Faster Exponential Algorithms for Multi-Machine Scheduling Problems

Anubhav Dhar
Max Planck Institute for Informatics, Saarbrücken, Germany
Saarland University, Saarbrücken, Germany

Anita Dürr
ETH, Zurich, Switzerland

Ahmed Ghazy
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
Saarland University, Saarbrücken, Germany

Jakob Greilhuber
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
Saarland University, Saarbrücken, Germany

Karol Węgrzycki
Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

Minimizing the weighted completion times ($P \mid\mid \Sigma w_j C_j$) and weighted number of tardy jobs ($P \mid\mid \Sigma w_j U_j$) on multiple identical machines are two classical NP-hard scheduling problems. As shown by Lenté et al. (2014), both problems can be solved in time $\mathcal{O}^\star(3^n)$. In this paper, we improve these bounds to $\mathcal{O}(2.755^n)$ and $\mathcal{O}^\star(2^n)$, respectively. Our algorithm for $P \mid\mid \Sigma w_j C_j$ exploits the meet-in-the-middle paradigm and an efficient data structure answering linear programming queries. Additionally, when the number of machines is at most 6, we show that the running time for $P \mid\mid \Sigma w_j C_j$ can further be improved.

Both scheduling problems are generalizations of the classical Bin Packing problem, which can be solved in $\mathcal{O}^\star(2^n)$ time. Improving this running time is an important open question. We show that, when assuming the Asymptotic Rank Conjecture (ARC), Bin Packing can be solved in time $\mathcal{O}((2-\varepsilon)^n)$ for some $\varepsilon > 0$. Our algorithm makes use of two main ingredients: the recent $\mathcal{O}((2-\varepsilon)^n)$-time algorithm of Nederlof et al. [SICOMP'23] for Bin Packing when the number of bins is a fixed constant, and the $\mathcal{O}((2-\varepsilon)^n)$-time algorithm of Björklund et al. [SODA'25] for special instances of the 3-way Partitioning problem when assuming ARC.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Scheduling, exact algorithms, exponential-time algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.53

Related Version *Full Version*: http://arxiv.org/abs/2608.12224

Funding *Anubhav Dhar*: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant number 559177164.
Anita Dürr: Part of this work was done while affiliated to Saarland University and Max Planck Institute for Informatics, Saarbrücken, Germany, where this work was part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 850979).
Karol Węgrzycki: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant number 559177164.

1 Introduction

A central goal in the study of NP-hard problems is to determine the smallest possible running time, specifically for problems that admit exponential-time algorithms. For many fundamental scheduling problems a $\mathcal{O}^\star(3^n)$-time algorithm constitutes a natural baseline, where n is the number of jobs [30].[1] Contrasting this, the Exponential Time Hypothesis (ETH) excludes $2^{o(n)}$-time algorithms [23]. Therefore, it is a key question to find $\mathcal{O}^\star(C^n)$-time algorithms, for a constant $C > 0$ as small as possible. In this paper, we address this question for two fundamental scheduling problems.

Typically, the input of scheduling problems consists of n jobs with processing times $p_1, \ldots, p_n$, weights $w_1, \ldots, w_n$ and due dates $d_1, \ldots, d_n$, as well as a number $m \in \mathbb{N}$ of machines available. A schedule is an assignment of each job to a machine and a starting time, such that no two jobs are processed at the same time by the same machine. The goal is to find a schedule that minimizes some cost, where the used cost function varies from one scheduling problem to another. In this paper, we focus on two cost functions: the total weighted completion time $\sum_{j=1}^{n} w_j C_j$ and the weighted number of tardy jobs $\sum_{j=1}^{n} w_j U_j$, where C_j denotes the completion time of job j and U_j is the indicator variable taking value 1 if job j completes after its due date d_j (i.e. $C_j > d_j$, in which case job j is called *tardy*) and 0 otherwise. In the Graham et al. [18] notation[2] these problems are denoted by $P \mid\mid \Sigma w_j C_j$ and $P \mid\mid \Sigma w_j U_j$, respectively.

The problems $P \mid\mid \Sigma w_j C_j$ and $P \mid\mid \Sigma w_j U_j$ are among the most fundamental problems in theoretical computer science. Minimizing the weighted number of tardy jobs is NP-hard even on a single machine [24, 29]. The problem $P \mid\mid \Sigma w_j C_j$ is polynomial-time solvable on a single machine [39], but NP-hard already when $m = 2$ [10, 29]. As a result, numerous approximation algorithms [17, 38] as well as exact algorithms with parameterized or pseudopolynomial running time for (special cases of) both $P \mid\mid \Sigma w_j U_j$ [1, 8, 9, 16, 20, 22, 25, 27, 28, 32] and $P \mid\mid \Sigma w_j C_j$ [21, 26–28, 37] are present in the literature.

