j| &= |\mathbb{S}^j_{\mathbb{J}^{'}}| + |\mathbb{S}^j_{\mathbb{J}^*}| + |\mathbb{S}^j_{\mathbb{J}^2}| \\ &\leq |\mathbb{S}^j_{\mathbb{J}^{'}}| + (|\mathbb{S}^j_{\mathbb{J}^*}| + |\mathbb{P}_{\mathbb{J}^*}|) \\ &\leq |\mathbb{J}^{'}| + |\mathbb{J}^*| \\ &< 2m \end{aligned}$$

□

Lemma 3 *If $|\mathbb{S}^j| \geq 2m$, then $j \notin \mathbb{B}^*$.*

Proof Suppose by contradiction that $|\mathbb{S}^j| \geq 2m$ and $j \in \mathbb{B}^*$. According to Lemma 2, $\exists j^{'} \in \mathbb{S}^j \cap \mathbb{J}^1$. Let one predecessor of j in $\mathbb{B}^{'}$ be j^{p}, we have $l(j^{'}) = l(j) < l(j^{\mathrm{p}})$. As $j^{'} \in \mathbb{J}^1$ while $j^{\mathrm{p}} \in \mathbb{J}^{'}$, by Lemma 1, $j^{'}$ is not final in $\mathbb{J}^1 \cup \mathbb{J}^{'}$, i.e., it must have a successor j^{s} in $\mathbb{J}^{'}$ which is strictly lower than j. No matter $j^{\mathrm{s}} \in \mathbb{A}^{'}$ or $j^{\mathrm{s}} \in \mathbb{B}^{'}$, we find a successor of $j^{'}$ in $\mathbb{L}$, the set of jobs strictly lower than j in $\mathbb{J}^* \cup \mathbb{A}^{'}$. This contradicts $j^{'} \in \mathbb{S}^j$, where the predecessors of $\mathbb{L}$ are removed. □

Lemma 4 *If $|\mathbb{S}^j| \geq 2m$ and $j \in \mathbb{A}^*$, then j is replaceable with any final job $j^* \in \mathbb{J}^1$, such that $l(j^*) \leq l(j)$.*

Proof We prove this lemma by building a lowest optimal schedule $\mathcal{S}^{'}$. Let the last predecessor of j, if exists, be j^{p}.

Because $j \in \mathbb{A}^*$, we have $j^{\mathrm{p}} \in \mathbb{J}^1$. As $l(j^{\mathrm{p}}) > l(j^*)$, and both j^{p} and j^* are final in $\mathbb{J}^1$, by Lemma 1, we have $C_{j^{\mathrm{p}}} \leq C_{j^*}$.

When $C_{j^{\mathrm{p}}} < C_{j^*}$ or j^{p} does not exist, we build $\mathcal{S}^{'}$ by interchanging j^* and j in $\mathcal{S}$. Note that the interchange does not break the precedence constraints. Obviously, $\mathcal{S}^{'}$ is lowest optimal.

When $C_{j^{\mathrm{p}}} = C_{j^*}$, j^{p} and j^* are executed in the same time slot in $\mathbb{J}^1$, let it be the kth time slot. We reschedule jobs in the first k time slots to obtain another lowest optimal schedule $\mathcal{S}^{''}$: As j^{p} and j^* are final in $\mathbb{J}^1$, by Lemma 1, all jobs in the $k+1$th time slot are not higher than j^*. Thus, at the first iteration when the jobs in the first k time slots are rescheduled by lowest-level-last, there are at least m final jobs which are not higher than j^{p}. We are free to select the m lowest jobs which include j^* but do not include j^{p} to fill the last time slot. After rescheduling the previous k time slots, another lowest optimal schedule $\mathcal{S}^{''}$ is obtained. Note that $(\mathbb{J}^* \cup \mathbb{J}^{'})_{\mathcal{S}^{''}} = (\mathbb{J}^* \cup \mathbb{J}^{'})_{\mathcal{S}}$, but now in $\mathcal{S}^{''}$ we have $C_{j^{\mathrm{p}}} < C_{j^*}$. Similarly, we build the lowest optimal schedule $\mathcal{S}^{'}$ by interchanging j^* and j in $\mathcal{S}^{''}$.

In both cases, we built a lowest optimal schedule $\mathcal{S}^{'}$ where j is replaced by j^* and the replaceability. □

Lemma 5 *If $j \in \mathbb{A}^{'}$, then j is replaceable with any job $j^{'} \in \mathbb{S}^j \cap \mathbb{J}^1$.*

Proof First, $j^{'}$ is final. Otherwise, according to Lemma 1, its final successor in $\mathbb{J}^1 \cup \mathbb{J}^{'}$ which is strictly lower than j will be in $\mathbb{J}^{'}$. Thus, $j^{'}$ has a successor in $\mathbb{L}$. This is a contradiction to $j^{'} \in \mathbb{S}^j$.

Then, the construction of $\mathcal{S}^{'}$ will be quite similar with the one in the proof for Lemma 4. We reschedule $\mathbb{J}^1 \cup \mathbb{J}^{'}$ in $\mathcal{S}$ by lowest-level-last. At the first iteration, to fill the slot of $(\mathbb{J}^{'})_{\mathcal{S}^{'}}$, we are free to choose $(\mathbb{J}^{'})_{\mathcal{S}} \setminus \{j\} \cup \{j^{'}\}$ as the m lowest jobs because $l(j^{'}) = l(j)$. Hence, we have $(\mathbb{J}^* \cup \mathbb{A}^{'})_{\mathcal{S}^{'}} = (\mathbb{J}^* \cup \mathbb{A}^{'})_{\mathcal{S}} \setminus \{j\} \cup \{j^{'}\}$ □

We are ready to prove the proposition.

Proof. for Proposition 2 To prove the first statement where $|\mathbb{S}^j| \geq 2m$, by Lemma 3, we only need to discuss $j \in \mathbb{A}^*$ or $j \in \mathbb{A}^{'}$.

Case 1: $j \in \mathbb{A}^*$. In this case, $\mathbb{S}^j \cap \mathbb{J}^{'} = \mathbb{S}^j \cap \mathbb{J}^* = \emptyset$. We first show that j is replaceable with jobs in $\mathbb{S}^j \cap \mathbb{J}^1$, then we show that $\mathbb{S}^j \cap \mathbb{J}^2 = \emptyset$.

Let $j^{'}$ be any job in $\mathbb{S}^j \cap \mathbb{J}^1$. If $j^{'}$ is not final in $\mathbb{J}^1$, then by Lemma 4, j can be replaced by the final successor of $j^{'}$ in $\mathbb{J}^1$. However, the replacement will lower the total levels, which contradicts the fact that $\mathcal{S}$ is lowest optimal. So $j^{'}$ is final. Again by Lemma 4, j is replaceable with $j^{'}$.

In fact, by Lemma 4, all jobs in $\mathbb{A}^*$ are on the same level because any job in $\mathbb{A}^*$ higher than j is replaceable with $j^{'}$, which must exist according to Lemma 2. This fact leads to $\mathbb{S}^j \cap \mathbb{J}^2 = \emptyset$. If there exists a job in $\mathbb{S}^j \cap \mathbb{J}^2$, then it has a predecessor j^{p} in $\mathbb{J}^*$. As $l(j^{\mathrm{p}}) > l(j)$ and all jobs in $\mathbb{A}^*$ are on the same level, we have $j^{\mathrm{p}} \in \mathbb{B}^*$. So j^{p} has a direct predecessor in $\mathbb{J}^{'}$, denoted by j^{pp}. We have $l(j^{\mathrm{pp}}) > l(j^{\mathrm{p}}) > l(j) = l(j^{'})$ but $C_{j^{\mathrm{pp}}} > C_{j^{'}}$, which contradicts Lemma 1.