Lenté et al. [30] were the first to provide non-trivial exact exponential-time algorithms for $P \mid\mid \Sigma w_j C_j$ and $P \mid\mid \Sigma w_j U_j$, concretely they proposed $\mathcal{O}^\star(3^n)$-time algorithms. Contrasting this result, the work of Jansen et al. [23] implies that those problems do not admit any $2^{o(n)}$-time algorithms, unless ETH fails. In this work, we improve upon the results of Lenté et al. [30].

▶ **Theorem 1.** *$P \mid\mid \Sigma w_j C_j$ can be solved in time $\mathcal{O}(2.755^n)$.*

▶ **Theorem 2.** *$P \mid\mid \Sigma w_j U_j$ can be solved in time $\mathcal{O}^\star(2^n)$.*

Both Theorems 1 and 2 are proven using standard dynamic programming. Interestingly, for $P \mid\mid \Sigma w_j C_j$ the base cases (computing the dynamic programming tables for a constant number of machines) can be handled efficiently by exploiting the meet-in-the-middle paradigm combined with an efficient data structure that supports linear programming queries in m dimensions. For $P \mid\mid \Sigma w_j U_j$ we observe that Fast Subset Convolution [3] can be used to solve the problem in the stated time, our concrete approach uses an algorithm of Moore [32] to handle the base case.

Additionally, we consider $P \mid\mid \Sigma w_j C_j$ for a fixed number of machines m, and show that even faster algorithms are possible when $m \leqslant 6$. For each integer $m \geqslant 1$, we denote the problem $P \mid\mid \Sigma w_j C_j$ where the number of machines is the fixed constant m as $Pm \mid\mid \Sigma w_j C_j$.

[1] A function $f(n)$ is in $\mathcal{O}^\star(g(n))$ if it is in $\mathcal{O}(g(n) \cdot n^c)$ for some constant c.
[2] This useful notation is discussed in more detail later in the introduction.

▶ **Theorem 3.** *$Pm \mid\mid \Sigma w_j C_j$ can be solved in time:*

- $\mathcal{O}^\star(m^{n/2})$ *if* $m \leqslant 3$,
- $\mathcal{O}(2.389^n)$ *if* $m = 4$,
- $\mathcal{O}(2.726^n)$ *if* $m = 5$,
- $\mathcal{O}(2.733^n)$ *if* $m = 6$.

Bin Packing

The scheduling problems we considered above are a generalization of the scheduling problem where the cost function is given by the maximum completion time $C_{\max} \coloneqq \max_j C_j$, called the *makespan* (see [8]). This scheduling problem is denoted as $P \mid\mid C_{\max}$ in the Graham notation. Its decision version coincides with the classical NP-hard Bin Packing problem.

Bin Packing can be solved in time $\mathcal{O}^\star(2^n)$ [4], and faster algorithms are known when $m = \mathcal{O}(1)$ or $m = \Theta(n)$ [33,35]. However, in the general case, it remains a major open problem to improve the running time of $\mathcal{O}^\star(2^n)$ [34]. As a step towards answering that question, we can ask whether a faster algorithm is possible when assuming Strassen's *Asymptotic Rank Conjecture* (ARC), a longstanding conjecture that provides significant algorithmic power. Indeed, this has been a successful approach in the past. Faster algorithms for several problems were designed under the ARC, e.g. for k-Set Cover [5], Chromatic Number [2], Hamiltonian Cycle on bounded treewidth graphs [6], Generalized Convolution [7], and k-Orthogonal Vectors [15]. In this work, we follow the same strategy and show the existence of a faster than 2^n-time algorithm for Bin Packing when assuming ARC.

▶ **Theorem 4.** *Assuming the Asymptotic Rank Conjecture, there exists a constant $\varepsilon > 0$ such that the* Bin Packing *problem on n items can be solved with high probability in time $\mathcal{O}((2-\varepsilon)^n)$.*

As it seems hard to obtain such a fast algorithm unconditionally, one might ask whether there exists a lower bound based on the *Strong Exponential Time Hypothesis* (SETH) that rules out algorithms running in time $\mathcal{O}^\star((2-\varepsilon)^n)$ for any $\varepsilon > 0$. However, such a lower bound would show that at most one of the SETH or the ARC is true, which would represent a major breakthrough in the understanding of these hypotheses. Therefore, we view our result as a barrier to obtaining a tight SETH lower bound. Contrasting this, since it is already known that at most one of the ARC and the *Set Cover Conjecture (SCC)* are true [5], it might be feasible to obtain a tight lower bound under SCC. We leave a lower bound under SCC as an interesting direction for future work.