Case 2: $j \in \mathbb{A}^{'}$. Given by Lemma 5, j is replaceable with any job in $\mathbb{S}^j \cap \mathbb{J}^1$. Similarly, we show that there is no job in $\mathbb{S}^j \cap \mathbb{J}^2$. If any, this job's predecessor in $\mathbb{J}^*$ is strictly higher than j but replaceable with j, which contradicts the minimum total levels of $\mathbb{J}^*$.

We succeed in proving the first statement. Now we turn to the second statement. As $|(\mathbb{J}^* \cup \mathbb{A}^{'})_{\mathcal{S}}| > m$, neither $\mathbb{A}^*$ nor $\mathbb{A}^{'}$ is empty. Then jobs in $\mathbb{B}^*$ are strictly lower than jobs in $\mathbb{A}^*$ or $\mathbb{A}^{'}$, because their predecessors cannot be higher than the predecessors of jobs in $\mathbb{A}^*$. We just proved that they are replaceable with any job on the same level in $\mathbb{A}^{'}$. Moreover, due to the lowest optimality, jobs in $\mathbb{A}^*$ are not higher than any job in $\mathbb{A}^{'}$ either. Thus, by replacing jobs in $\mathbb{A}^*$ with jobs in $\mathbb{A}^{'}$, we can build a lowest optimal schedule $\mathcal{S}^{'}$ where $\mathbb{J}^*$ is a subset of the m lowest job of $\mathbb{J}^* \cup \mathbb{A}^{'}$. □

3.3 The algorithm

When $\mathbb{L}$ is known, Proposition 2 enables us to search the next lowest job j of $\mathbb{J}^* \cup \mathbb{A}^{'}$ in a smaller space, which is realized by Algorithm 1. Accordingly, Algorithm 2 finds all candidates sets.

Algorithm 1: Find the next lowest job j

```
Input: Chosen lowest jobs 𝕃
Output: 𝕏 such that j ∈ 𝕏
Calculate the pruned tree by removing jobs in 𝕃 and their
predecessors;
foreach k = 1, ..., h do
    Let 𝕏_k be the set of jobs on the kth level in the pruned tree;
    if |𝕏_k| < 2m then
        Add all jobs in 𝕏_k to 𝕏;
    end
    if |𝕏_k| ≥ 2m then
        Arbitrarily pick a job in 𝕏_k and add it to 𝕏;
    end
end
```

The space complexity of Algorithm 2 is $O((2hm)^m 2^m)$, which is $O(h^m)$ when m is a constant. Note that Algorithm 1 involves tree-pruning, which takes linear time, so the time complexity of Algorithm 2 is $O(h^m n)$.

To recognize the optimal set, we check the existence of the non-idle schedule of $\mathbb{J}^1 \cup \mathbb{J}^{'}$ in $O(h)$ time, as shown by Proposition 1. Note that the total completion time of a non-idle schedule of km jobs is $\frac{km(1+k)}{2}$, which can be calculated in $O(1)$. Since $\mathbb{J}^2$ contains less than hm jobs, the active schedule of $\mathbb{J}^2$ and its total completion time can be obtained in $O(h)$ too. Finally, we have:

Algorithm 2: Get candidate sets

Output: $\mathcal{C}$: candidate choices of an optimal set $\mathbb{J}^*$

```
/* C_k: the candidate choices of J^m, where k
   is the cardinality                          */
```

1 $\mathcal{C}_0 \leftarrow \{\emptyset\}$;
2 **foreach** $k = 1, \ldots, m$ **do**
3 $\mathcal{C}_k \leftarrow \emptyset$;
4 **foreach** $\mathbb{L} \in \mathcal{C}_{k-1}$ **do**
5 Applying Algorithm 1, calculate $\mathbb{X}$ according to $\mathbb{L}$;
6 **foreach** $j \in \mathbb{X}$ **do**
7 Add $\{j\} \cup \mathbb{L}$ to $\mathcal{C}_k$;
8 Add all subsets of $\{j\} \cup \mathbb{L}$ to $\mathcal{C}$;
9 **end**
10 **end**
11 **end**

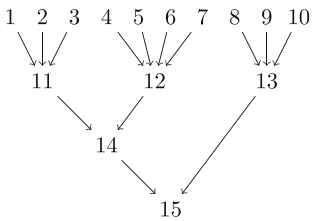


Fig. 6 Example

Theorem 1 *$Pm|p_j = 1, i.t.|\sum C_j$ can be solved in $O(h^{m+1}n)$ time and $O(h^{m+1})$ space.*

3.3.1 Example

We hereby illustrate how the algorithm restores candidate sets on the in-trees example shown in Fig. 6 with $m = 3$ machines.

Whenever there are more than $2m = 6$ jobs on the first level of the pruned tree, we choose the job with the smallest label. For example, at the first iteration of Line 2 in Algorithm 2, we choose job 1 as the lowest job on the first level. As other levels contain at most 3 jobs, they are all added. Thus, after the first iteration, $\mathcal{C}_1 = \{1, 11, 12, 13, 14, 15\}$. At the second iteration, for instance, when $\mathbb{L} = \{14\}$, the first level of the pruned tree is $\{8, 9, 10\}$, which is now less than $2m$ jobs. So, they are all added. The search tree built by algorithm 2 is illustrated in Fig. 7 where k is the cardinality.

After traversing all subsets of candidate $\mathbb{J}^m$, there are totally 46 subsets listed below. It is much less than $\binom{2}{15} + \binom{1}{15} = 225$, required by the brute-force search proposed by Baptiste et al. (2004). We list all the candidate sets in the appendix.

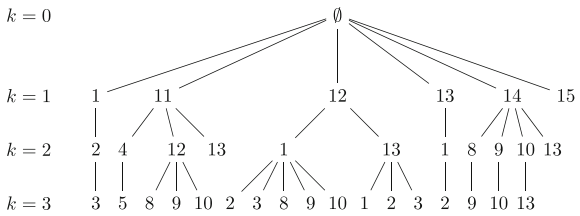


Fig. 7 Search tree of $\mathbb{J}^m$

Table 2 Average height of instances

n	35	45	55	65	75	85	95	105	115
h	4.3	5.1	5.2	6.3	7.3	7.4	7.9	8.5	8.6

3.4 Experiments

The algorithm has lower time and space complexity than the brute-force search proposed by Baptiste et al. (2004), which suggests that it runs faster when $n \to \infty$. However, it involves extra calculation in each step, such as tree-pruning. It is still meaningful to see the numerical comparison on small instances.

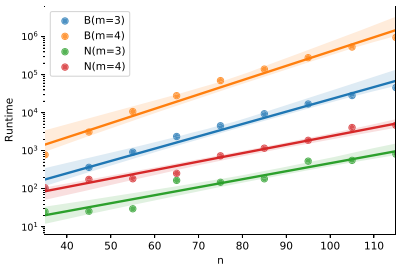
We implement the two algorithms and test them on randomly generalized in-trees instances. The instances are the sub-trees of the first n nodes generated by a Galton–Watson branching process, where the number of offspring is uniformly distributed in $\{0, 1, \ldots, n-1\}$. Note that this generator can output any specific tree form with non-zero probability.

The test environment is: Intel(R) Core(TM) i7-8550U at 1.80GHz with 16GB RAM. For each number n of jobs and each number m of machines, we create 10 different instances with average height reported in Table 2.

We compare the runtime to solve the instance by two algorithms in Fig. 8a with different n, m pair, and record the runtime of our algorithm for different n but same $m = 3$ in Fig. 8b. Note that the runtime is shown in log scale. Both algorithms explode with n, while the new algorithm performs considerably better, even with small instances. We also find that to some extent, the randomness of instances affects the runtime.

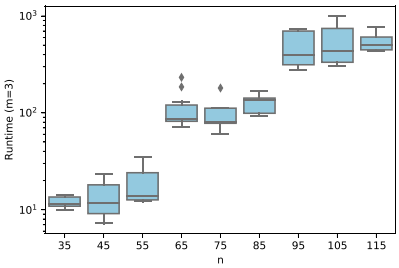
4 $Pm|p_j = 1,\text{l.o.}|\sum C_j$

Dolev Warmuth (1985) proposed a dynamic programming algorithm which solves the problem $P|p_j = 1, l.o.|C_{\max}$ in $O(m^q n^q)$, where q is the number of components of level-orders. Then, using the *reduction theorem* and properties of *median*, they prove that $Pm|p_j = 1, l.o.|C_{\max}$ can be solved in polynomial time. In this section, we study the problem $Pm|p_j = 1,\text{l.o.}|\sum C_j$, where the objective is total com-



(a) Comparison

B,N stand for Baptiste's algorithm and the new algorithm.



(b) Performance of new algorithm

Fig. 8 Experiment results

pletion time. We show that a similar dynamic programming algorithm in $O(m^q n^q)$ also works for minimizing $\sum C_j$. Then, based on the structure of an optimal schedule, we show that the problem $Pm|p_j = 1$,l.o.$|\sum C_j$ is polynomial solvable.

Let $\mathcal{S}$ be an optimal schedule for an instance of $Pm|p_j = 1, l.o.|\sum C_j$. Denote the makespan of $\mathcal{S}$ by t^*. $\mathbb{J}^\circ$ and $\mathbb{J}^k$ stands for the set of total jobs and the set of jobs in the kth time slot in $\mathcal{S}$ resp., where $k = 1, \ldots, t^*$.

Proposition 3 *The sub-schedule of $\mathcal{S}$ for jobs in $\mathbb{J}^\circ \setminus \mathbb{J}_1$ is an optimal schedule for the sub-instance of jobs in $\mathbb{J}^\circ \setminus \mathbb{J}_1$.*

This proposition is obvious. If the sub-schedule is not optimal, then replacing the sub-schedule of $\mathcal{S}$ with an optimal one will reduce the total completion time, which contradicts the optimality of $\mathcal{S}$. Based on Proposition 3, the dynamic programming algorithm, Algorithm 3, finds an optimal schedule.

Remark Line 5 only searches subsets with different allocations, where *allocation* means the numbers of jobs picked from components. Remind that a component in level-orders graph is a maximal weakly connected sub-graph, as presented in Sect. 1. Thus, the complexity of this step is not $O(2^n)$, but $O(m^q)$. To justify, using the term defined previously, the initial jobs in the same component are *replaceable* with each other, and the candidates with the same allocation are equivalent [Corrollary 4.1]Dolev Warmuth 1985. Line 6 and 11 require calculation of optimal sub-schedules. They are recursive calls of Algorithm 3 itself. To see the complexity of the

Algorithm 3: Solving $Pm|p_j = 1, l.o.|\sum C_j$

```
1  if the number of initial jobs is at most m then
2  |  J_1 ← set of all initial jobs;
3  end
4  if there are more than m initial jobs then
   |  /* allocation : the numbers of jobs
   |     picked from components                */
   |  /* X : the candidate choices of J_1      */
5  |  foreach X of initial jobs with different allocations do
6  |  |  if the optimal schedule of J° \ X has minimum ΣC_j
   |  |  among all the X then
7  |  |  |  J_1 ← X;
8  |  |  end
9  |  end
10 end
11 Concatenate J_1 with the optimal schedule of J° \ J_1;
```

dynamic programming algorithm, we investigate its look-up table. It contains all possible cases of sub-problems represented by tuple $(n_1, n_2, \ldots, n_q)$, where n_i is the number of jobs in the ith component for $i = 1, \ldots, q$. As $n_i \leq n$, the size of the look-up table is then n^q, and the total complexity is $O(m^q n^q)$.

So when $q \leq m$, Algorithm 3 solves $Pm|p_j = 1, l.o.|\sum C_j$ in polynomial time. Now we are interested in the case where $q > m$.

When $q > m$, $\mathcal{S}$ starts with at least one non-idle slot. We apply the same idea of solving $Pm|p_j = 1, i.t.|\sum C_j$: If $\mathcal{S}$ itself is a non-idle schedule, it can be found in polynomial time because it is optimal for makespan too. Otherwise, let its first non-idle time slot with less than m jobs be $\mathbb{J}^*$. Then, the jobs after $\mathbb{J}^*$ are its successors, and the rest jobs are executed before $\mathbb{J}^*$. Given $\mathbb{J}^*$ of $\mathcal{S}$, an optimal schedule can be restored: the non-idle sub-schedule before $\mathbb{J}^*$ can be obtained polynomially (Dolev Warmuth 1985); the jobs after $\mathbb{J}^*$ can be optimally solved by Algorithm 3 in polynomial time too, because they form a sub-graph of less than m components. As the brute-force traversal of $\mathbb{J}^*$ runs in $O(n^m)$, we have

Theorem 2 *$Pm|p_j = 1, l.o.|\sum C_j$ can be solved in polynomial time.*

5 $P|p_j = 1$,i.t.,pmtn$|\sum C_j$

We show the strong NP-hardness of $P|p_j = 1,\text{i.t.,pmtn}|\sum C_j$ by a reduction from 3- PARTITION. A formulation of a 3- PARTITION instance is: Consider a set α of $3q$ elements. Each element $a \in \alpha$ has an integer weight w_a with $\frac{b}{2} \geq w_a \geq \frac{b}{4}$, where integer $b := \frac{\sum_{a\in\alpha} w_a}{q}$. The decision question asks if α has a partition, i.e., $\alpha_1, \ldots, \alpha_q$ such that $\sum_{a\in\alpha_i} w_a = b$, $i = 1, \ldots, q$. From this 3- PARTITION instance, we will build an instance of $P|p_j = 1$,i.t.,pmtn$|\sum C_j$ and show their equivalence.

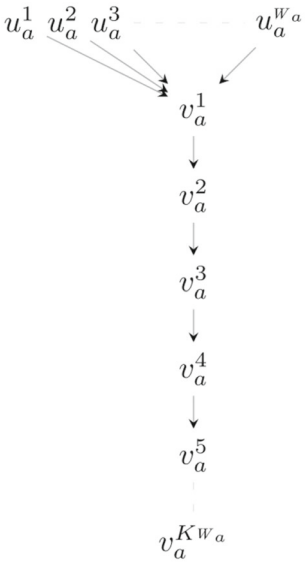


Fig. 9 U-jobs and v-jobs

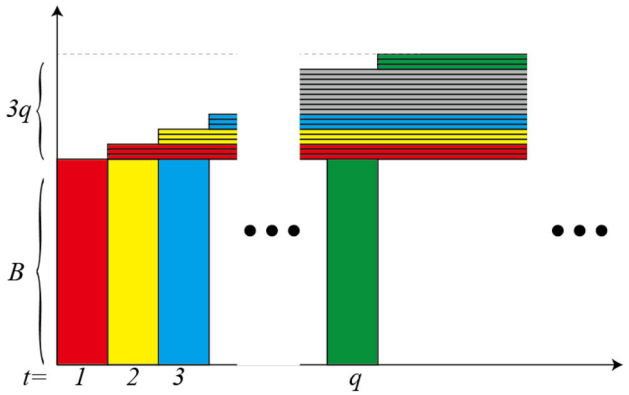


Fig. 10 Gantt chart of an acceptable schedule. The horizontal and vertical axes represent time and machine resp. Each color block contains the jobs in the three trees corresponding to a subset of the partition. As is shown, in the first time slot, B machines execute the u-jobs of α_1 and $3q$ machines idle. In the second time slot, B machines execute the u-jobs of α_1, while 3 other machines start the v-chain of α_1 and the rest $3q-3$ machines idle, etc.