Further scheduling problems

In this paper we mainly focus on scheduling problems where the cost function is given by $\sum_{j=1}^n w_j C_j$ and $\sum_{j=1}^n w_j U_j$. Since there are many other possibilities, one might wonder about the state-of-the-art for different scheduling problems, for example for unweighted variants or entirely different cost functions. This leads to a systematic study of scheduling problems.

In order to discuss this further, we first provide additional background on problems on parallel machines. The three-field notation $\alpha \mid \beta \mid \gamma$ introduced by Graham et al. [18] is a shorthand notation for scheduling problems. The first field α describes the machine environment, which in this paper will always be "P" or "Pm", denoting identical parallel machines where the number of machines is respectively part of the input or the fixed constant m. The second field β specifies additional constraints, and will always be empty in this paper.

The third field γ denotes the objective function that should be minimized. Natural objective functions are $\sum_{j=1}^{n} f_j$, where f_j is a cost function associated to the job j which may take either of the following values:

- the completion time C_j of job j;
- the indicator variable U_j which takes value 1 if job j completes after its due date (i.e. $C_j > d_j$) and 0 otherwise;
- the lateness $L_j \coloneqq C_j - d_j$ of job j; or
- the tardiness $T_j \coloneqq \max\{C_j - d_j, 0\}$ of job j;

as well as their weighted variants, i.e. where the above measures are multiplied by w_j. By restricting the study to identical parallel machines without additional constraints (i.e. the first field α is always "P", and the second field β is always empty), this defines 8 distinct scheduling problems. Note, however, that the objectives $\sum_{j=1}^{n} w_j L_j$ and $\sum_{j=1}^{n} w_j C_j$ (as well as their unweighted variants) simply differ by the quantity $\sum_j d_j w_j$, which is independent of any underlying schedule. Additionally, the problem of minimizing the total completion time $P \mid\mid \Sigma C_j$ can be solved in polynomial time [10]. This leaves 5 scheduling problems worth studying in the context of exponential-time algorithms. All of these remaining 5 problems were shown to be NP-hard early on: $P \mid\mid \Sigma w_j C_j$ and $P \mid\mid \Sigma U_j$ are NP-hard already for $m = 2$, and $P \mid\mid \Sigma w_j U_j$ is NP-hard already for $m = 1$ [10, 24, 29].[3] Meanwhile, Du and Leung [14] showed that $P \mid\mid \Sigma T_j$ (and thus $P \mid\mid \Sigma w_j T_j$) is NP-hard when $m = 1$. On the other hand, Lenté et al. [30] proposed an algorithm solving any of the above scheduling problems in time $\mathcal{O}^\star(3^n)$.[4]

Theorems 1 and 2 show that this upper bound can be improved for $P \mid\mid \Sigma w_j C_j$ and $P \mid\mid \Sigma w_j U_j$. We leave whether the upper bound can also be improved for $P \mid\mid \Sigma w_j T_j$ and $P \mid\mid \Sigma T_j$ as an interesting question for future work.

Organization of the paper

We give a technical overview of our results in Section 2. After defining the used notation and stating technical tools in Section 3, we prove Theorem 1 in Section 4 and prove Theorem 2 in Section 5. We defer the proof of Theorem 3 to Appendix A. Finally, Appendix B establishes a technical lemma. The proof of Theorem 4 can be found in the full version.

2 Technical Overview

We now define the studied problems and give a high-level overview of Theorems 1–4.

2.1 Weighted completion time

$P \mid\mid \Sigma w_j C_j$ denotes the scheduling problem where the objective is to minimize the total weighted completion time.

$P \mid\mid \Sigma w_j C_j$

Input:	n jobs, where the j-th job consists of processing time $p_j \in \mathbb{N}$ and weight $w_j \in \mathbb{N}$; number of machines $m \in \mathbb{N}$ with $m \geqslant 1$.
Task:	Compute the minimum total weighted completion time $\sum_{j=1}^{n} w_j C_j$ of scheduling the given n jobs on m identical parallel machines.

[3] For $m = 1$, the problems $P \mid\mid \Sigma w_j C_j$ and $P \mid\mid \Sigma U_j$ become polynomial-time solvable [31, 32, 39].
[4] In fact, the work of Lenté et al. [30] is applicable for an even wider range of cost functions f_j.

Whenever the number of machines m is a fixed constant, we denote the problem as $Pm \mid\mid \Sigma w_j C_j$, e.g. $P3 \mid\mid \Sigma w_j C_j$ denotes the special case of $P \mid\mid \Sigma w_j C_j$ with $m = 3$.