To begin with, define some constants: multiplier $d := q^{20}b^2$; weights $W_a := w_a d$, $B := bd$; large number $K := B^5$; number of machines $m := B + 3q$ and objective value $\sigma := \sum_{a\in\alpha}\sum_{i=1}^{KW_a} i + (1+K)\sum_{i=1}^{q} iB$.

We build jobs and their precedence constraints. For each element $a \in \alpha$, we build a chain of $W_a K$ jobs. The jobs in this chain are named v-jobs: $v_a^1, \ldots, v_a^{W_a K}$. Then we build W_a u-jobs: $u_a^1, \ldots, u_a^{W_a}$. They precede v_a^1, and form an in-tree component, see Fig. 9. The decision question asks if there exists an *acceptable* schedule, i.e., a schedule whose total completion time is at most σ. Note that the reduction is pseudo-polynomial (polynomial to b) because of the strong NP-hardness of 3- PARTITION.

In the remainder of this section, we show there exists an acceptable schedule if and only if α has a partition. We show how to build an acceptable schedule with a partition in 5.1 and find a partition from an acceptable schedule in 5.2.

5.1 3-PARTITION $\Longrightarrow$ $P|p_j = 1, \text{i.t.}, \text{pmtn}|\sum C_j$

We build a schedule from a partition, then we show it is acceptable, i.e., $\sum C_j \le \sigma$.

For each α_i, we execute the three corresponding in-tree components from the i^{th} slot without any interruption. There are B u-jobs in one slot. As at most $3q$ v-jobs can be executed in parallel, the schedule respects the number of machines. See the a Gantt chart in Fig. 10.

Clearly, the schedule is not optimal, yet acceptable: Let $\mathbb{U}^\circ$ ($\mathbb{V}^\circ$) be the set of all u-jobs (v-jobs). We compute $\sum C_j$ separately for $\mathbb{U}^\circ$ and $\mathbb{V}^\circ$, and define them as $\sigma_{\mathbb{U}^\circ}$ and $\sigma_{\mathbb{V}^\circ}$:

$$\begin{aligned}\sigma_{\mathbb{U}^\circ} &:= \sum_{u\in\mathbb{U}^\circ} C_u = \sum_{k=1}^{q}\sum_{a\in\alpha_k}\sum_{i=1}^{W_a} C_{u_a^i} = \sum_{k=1}^{q}\sum_{a\in\alpha_k}\sum_{i=1}^{W_a} k \\ &= \sum_{k=1}^{q} k \sum_{a\in\alpha_k}\sum_{i=1}^{W_a} 1 = B\sum_{k=1}^{q} k \end{aligned} \tag{1}$$

$$\begin{aligned}\sigma_{\mathbb{V}^\circ} &:= \sum_{v\in\mathbb{V}^\circ} C_v = \sum_{k=1}^{q}\sum_{a\in\alpha_k}\sum_{i=1}^{W_a K} C_{v_a^i} \\ &= \sum_{k=1}^{q}\sum_{a\in\alpha_k}\sum_{i=1}^{W_a K}(C_{u_a^1} + i) = \sum_{k=1}^{q}\sum_{a\in\alpha_k}\sum_{i=1}^{W_a K}(k+i) \\ &= \sum_{k=1}^{q} k\sum_{a\in\alpha_k}\sum_{i=1}^{W_a K} 1 + \sum_{a\in\alpha}\sum_{i=1}^{W_a K} i \\ &= \sum_{k=1}^{q} kKB + \sum_{a\in\alpha}\sum_{i=1}^{W_a K} i \end{aligned} \tag{2}$$

We see at once $\sigma_{\mathbb{U}^\circ} + \sigma_{\mathbb{V}^\circ} = \sigma$.

5.2 $P|p_j = 1, \text{i.t.}, \text{pmtn}|\sum C_j$ $\Longrightarrow$ 3-PARTITION

Our reduction takes some similarity with that by Wang and Bellenguez (2019a) for the non-preemptive problem, where the authors also show how a non-preemptive acceptable schedule implies a partition. Intuitively, we may want to eliminate the preemptions in an acceptable schedule without increasing the total completion time, and then their result could be applicable to our problem. Unfortunately, preemption does improve the total completion time and sometimes cannot be eliminated.

However, this improvement does not have decisive influence. Preemptions can advance jobs to only a tiny extent, but

cannot change any non-acceptable schedule into an acceptable one without a partition.

We observe that the v-jobs dominate the total completion time as they are much more than u-jobs. The execution of a v-chain is decided by the completion time of its last predecessor. In an acceptable schedule of the non-preemptive problem, its last predecessor is executed in its previous time slot. Then, by the machine number constraint, we obtain a partition. Here, for the preemptive problem, the logic is the same. We need to (1) define a substitute of time slot, namely *big group*, where the machine number constraint is still subjected, (2) show that the execution of *the majority part of* each big group releases exactly three v-chains whose corresponding element in α forms a partition.

The first step requires that big groups are properly defined and executed like time slots, which is depicted by Proposition 4. To quantitatively specify the majority part of big group in the second step, we further introduce *small groups which are principally in* a big group. Then the second step is realized by Proposition 5. We leave their proofs later in Sect. 5.3.

We can no longer study, say, *the u-jobs in the* qth time slot, yet we can study *the last* m executed u-jobs. To put forward the definitions of groups, we first list the u-jobs by their completion time and name the list as L^{u}; then we list the v-jobs by their starting time and name the list as L^{v}. The u-jobs (v-jobs) with the same completion (starting) time can be arbitrarily arranged. Let $\#(j)$ return the sequence number of job j in its list, e.g., $\#(u) = 5$ if u is the fifth job in L^{u}. Clearly, each job is in either L^{v} or L^{u}.

For $u_1 \in \mathbb{U}^{\circ}$ directly precedes v_1, $u_2 \in \mathbb{U}^{\circ}$ directly precedes v_2, and $\#(u_1) < \#(u_2)$, we assume $\#(v_1) < \#(v_2)$. This is called *grouping assumption*, which can be realized by interchanging such u-job in $\mathcal{S}$ by limited times.

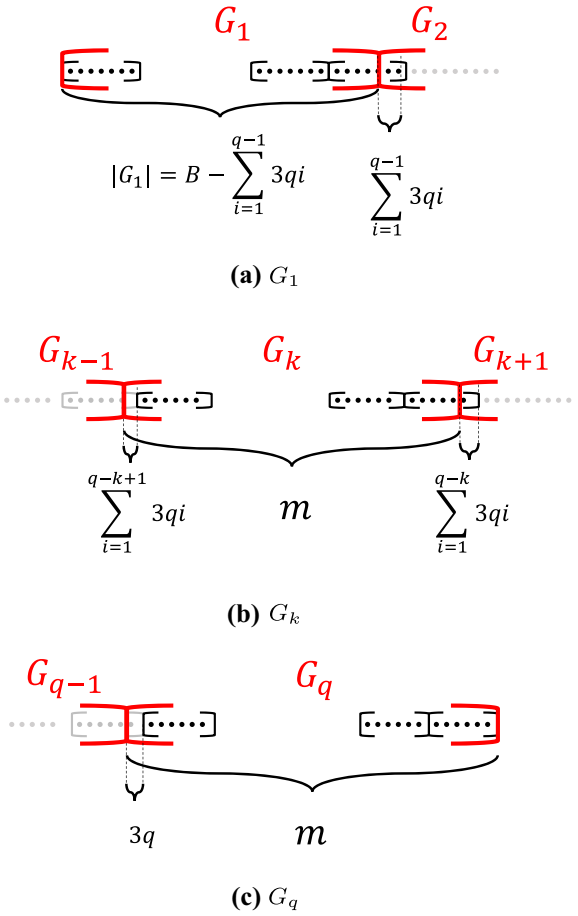
We cut L^{u} into q segments, such that the length of each segment equals to m except the first one. The *big group* G_i is defined as the set of jobs in the ith segment, $i = 1, \ldots, q$. Formally, $G_i := \{u | \#(u) \in [|G_1| + 1 + (i-2)m, |G_1| + (i-1)m]\}$ where $i = 2, \ldots, q$; and $G_1 := \{u | \#(u) \in [1, |G_1|]\}$ where $|G_1| := B - 3q \sum_{i=1}^{q-1} i$. The big groups are illustrated by big red braces in Fig. 11. The similarity of big groups and time slot is depicted by:

Proposition 4 *These two statements about* G_i *hold:*

S1 all jobs in G_i *are finished before* $t = i + i\delta$, *where* $\delta := \frac{q^8}{d}$;

S2 at least $|G_i| - d$ *u-jobs in* G_i *are finished after* $t = i - \Delta$, *where* $\Delta := \frac{q^{10}b}{d}$.

Proposition 4 gives a sketch of when jobs in G_i are executed. As stated above, due to the preemption, the jobs in G_i are not necessarily executed in the ith time slot, while Proposition 4 indicates that loosely speaking, they are *principally* in that time slot. Precisely, the completion time of the majority part (at least $|G_i| - d$) of jobs in G_i falls in a very narrow interval $[i + i\delta, i - \Delta]$, whose width $i\delta + \Delta$ is small because $d = q^{20}b^2$ is large in comparison with q and b.



(a) G_1

(b) G_k

(c) G_q

Fig. 11 L^{u}. Jobs are represented by dots and groups are enclosed in braces

If, ideally, each big group G_i corresponds to the u-jobs of three in-tree components, say, the in-tree components of a_1, a_2, a_3, then we will be able to find a partition by Proposition 4. We hereby give an informal proof: As G_i is *principally* in a time slot, and the capacity of each time slot is the number of machines $m = B + 3q$, the number of u-jobs of G_i, corresponding to a_1, a_2, a_3, is

$$W_{a_1} + W_{a_2} + W_{a_3} \approx B + 3q \tag{3}$$

Remind that $W_a = w_a d$, $B = bd$, and again, note that $d \gg q$, (3) is in fact

$$W_{a_1} + W_{a_2} + W_{a_3} = B \tag{4}$$

which is equivalent to $w_{a_1} + w_{a_2} + w_{a_3} = b$. Thus, a partition is found. Of course, this ideal assumption that each big group corresponds to the u-jobs of three in-tree components does

not hold all the time. Fortunately, there does exist a relation between group and in-tree component, which is revealed by Proposition 5.

To establish Proposition 5, we firstly we introduce small groups and formally define the terms *principally* above: cut L^{u} into qb equal-length segments and define *small group* g_i as the set of jobs in the ith segment, $i = 1, \ldots, qb$. Formally, $g_i := \{u | \#(u) \in [(i-1)d + 1, id]\}$ and $|g_i| = d$. The small groups are illustrated by small braces in Fig. 11. As a result of our grouping assumption, the jobs in the same small group are from the same in-tree component. Furthermore, we have: $C_u \leq C_u^{'} \leq C_u^{''}$, where $C_u^{'}$ is the completion time of the last u-job in the same small group with u, and $C_u^{''}$ is the completion time of the last u-job in the same component with u.

Since the cardinality of big groups are not integer multiple of small groups, we hereby define the relation: Small group $g_{i'}$ is said to be *principally in* big group G_i if $|g_{i'} \cup G_i| \geq d - q^4$, noted as $g_{i'} \hat{\subseteq} G_i$. There are always b small groups principally in a big group, and they are illustrated by dark black braces in Fig. 11.

With all the preliminaries above, the relation between big groups, small groups, and in-tree components is revealed by:

Proposition 5 *$\forall g_i \hat{\subseteq} G_{i'}$ and $\forall u \in g_i$, we have*

$$i^{'} - \Delta \leq C_u^{''} \leq i^{'} - \Delta + \frac{1}{3}$$

Note that u is not necessarily in $G_{i'}$. Proposition 4 merely depicts how a majority of jobs in a big group is executed, while the relation between big group and the in-tree component is unclear. Proposition 5 more precisely shows that the earliest start time of successor (v-job) of the corresponding in-tree component is in an interval, which associates big groups with in-tree components and allows us to find a partition. Define

$$\alpha_i := \left\{ a | C_{u_a^1}^{''} \in \left[i - \Delta, i - \Delta + \frac{1}{3} \right] \right\} \qquad i = 1, \ldots, q$$

Proposition 6 *$\alpha_1, \ldots, \alpha_q$ is a partition of α.*

Proof It is easy to check that $\sum_{a \in \alpha_i} w_a = b$ if and only if $\sum_{a \in \alpha_i} W_a = B$, and now we prove the latter for any $i = 1, \ldots, q$.

We denote the small groups which are principally in G_i by $\overline{G}_i$ and denote the corresponding u-jobs of α_i by U_i. Formally, $\overline{G}_i := g_{(i-1)b+1} \cup g_{(i-1)b+2} \cdots \cup g_{ib}$, $U_i := \{u_a^i | a \in \alpha_i; \forall i \leq w_a\}$.

As $\sum_{a \in \alpha_i} W_a = |U_i|$ and $|\overline{G}_i| = B$, it suffices to show $U_i = \overline{G}_i$. $U_i \supset \overline{G}_i$ is given by Proposition 5. To see $U_i \subseteq \overline{G}_i$, we only need to observe that $U_i \cap U_{i'} = \overline{G}_i \cap \overline{G}_{i'} = \emptyset$ for any $i \neq i^{'}$, and that $\bigcup_{i=1}^{q} U_i = \bigcup_{i=1}^{q} \overline{G}_i = \mathbb{U}^{\circ}$. □

We conclude

Theorem 3 *$P | p_j = 1, i.t., pmtn | \sum C_j$ is strongly NP-hard.*

5.3 Proofs for Propositions 4 and 5

Proposition 4 has two statements: S1 states that the jobs cannot be executed too late, and S2 states that the (majority part of) jobs cannot be executed too early. To prove S1, the deduction is natural: If the jobs are executed too late, then the objective function is violated. To this purpose, we focus on a property of an acceptable schedule deduced from the constraint of the objective function. Then we show that the property is violated without S1. This property is formally the inequality (9). To prove S2, we need to show that, the jobs cannot start too early because the machines are busy executing jobs in previous groups, and there is little space for G_i. This depends on how previous groups are executed, and a mathematical deduction is used. How the previous groups occupy the machines is quantified by (5).