We solve $P \mid\mid \Sigma w_j C_j$ using a standard dynamic programming approach as follows. Each dynamic programming entry tracks the minimum total weighted completion time for a given subset of jobs and a given number of machines. Instead of considering the naive transition which iterates through all possible subsets of jobs over a single machine, we restrict our focus to the set of all possible jobs on the machine with the least number of jobs. This ensures that the transitions are computable in time faster than $\mathcal{O}^\star(3^n)$. Another crucial observation is that the base cases corresponding to smaller instances with $m \leqslant 3$ machines can be computed in time $\mathcal{O}^\star(m^{n/2})$ (as stated in Theorem 3). In the end, we show that the worst-case running time is $\mathcal{O}(2.755^n)$. We refer to Section 4 for more details of the proof of Theorem 1. For now, we discuss the algorithm for $Pm \mid\mid \Sigma w_j C_j$ for $m \leqslant 3$. Similar ideas can be used to improve the running time for $m \in \{4, 5, 6\}$ machines as well.

Number of machines $m \in \{2, 3\}$. To solve $Pm \mid\mid \Sigma w_j C_j$ in time $\mathcal{O}^\star(m^{n/2})$ for $m \in \{2, 3\}$, we use a meet-in-the-middle approach, along with the following data structure due to Guibas et al. [19], which is capable of supporting efficient linear programming queries.[5]

▶ **Lemma 5** (Linear Programming Queries [19]). *Let $d \in \{2, 3\}$ and $\mathcal{P} \subseteq \mathbb{Z}^d$ be a set of size n. A data structure can be constructed in time $\mathcal{O}(n \log n)$ that answers each following query in time $\mathcal{O}(\log n)$: Given $(q_1, \ldots, q_d) \in \mathbb{Z}^d$, compute $\min_{(r_1,\ldots,r_d)\in\mathcal{P}} \sum_{i\in[d]} q_i \cdot r_i$.*

We split the jobs of the instance into two halves, $A = \{1, \ldots, \lfloor n/2 \rfloor\}$, and $B = \{\lfloor n/2 \rfloor + 1, \ldots, n\}$. Using Lemma 5, we create a data structure $\mathcal{D}$ on $d = m$ dimensions, which contains useful information for every partition $Y_1 \uplus \cdots \uplus Y_m = B$. Preprocessing of this data structure takes time $\mathcal{O}^\star(m^{n/2})$. Next, for every partition $X_1 \uplus \cdots \uplus X_m = A$, we query $\mathcal{D}$ to obtain the minimum weighted completion time when $X_i \cup Y_i$ are the jobs scheduled on the i-th machine, over all partitions $Y_1 \uplus \cdots \uplus Y_m = B$. Together, all queries take time $\mathcal{O}^\star(m^{n/2})$. Finally, we combine all queries to obtain the minimum weighted completion time over all schedules. In total, this approach takes time $\mathcal{O}^\star(m^{n/2})$. Refer to Lemma 11 in Section 4 for more details.

Number of machines $m = 4$. From the previous discussion about improving the transitions in the dynamic program, we can use the algorithm for $m \in \{2, 3\}$ to obtain an algorithm solving $P4 \mid\mid \Sigma w_j C_j$ in time $\mathcal{O}(2.755^n)$ (see Theorem 1). We improve this running time further. Although the meet-in-the-middle approach we used for $m \in \{2, 3\}$ still works for $m \geqslant 4$ using known data structures analogous to that of Lemma 5, albeit with worse running times, (see, e.g., [11, Corollary 3.2]), we obtain significantly better running times with a different approach. A clever trick helps us to reduce 4-dimensional queries to 3-dimensional queries, letting us exploit Lemma 5. While this results in an extra preprocessing overhead, it nevertheless allows us to solve $P4 \mid\mid \Sigma w_j C_j$ even faster. Interestingly, this extra preprocessing overhead causes the algorithm to achieve a better running time when we split the set of jobs into unequal parts, as opposed to the traditional meet-in-the-middle approach which splits into equal parts.