Proposition 5 is an inequality about $C_u^{''}$, the earliest start time of the successor. This inequality is obtained from the fact that the v-chain is extremely long with a multiplier $K \gg B$. A small increment of $C_u^{''}$ can cause a large increase of the total completion time. In other words, the constraint of objective function can be converted to a constraint of $\sum_{u \in \mathbb{U}^{\circ}} C_u^{''}$, which is (6). Similarly, the proof for the right side of Proposition 5 shows that a single $C_u^{''} > i^{'} - \Delta + \frac{1}{3}$ causes a contradiction to (6), the constraint of the total $\sum_{u \in \mathbb{U}^{\circ}} C_u^{''}$. The left side of Proposition 5 can be directly deduced from S2 of Proposition 4.

Based on (5), (6) and (9), the proofs for Propositions 4 and 5 are essentially the relaxations of inequalities, where some intermediate sets such as X, G_k^A, G_k^B, G_k^{AB} are created. We start with the proofs for the inequalities, then we give the formal proofs for Propositions 4 and 5.

Let $\mathbb{U} \subseteq \mathbb{U}^{\circ}$ be a set of u-jobs and $C_{\mathbb{U}}^{'} := \max\{C_u | u \in \mathbb{U}\}$. The amount of work in $\mathbb{U}$ executed before time t^* is denoted by $T(\mathbb{U}, t^*)$, then

$$C_{\mathbb{U}}^{'} \geq t^* + 1 - \frac{T(\mathbb{U}, t^*)}{|\mathbb{U}|} \tag{5}$$

Proof After t^*, the number of machine working in parallel to execute $\mathbb{U}$ is at most $|\mathbb{U}|$. The total amount of work is: $|\mathbb{U}| \leq (C_{\mathbb{U}}^{'} - t^*)|\mathbb{U}| + T(\mathbb{U}, t^*)$, which implies $C_{\mathbb{U}}^{'} \geq t^* + 1 - \frac{T(\mathbb{U}, t^*)}{|\mathbb{U}|}$. □

$$\sum_{u \in \mathbb{U}^{\circ}} C_u \leq \sum_{u \in \mathbb{U}^{\circ}} C_u^{'} \leq \sum_{u \in \mathbb{U}^{\circ}} C_u^{''} \leq B \sum_{i=1}^{q} i + 1 \tag{6}$$

Proof For legibility, we denote the equivalent terms $C''_{u_a^1}, C''_{u_a^2}, \ldots, C''_{u_a^{W_a}}$ by C''_{u_a}. To simplify the expression of the total completion time, we denote the difference $D_{v_a^i} := C_{v_a^i} - i$ for each v_a^i. So, $\sum_{v\in\mathbb{V}^\circ} C_v = \sum_{v\in\mathbb{V}^\circ} D_v + \sum_{a\in\alpha}\sum_{i=1}^{KW_a} i$. As $\mathcal{S}$ is acceptable and $\sigma = \sigma_{\mathbb{U}^\circ} + \sigma_{\mathbb{V}^\circ}$, the total completion time of $\mathcal{S}$ is

$$\sum_{u\in\mathbb{U}^\circ} C_u + \sum_{v\in\mathbb{V}^\circ} D_v + \sum_{a\in\alpha}\sum_{i=1}^{KW_a} i \le \sigma_{\mathbb{U}^\circ} + \sigma_{\mathbb{V}^\circ} \tag{7a}$$

by (2),

$$\sum_{u\in\mathbb{U}^\circ} C_u + \sum_{v\in\mathbb{V}^\circ} D_v \le \sigma_{\mathbb{U}^\circ} + K\sum_{i=1}^{q} iB \tag{7b}$$

so

$$\sum_{v\in\mathbb{V}^\circ} D_v \le \sigma_{\mathbb{U}^\circ} + K\sum_{i=1}^{q} iB \tag{7c}$$

(7c) divided by K is

$$\frac{\sum_{v\in\mathbb{V}^\circ} D_v}{K} \le \frac{\sigma_{\mathbb{U}^\circ}}{K} + \sum_{i=1}^{q} iB \tag{7d}$$

we see $\sigma_{\mathbb{U}^\circ} \ll K$ from (1), thus,

$$\frac{\sum_{v\in\mathbb{V}^\circ} D_v}{K} \le 1 + \sum_{i=1}^{q} iB \tag{7e}$$

Remember

$$D_{v_a^i} := C_{v_a^i} - i \ge (S_{v_a^1} + i) - i = S_{v_a^1}$$

we have:

$$\begin{aligned}\sum_{v\in\mathbb{V}^\circ} D_v = \sum_{a\in\alpha}\sum_{i=1}^{KW_a} D_{v_a^i} &\ge \sum_{a\in\alpha}\sum_{i=1}^{KW_a} S_{v_a^1} = \sum_{a\in\alpha} KW_a S_{v_a^1} \\ &= \sum_{a\in\alpha} KW_a C''_{u_a} = \sum_{a\in\alpha} K\sum_{i=1}^{KW_a} C''_{u_a} \\ &= \sum_{a\in\alpha} K\sum_{i=1}^{KW_a} C''_{u_a^i} = K\sum_{u\in\mathbb{U}^\circ} C''_u\end{aligned} \tag{8}$$

We obtain (6) from (7e) and (8). □

$$Bi + q^6 \ge \sum_{u\in G_i} C'_u \ge Bi - q^5 \tag{9}$$

Proof The deduction of (9) is based on an inequality in the literature. To introduce it, we define L° as the list of all u-jobs and v-jobs in the order of their completion time, and $\#^\circ(j)$ returns the sequence number of job j in L°. Define $\mathbb{J}_i$ as the set of m jobs starting from the ith one in L°, formally, $\mathbb{J}_i := \{j | \#^\circ(j) \in [i, i+m-1]\}$, where $i = 1, \ldots, (K+1)B - m + 1$. We have the following inequality ([Lemma 4]Brucker et al. 2003):

$$\sum_{j\in\mathbb{J}_i} C_j \ge i + m - 1 \tag{10}$$

As for each u-job u, $\#(u) \le \#^\circ(u)$, where "$\ge$" takes "$=$" only when there is no v-job executed before u. The kth job in L° cannot be finished earlier than the kth job in L^{u}. So, for $i \ne 1$, we have:

$$\sum_{u\in G_i} C'_u \ge \sum_{u\in G_i} C_u \ge \sum_{j\in\mathbb{J}_{i'}} C_j \tag{11}$$

where i' is the sequence number in L° of the first job in G_i. Recall definition of G_i, we have $i' = |G_1| + 1 + (i-2)m$, and $|G_1| \ge B - q^5$. Combining (10) and (11):

$$\sum_{u\in G_i} C'_u \ge i' + m - 1 = |G_1| + (i-1)m > iB - q^5 \tag{12}$$

It is worth pointing out that (12) yields for $i = 1$ too, because $C_u \ge 1$ for $u \in G_1$.

Then consider an arbitrary i^*, and sum up (12) for all $i \ne i^*$, we have

$$\sum_{i\ne i^*}\sum_{u\in G_i} C'_u \ge \sum_{i\ne i^*} Bi - q^5(q-1) \tag{13}$$

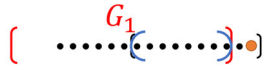
Using $\sum_{u\in\mathbb{U}^\circ} C'_u \le B\sum_{i=1}^{q} i + 1$ from (6), minus (13),

$$\sum_{u\in G_{i^*}} C'_u \le Bi^* + q^5(q-1) + 1 \le Bi^* + q^6 \tag{14}$$

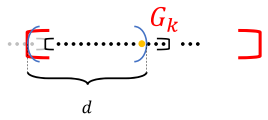
We get (9) from (14) and (12). □

Proof for Proposition 4 Now we give the formal proof. When $i = 1$, S2 is evident. To prove S1, we take a zoom-in view of G_1, see Fig. 12a. The last u-job in G_1 is denoted by u^* and $u^* \in g_{i^*}$. Define $X := g_{i^*} \cap G_1$. Suppose by absurd that $C_{u^*} \ge i + \delta$. Then:

$$\sum_{u\in X} C'_u \ge |X| C_{u^*} \ge |X|(i+\delta)$$



(a) u^*, g_{i^*} and X are represented by orange dot, black brace and blue brace.



(b) G_k^d and u^k are represented by the blue braces and the orange dot.

(c) G_k^A, G_k^B and u^* are represented by blue braces, green braces and orange dot

(d) G_k^{AB} is represented by blue braces

Fig. 12 Zoom-in of big group

and

$$\begin{aligned}\sum_{u\in G_1} C_u^{'} &= \sum_{u\in X} C_u^{'} + \sum_{u\in G_1 - X} C_u^{'} \\ &\geq |X|(1+\delta) + (|G_1| - |X|)\cdot 1 \\ &\geq |G_1| + \delta|X|\end{aligned}$$

By definition of G_1, observe that $g_{i^*} \hat{\subseteq} G_1, |X| \geq d - q^4 \gg \frac{d}{q}$, so

$$\begin{aligned}\sum_{u\in G_1} C_u^{'} &\geq |G_1| + \delta\frac{d}{q} \\ &\geq (B - q^4) + (q^7) \\ &\geq B + q^6\end{aligned}$$

It contradicts (9). We finish the proof for $i = 1$.

Now, we suppose that the proposition holds for $i = 1, \ldots, k-1$, then we prove the $i = k$. We first prove S2 by showing that the dth job of G_k, denoted by u^k, is finished after $k - \Delta$. To make use of (5), we study the set of the first d jobs in G_k, denoted by G_k^d, so that we have $C_{u^k} = C_{G_k^d}^{'}$. See Fig. 12b.

Beforehand, we examine $T(G_k, t_{k-1})$ for preparation, where $t_{k-1} := k - 1 + \delta(k-1)$. As S1 holds for $i = 1, \ldots, k-1$, there are at least $\sum_{i=1}^{k-1} |G_i|$ u-jobs finished before t_{k-1}. We have

$$T(G_k, t_{k-1}) \leq t_{k-1} m - \sum_{i=1}^{k-1} |G_i| \tag{15a}$$

by $m = B + 3q$ and $\sum_{i=1}^{k-1} |G_i| > (k-1)B - q^4$,

$$\begin{aligned}T(G_k, t_{k-1}) &\leq (k - 1 + \delta(k-1))(B + 3q) \\ &\quad - (k-1)B + q^4 \\ &\leq \delta(k-1)B + (k - 1 + \delta(k-1))3q + q^4 \\ &< \delta(k-1)B + q^5\end{aligned} \tag{15b}$$

Now apply (5) on G_k^d:

$$C_{u^k} = C_{G_k^d}^{'} \geq t_{k-1} + 1 - \frac{T(G_k^d, t_{k-1})}{d} \tag{16a}$$

as $T(G_k^d, t_{k-1}) \leq T(G_k, t_{k-1})$ and by (15b),

$$\begin{aligned}C_{u^k} &\geq k + \delta(k-1) - \frac{\delta(k-1)B + q^5}{d} \\ &> k - \frac{\delta(k-1)B + q^5}{d}\end{aligned} \tag{16b}$$

as $\delta(k-1) \leq \frac{q^9}{d}$,

$$\frac{\delta(k-1)B + q^5}{d} \leq \frac{q^9 b + q^5}{d} \leq \Delta \tag{16c}$$

Thus,

$$C_{u^k} > k - \Delta \tag{16d}$$

We proved S2. We turn to proving S1. We calculate the completion time of the last job u^* in G_k, see Fig. 12c. To this purpose, we consider the two subsets of G_k: $G_k^A := g_{(k-1)b+1} \cup g_{(k-1)b+2} \cup \ldots g_{kb-1}$. It is the union of all small groups which are entirely in G_k; $G_k^B := g_{kb} \cap G_k$.

$$\begin{aligned}\sum_{u\in G_k^A} C_u^{'} &= d \sum_{i=(k-1)b+1}^{kb-1} C_{g_i}^{'} \\ &\geq d \sum_{i=(k-1)b+1}^{kb-1} (t_{k-1} + 1 - \frac{T(g^i, t_{k-1})}{d}) \\ &\geq d(k + \delta(k-1))(b-1) \\ &\quad - \sum_{i=(k-1)b+1}^{kb-1} T(g^i, t_{k-1})\end{aligned} \tag{17a}$$

by $\sum_{i=(k-1)b+1}^{kb-1} T(g^i, t_{k-1}) \le T(G_k, t_{k-1})$ and (15b):

$$\begin{aligned}\sum_{u\in G_k^A} C_u^{'} \ge&(k+\delta(k-1))(b-1)d-\delta(k-1)B-q^5\\ \ge&(k+\delta(k-1))(B-d)-\delta(k-1)B-q^5\\ =&kB+\delta(k-1)B-kd-\delta(k-1)d\\ &-\delta(k-1)B-q^5\\ =&kB-kd-\delta(k-1)d-q^5\end{aligned} \tag{17b}$$

Calculate (9)–(17b):

$$\begin{aligned}\sum_{u\in G_k-G_k^A} C_u^{'} \le&kB+q^6-[kB-kd-\delta(k-1)d-q^5]\\ =&kB+q^6-kB+kd+\delta(k-1)d+q^5\\ =&q^6+kd+\delta(k-1)d+q^5\\ =&q^6+(k+\delta(k-1))d+q^5\\ =&q^6+(k+\delta(k-1))(d-q^4)\\ &+(k+\delta(k-1))q^4+q^5\\ <&q^6+(k+\delta(k-1))(d-q^4)+2q^5\\ <&(k+\delta(k-1))(d-q^4)+2q^6\end{aligned} \tag{17c}$$

as $G_k^B \subseteq G_k - G_k^A$,

$$\sum_{u\in G_k^B} C_u^{'} \le \sum_{u\in G_k-G_k^A} C_u^{'} \le (k+\delta(k-1))(d-q^4)+2q^6 \tag{17d}$$

by $g_{kb}\hat{\subseteq}G_k$ and $|G_k^B| = |g_{kb}\cap G_k| \ge d-q^4$,

$$\sum_{u\in G_k^B} C_u^{'} \ge |G_k^B| C_{u^*} \ge (d-q^4)C_{u^*} \tag{17e}$$

Combining (17d) and (17e),

$$(d-q^4)C_{u^*} \le \sum_{u\in G_k^B} C_u^{'} \le (k+\delta(k-1))(d-q^4)+2q^6 \tag{17f}$$

As $d-q^4 \ge \frac{d}{q}$ and $\delta := \frac{q^8}{d}$, we get

$$\begin{aligned}C_{u^*} \le& \frac{(k+\delta(k-1))(d-q^4)+2q^6}{d-q^4}\\ \le& k+\delta(k-1)+\frac{2q^7}{d}\\ \le& k+k\delta\end{aligned} \tag{17g}$$

We have proved S1 and completed the proof for Proposition 4. □

Proof for Proposition 5 By S2 in Proposition 4, the dth job is finished after $i^{'}-\Delta$. As $g_i\hat{\subseteq}G_{i'}$,

$$C_u^{''} \ge C_u^{'} \ge C_{g_i}^{'} \ge i^{'}-\Delta \tag{18}$$

Now we prove the right side. Suppose by absurd that $\exists g_{i^*}\hat{\subseteq}G_{i'}$, $\exists u \in g_{i^*}$ with $C_u^{''} > i^{'}-\Delta+\frac{1}{3}$. We show that this causes an increase of $\sum_{u\in\mathbb{U}^\circ} C_u^{''}$ by $\frac{d}{3}$, which leads to an contradiction to (6).