To capture the split into unequal parts consider a fraction $\alpha \in (0, 1)$ which will be set later, let A be the first $\lfloor \alpha n \rfloor$ of the jobs, and B be the remaining. We guess a partition $X_1 \uplus X_2 \uplus X_3 \uplus X_4 = A$. Without loss of generality assume $|X_1| \leqslant |X_2| \leqslant |X_3| \leqslant |X_4|$. We

[5] The data structure of Guibas et al. [19] preprocesses a convex polyhedron (the intersection of n given half-spaces). We obtain the required polyhedron by computing the convex hull of the n input points in time $O(n \log n)$ [12, 36].

create a separate linear programming query data structure (as in Lemma 5) for *every* possible X_1, which contains relevant information for every partition $Y_1 \uplus Y_2 \uplus Y_3 \uplus Y_4 = B$. Note that this steps differs from the algorithm for $m \in \{2, 3\}$ which used a single data structure. Moreover, since we create a separate data structure for every X_1, it suffices to set these up on $d = m - 1 = 3$ dimensions. We show that preprocessing all data structures takes time $\mathcal{O}^\star(1.755^{|A|} \cdot 4^{|B|})$. Next, for all partitions $X_1 \uplus X_2 \uplus X_3 \uplus X_4 = A$, we query the data structure specific to X_1 for the minimum weighted completion time when $X_i \cup Y_i$ are the jobs scheduled on the i-th machine, over all partitions $Y_1 \uplus Y_2 \uplus Y_3 \uplus Y_4 = B$. Together, all queries take $\mathcal{O}^\star(4^{|A|})$ time. Finally, we combine all queries to obtain the minimum weighted completion time over all schedules. In total this takes time $\mathcal{O}^\star(1.755^{|A|} \cdot 4^{|B|} + 4^{|A|})$ which for $\alpha = 0.628$ turns out to be $\mathcal{O}(2.389^n)$. Refer to Lemma 15 in Appendix A for more details.

Number of machines $m \in \{5, 6\}$. To solve $Pm \mid\mid \Sigma w_j C_j$ with $m \in \{5, 6\}$ faster than $\mathcal{O}(2.755^n)$, we revisit the dynamic programming approach used to prove Theorem 1. Each entry, $\texttt{DP}[i, S]$, tracks the minimum total weighted completion time for a given subset $S \subseteq \{1, \dots n\}$ of jobs when scheduled on i machines. Note that the value of $\texttt{DP}[m, \{1, \dots, n\}]$ contains the minimum weighted completion time for the original input instance. We do the following.

- For $m = 5$, we compute $\texttt{DP}[2, S]$ for all subsets $S \subseteq \{1, \dots, n\}$ and compute $\texttt{DP}[3, S]$ only for subsets $S \subseteq \{1, \dots, n\}$ with $|S| \leqslant 3n/5$ using the above algorithms for $P2 \mid\mid \Sigma w_j C_j$ and $P3 \mid\mid \Sigma w_j C_j$. We show that these computed entries are sufficient to directly compute the value of $\texttt{DP}[5, \{1, \dots, n\}]$. This algorithm takes time $\mathcal{O}(2.726^n)$, with the bottleneck being the computation of $\texttt{DP}[3, S]$ for all subsets $S \subseteq \{1, \dots, n\}$ with $|S| \leqslant 3n/5$. Refer to Lemma 16 in Appendix A for more details.
- For $m = 6$, we compute $\texttt{DP}[3, S]$ for all subsets $S \subseteq \{1, \dots, n\}$ by solving $P3 \mid\mid \Sigma w_j C_j$ on S. We show that these computed entries are sufficient to directly compute the value of $\texttt{DP}[6, \{1, \dots, n\}]$. This algorithm takes time $\mathcal{O}(2.733^n)$, with the bottleneck being the computation of $\texttt{DP}[3, S]$ for all subsets $S \subseteq \{1, \dots, n\}$. Refer to Lemma 17 in Appendix A for more details.

2.2 Weighted number of tardy jobs

$P \mid\mid \Sigma w_j U_j$ denotes the scheduling problem where the objective is to minimize the total weighted number of tardy jobs.

$P \mid\mid \Sigma w_j U_j$

Input:	n jobs where the j-th job consists of processing time $p_j \in \mathbb{N}$ and weight $w_j \in \mathbb{N}$ and due date $d_j \in \mathbb{N}$; number of machines $m \in \mathbb{N}$ with $m \geqslant 1$.
Task:	Compute the minimum total weight of tardy jobs $\sum_{j=1}^{n} w_j U_j$ of scheduling the given n jobs on m identical parallel machines.

To solve $P \mid\mid \Sigma w_j U_j$ in time $\mathcal{O}^\star(2^n)$, we iterate over all subsets $S \subseteq [n]$, check whether S can be scheduled on m machines with every job meeting its deadline, and output the minimum possible value of $w([n] \setminus S)$. To check whether a subset S can be scheduled on m machines with every job meeting its deadline, we use dynamic programming to maintain whether a subset $S \subseteq [n]$ on $i \leqslant m$ machines can be scheduled with every job meeting its deadline. The base case of the dynamic program (i.e. for $i = 1$) can be computed for each subset $S \subseteq [n]$ using a polynomial-time algorithm due to Moore [32], and the transitions together for a fixed $i \geqslant 2$ can be computed using a Fast Subset Convolution algorithm due to Björklund et al. [3]. For more details refer to the proof of Theorem 2 in Section 5.