For any big group G_k, define its subset $G_k^{AB} := \{u \in G_k | u \in g, g\hat{\subseteq}G_k\}$, as illustrated in Fig. 12d.

$$\sum_{u\in G_k} C_u^{''} \ge \sum_{u\in G_k^{AB}} C_u^{''} \tag{19a}$$

By (18),

$$\sum_{u\in G_k^{AB}} C_u^{''} \ge |G_k^{AB}|(k-\Delta) \tag{19b}$$

As $|G_k^{AB}| = m - \sum_{k=1}^{q-k+1} 3qk > B-q^4$,

$$|G_k^{AB}|(k-\Delta) \ge (B-q^4)(k-\Delta) \ge B(k-\Delta)-q^5 \tag{19c}$$

which is

$$\sum_{u\in G_k} C_u^{''} \ge B(k-\Delta)-q^5 \tag{19d}$$

Specially,

$$\begin{aligned}\sum_{u\in G_{i'}} C_u^{''} \ge& \sum_{u\in G_{i'}^{AB}-g_{i^*}^s} C_u^{''} + \sum_{u\in g_{i^*}^s} C_u^{''}\\ \ge& (i^{'}-\Delta)(|G_{i'}^{AB}|-|g_{i^*}^s|)+\left(i^{'}-\Delta+\frac{1}{3}\right)|g_{i^*}^s|\\ =&(i^{'}-\Delta)|G_{i'}^{AB}|+\frac{1}{3}|g_{i^*}^s|\\ \ge&(i^{'}-\Delta)(B-q^4)+\frac{1}{3}(d-q^4)\\ \ge&(i^{'}-\Delta)B+\frac{1}{3}d-q^5\end{aligned} \tag{20}$$

where $g_{i^*}^s := g_{i^*}\cap G_{i'}$.

Then calculate the total by adding (20) to (19d):

$$\begin{aligned}\sum_{u\in\mathbb{U}^{\circ}} C_u^{''} &\geq \sum_{k\neq i'} B(k-\Delta)+(i^{'}-\Delta)B+\frac{1}{3}d-2q^5\\ &=\sum_{k=1}^{q} B(k-\Delta)+\frac{1}{3}d-2q^5\\ &\geq \sum_{k=1}^{q} Bk+\left(\frac{1}{3}d-qB\Delta-q^6\right)\\ &> \sum_{k=1}^{q} Bk+1\end{aligned}$$

A contradiction to (6). □

6 Final remarks

We developed an algorithm for the scheduling problem $Pm|p_j = 1,\text{i.t.}|\sum C_j$, with a complexity $O(h^{m+1}n)$. This leads to the conclusion that the complexity is principally determined by the height of the graph. Using a similar idea, we show how problem $Pm|p_j = 1,\text{l.o.}|\sum C_j$ can be solved in polynomial time. Moreover, we proved the strong NP-hardness of $P|p_j = 1,\text{i.t.,pmtn}|\sum C_j$. Both are new complexity results. For future research, there are still interesting open problems as shown in Table 1.

Appendix

The candidate sets of the example in Fig. 6. {1}, {2}, {3}, {4}, {5}, {8}, {9}, {10}, {11}, {12}, {13}, {14}, {15}, {1, 2}, {1, 3}, {1, 8}, {1, 9}, {1, 10}, {1, 12}, {1, 13}, {2, 3}, {2, 12}, {2, 13}, {3, 12}, {3, 13}, {4, 5}, {4, 11}, {5, 11}, {8, 11}, {8, 12}, {8, 9}, {8, 12}, {8, 11}, {8, 14}, {9, 10}, {9, 11}, {9, 12}, {9, 14}, {10, 11}, {10,12}, {10, 13}, {10, 14}, {11, 12}, {11, 13}, {12, 13}, {13, 14}.

References

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., Hurink, J. L., & Knust, S. (2001). A polynomial algorithm for $P|p_j = 1, r_j, outtree|\sum C_j$. *Mathematical Methods of Operations Research, 56,* 407–412.

Brucker, P., Heitmann, S., & Hurink, J. (2003). How useful are preemptive schedules? *Operations Research Letters, 31*(2), 129–136.

Chen, B., Coffman, E., Dereniowski, D., & Kubiak, W. (2016). Normal-form preemption sequences for an open problem in scheduling theory. *Journal of Scheduling, 19*(6), 701–728.

Coffman, E. G., Dereniowski, D., & Kubiak, W. (2012). An efficient algorithm for finding ideal schedules. *Acta Informatica, 49*(1), 1–14.

Coffman, E. G., Sethuraman, J., & Timkovsky, V. G. (2003). Ideal preemptive schedules on two processors. *Acta Informatica, 39*(8), 597–612.

Cygan, M., Fomin, F. V., Kowalik, Ł, Lokshtanov, D., Marx, D., Pilipczuk, M., et al. (2015). *Parameterized algorithms* (Vol. 5). Springer.

Dolev, D., & Warmuth, M. K. (1984). Scheduling precedence graphs of bounded height. *Journal of Algorithms, 5*(1), 48–59.

Dolev, D., & Warmuth, M. K. (1985). Profile scheduling of opposing forests and level orders. *SIAM Journal on Algebraic Discrete Methods, 6*(4), 665–687.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5,* 287–326.

Hu, T. C. (1961). Parallel sequencing and assembly line problems. *Operations Research, 9*(6), 841–848.

Huo, Y., & Leung, J.Y.-T. (2006). Minimizing mean flow time for UET tasks. *ACM Transactions on Algorithms (TALG), 2*(2), 244–262.

Kubiak, W., Rebaine, D., & Potts, C. (2009). Optimality of HLF for scheduling divide-and-conquer UET task graphs on identical parallel processors. *Discrete Optimization, 6*(1), 79–91.

Mnich, M., & van Bevern, R. (2018). Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research, 100,* 254–261.

Muntz, R. R., & Coffman, E. G., Jr. (1970). Preemptive scheduling of real-time tasks on multiprocessor systems. *Journal of the ACM, 17*(2), 324–338.

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

Prot, D., & Bellenguez-Morineau, O. (2018). A survey on how the structure of precedence constraints may change the complexity class of scheduling problems. *Journal of Scheduling, 21*(1), 3–16.

Soper, A. J., & Strusevich, V. A. (2019). Schedules with a single preemption on uniform parallel machines. *Discrete Applied Mathematics, 261,* 332–343.

van Bevern, R., Bredereck, R., Bulteau, L., Komusiewicz, C., Talmon, N., & Woeginger, G. J. (2016). Precedence-constrained scheduling problems parameterized by partial order width. In *International conference on discrete optimization and operations research* (pp. 105–120). Springer.

Wang, T., & Bellenguez-Morineau, O. (2019). A complexity analysis of parallel scheduling unit-time jobs with in-tree precedence constraints while minimizing the mean flow time. *Journal of Scheduling, 22,* 1–6.

Wang, T., & Bellenguez-Morineau, O. (2019). The complexity of parallel machine scheduling of unit-processing-time jobs under level-order precedence constraints. *Journal of Scheduling, 22*(3), 263–269.