2.3 Bin Packing

Let us now give a high level overview of our result for the Bin Packing problem. We solve the decision version of Bin Packing as constructing a solution can be done by a standard reduction to polynomially many calls to a decision oracle.

Bin Packing
Input: Multiset $I \subseteq \mathbb{N}$ of n items; number of bins $m \in \mathbb{N}$ with $m \geqslant 1$; capacity $t \in \mathbb{N}$.
Task: Decide if there is a partition $X_1 \uplus \cdots \uplus X_m$ of I into m parts such that $\sum(X_i) \leqslant t$ for all $i \in [m]$.

To solve Bin Packing in faster than 2^n-time, we will make use of the following two important results. First, we recall that Nederlof et al. [35] showed that if the number of bins m is a fixed constant (e.g. $m = 6$), then Bin Packing can be solved in time faster than $\mathcal{O}^\star(2^n)$.

▶ **Lemma 6** ([35, Theorem 1.1]). *For every $m \in \mathbb{N}$ there is a constant $\varepsilon_m > 0$ such that* Bin Packing *on instances with exactly m bins can be solved with high probability in time $\mathcal{O}^\star(2^{(1-\varepsilon_m)n})$.*

Next, since we assume the Asymptotic Rank Conjecture, we can leverage the fast algorithm of Björklund et al. [2] that solves the 3-way Partitioning problem as defined below. For a universe U and a family $\mathcal{F}$ of subsets of U, we say that $\mathcal{F}$ is ν-bounded if no set in $\mathcal{F}$ is larger than $\nu|U|$.

3-way Partitioning for ν-bounded set families
Input: A universe $U \subseteq \mathbb{N}$; ν-bounded set families $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3 \subseteq 2^U$.
Task: Decide whether there exist pairwise disjoint subsets $S_1 \in \mathcal{F}_1$, $S_2 \in \mathcal{F}_2$ and $S_3 \in \mathcal{F}_3$ such that $S_1 \uplus S_2 \uplus S_3 = U$.

▶ **Lemma 7** ([2, Theorem 1]). *Assuming the Asymptotic Rank Conjecture, for every $\varepsilon > 0$, there exists $\eta_\varepsilon > 0$ such that the* 3-way Partitioning *problem over an n-element universe for $(1-\varepsilon)/2$-bounded set families can be solved deterministically in time $\mathcal{O}^\star(2^{(1-\eta_\varepsilon)n})$.*

The general idea to prove Theorem 4 is to reduce solving the given Bin Packing instance either to solving a Bin Packing instance with a constant number of bins, which can be solved efficiently using Lemma 6, or to solving a 3-way Partitioning instance over an n-element universe for $(1-\varepsilon)/2$-bounded set families, which can be solved efficiently using Lemma 7 when assuming ARC. We now give a high-level overview of this strategy.

Consider a Bin Packing instance (m, t, I). Any partition of I into m parts $X_1, \ldots, X_m$ is called a *packing* of I into m bins. The quantity $\Sigma(X_i)$ is called the *load of the i-th bin*. The packing forms a *solution* if $\Sigma(X_i) \leqslant t$ for all $i \in [m]$, i.e. the load of each bin does not exceed the capacity t. The goal of the Bin Packing problem is to decide whether the given instance admits a solution. To do that, we will distinguish between four types of solutions. For each solution type, we will show a faster than 2^n-time algorithm that, given a Bin Packing instance, outputs No if the instance has no solution and outputs Yes if the instance admits a solution of that solution type. In the end, given any Bin Packing instance, we can simply run the four algorithms for each solution type and output No if and only if all the algorithms output No.

We now define the different solution types we consider. For that, consider a solution $X_1, \ldots, X_m$ to that given Bin Packing instance (m, t, I). Let $b_i := |X_i|$ for all $i \in [m]$. Without loss of generality, we can assume that $b_1 \geqslant \ldots \geqslant b_m$. Let $\varepsilon > 0$ be a small enough constant. Then, we distinguish the following cases.

Case A. If $b_1 \geqslant (1+\varepsilon)\cdot n/2$, i.e. *more than half of the items are packed into the first bin*, then we can guess $\leqslant (1-\varepsilon)n/2$ items and use standard dynamic programming to verify that they can be packed into $m-1$ bins.

Case B. If $\sum_{i=1}^{6} b_i \geqslant (1-\varepsilon)\cdot n$, i.e. *almost all items are packed into the first 6 bins*, then we can guess $\leqslant \varepsilon n$ items and use standard dynamic programming to verify that they can be packed into $m-6$ bins, and then use Lemma 6 to verify that the $\geqslant (1-\varepsilon)n$ items can be packed into 6 bins.

Case C. If $b_1 \in [\frac{n}{2} \pm \varepsilon\cdot\frac{n}{2}]$ and $\sum_{i=1}^{6} b_i < (1-\varepsilon)\cdot n$, i.e. *the first bin contains approximately half of the items, but the remaining items are not concentrated on the first 6 bins*, then we use a randomized strategy that samples witnesses of the existence of such a packing. This algorithm is inspired by the proof of [35, Lemma 3.5].

Case D. If $b_1 \leqslant (1-\varepsilon)\cdot n/2$ and $\sum_{i=1}^{6} b_i < (1-\varepsilon)\cdot n$, i.e. *the first bin contains less than half of the items, and the items are not concentrated on the first 6 bins*, we reduce solving the problem to solving specific instances of 3-way Partitioning, which we then solve using Lemma 7. This is the only case in which we make use of the ARC.

Of course, when given a yes-instance of Bin Packing, we do not know in which case we are. Hence, we simply apply the algorithms for all the four cases in sequence, one of which will detect that the instance is a yes-instance (with high probability). For details, see the full version of this paper.

3 Preliminaries

In this paper, we work in a unit-cost RAM model with a word size of $\Theta(L)$ bits, where L is the encoding length of the input. Thus, arithmetic operations on integers with $O(L)$ bits take constant time. The unit-cost assumption can be dropped at the cost of an additional factor in the running time that is polynomial in L.

Notation

We use $\mathbb{N}$ to denote the set of natural numbers, and $0 \in \mathbb{N}$. For $n, \delta \in \mathbb{N}$, let $[n] \coloneqq \{1, 2, \ldots, n\}$ and $[n \pm \delta] \coloneqq \{n-\delta, \ldots, n+\delta\}$. We use log to denote the binary logarithm $\log_2$, and ln to denote the natural logarithm with base e. For $x \in \mathbb{R}$, we adopt the convention that $\infty \cdot x$ is equal to 0 if $x = 0$, to ∞ if $x > 0$, and to $-\infty$ if $x < 0$. By convention, the minimum over an empty set is defined to be ∞.

We use the $\mathcal{O}^\star$-notation to hide polynomial factors in the following way: If function $f(n)$ is in $\mathcal{O}^\star(g(n))$, then $f(n) \in \mathcal{O}(g(n)\cdot n^c)$ for some constant c.

The Iverson bracket $[\![E]\!]$ is used to denote the indicator variable of predicate E, i.e. it takes value 1 if E is true, and value 0 otherwise.

For a set X, we say that $X_1, \ldots, X_p \subseteq X$ form a partition of X into p parts if $\bigcup_{i\in[p]} X_i = X$, and for any $i, j \in [p]$, $i \neq j$ we have $X_i \cap X_j = \emptyset$. Note that in this definition we allow (even multiple) parts X_i to be the empty set. For a multiset of integers Y, we write $\Sigma(Y) \coloneqq \sum_{y\in Y} y$. We say that $Y_1, \ldots, Y_p \subseteq Y$ is a partition of Y into p parts if for any fixed ordering of the elements in $Y = \{y_1, \ldots, y_n\}$, the sets $X_1, \ldots, X_p$ defined as $X_i \coloneqq \{j \mid y_j \in Y_i\}$ for all $i \in [p]$ form a partition of $[n]$. We use the shorthand notation $Y = Y_1 \uplus \cdots \uplus Y_p$ to denote that $Y_1, \ldots, Y_p$ is a partition of Y into p parts.

For a collection $\mathcal{W}$ of subsets of $[n]$, we define the upward closure of $\mathcal{W}$ as $\uparrow\mathcal{W} \coloneqq \{W' \supseteq W \mid W \in \mathcal{W}, W' \subseteq [n]\}$, and the downward closure of $\mathcal{W}$ as $\downarrow\mathcal{W} \coloneqq \{W' \subseteq W \mid W \in \mathcal{W}\}$.

In the context of $P \mid\mid \Sigma w_j C_j$ and $P \mid\mid \Sigma w_j U_j$, job j has processing time p_j and weight w_j. We denote $p(S) \coloneqq \sum_{j \in S} p_j$ and $w(S) \coloneqq \sum_{j \in S} w_j$ for any $S \subseteq [n]$.

In the context of Bin Packing algorithms, the term *with high probability* means a probability of at least $1 - n^{-\Omega(1)}$, where n is the number of items.

Binomial Coefficients

For a set S and non-negative integer $i \leqslant |S|$, the set of subsets of S of size i is denoted by $\binom{S}{i}$. Further, for non-negative real number x, let $\binom{S}{\leqslant x} \coloneqq \bigcup_{j=0}^{\lfloor x \rfloor} \binom{S}{j}$. For integer n and non-negative real number r we additionally define $\binom{n}{\leqslant r} \coloneqq \sum_{i=0}^{\lfloor r \rfloor} \binom{n}{i}$.

We use the following bound for $\alpha \in [0,1]$ (see [13, Page 353]):

$$\binom{n}{\alpha n} \leqslant 2^{H(\alpha) n}$$

where $H(\cdot)$ is the entropy function defined as $H(\alpha) \coloneqq -\alpha \log_2(\alpha) - (1-\alpha)\log_2(1-\alpha)$ for $\alpha \in (0,1)$ and $H(\alpha) = 0$ for $\alpha \in \{0,1\}$. Note that $H(\alpha) \in (0,1)$ for $\alpha \in (0,1) \setminus \{1/2\}$, so $2^{H(\alpha)} \in (1,2)$. Hence, by setting $\varepsilon_\alpha \coloneqq 2 - 2^{H(\alpha)} \in (0,1)$ for $\alpha \in (0,1) \setminus \{1/2\}$, we can rewrite the above bound as

$$\binom{n}{\alpha n} \leqslant (2 - \varepsilon_\alpha)^n.$$

Plugging in the value $\alpha = 1/4$, and by leveraging the fact that $\binom{n}{r_1} \leqslant \binom{n}{r_2}$ when $r_1 \leqslant r_2 \leqslant n/2$, we obtain the following bound:

$$\binom{n}{\leqslant n/4} \leqslant \mathcal{O}^\star(2^{H(\frac{1}{4})n}) \leqslant \mathcal{O}(1.7548^n). \tag{1}$$

Note that in Equation (1) we round up the exponent base to 4 digits after the decimal point. This rounding up additionally allows us to replace $\mathcal{O}^\star(\cdot)$ with $\mathcal{O}(\cdot)$.

Technical Lemma

We prove the following general result which can be applied to $P \mid\mid \Sigma w_j C_j$ or $P \mid\mid \Sigma w_j U_j$ by setting f_j to C_j or U_j respectively.

▶ **Lemma 8.** *Let $P \mid\mid \Sigma w_j f_j$ be a scheduling problem on m parallel identical machines and n jobs where the objective is to minimize $\sum_{j=1}^{n} w_j f_j$, where f_j is a cost function depending on the job j and its completion time C_j. For $i \in [m]$ and $S \subseteq [n]$, let $\texttt{OPT}(i, S)$ denote the minimum value of $\sum_{j \in S} w_j f_j$ for the instance restricted to i machines and the jobs in S.*

Then, for all $i \in \{2, 3, \ldots, m\}$, $S \subseteq [n]$, and $j \in [i-1]$, the following holds true:

$$\texttt{OPT}(i,S) = \min_{S' \subseteq S} \{\texttt{OPT}(j,S') + \texttt{OPT}(i-j, S \setminus S')\} = \min_{S' \in \binom{S}{\leqslant j|S|/i}} \{\texttt{OPT}(j,S') + \texttt{OPT}(i-j, S \setminus S')\}.$$

Proof. Consider an optimal schedule of the jobs S on i machines. Let $S^* \subseteq S$ be the jobs scheduled on the j machines containing the least number of total jobs; therefore $|S^*| \leqslant j|S|/i$. Since the objective is the weighted sum of the cost functions over all jobs, $\sum_{i \in [S]} w_j f_j$, the jobs S^* must be scheduled optimally on j machines, and the jobs $S \setminus S^*$ must be also scheduled optimally on $i-j$ machines. This gives us $\texttt{OPT}(i,S) = \texttt{OPT}(j,S^*) + \texttt{OPT}(i-j, S \setminus S^*)$. Moreover, as $S^* \in \binom{S}{\leqslant j|S|/i}$, we have

$$\texttt{OPT}(i,S) = \texttt{OPT}(j,S^*) + \texttt{OPT}(i-j, S \setminus S^*) \geqslant \min_{S' \in \binom{S}{\leqslant j|S|/i}} \{\texttt{OPT}(j,S') + \texttt{OPT}(i-j, S \setminus S')\}. \tag{2}$$

Next, consider a subset $S^\dagger \in \arg\min_{S' \subseteq S}\{\texttt{OPT}(j, S') + \texttt{OPT}(i-j, S \setminus S')\}$. One way of scheduling the jobs S on i machines is scheduling the jobs $S^\dagger$ on j machines optimally, and scheduling the jobs $S \setminus S^\dagger$ on $i-j$ machines optimally. Such a schedule has the weighted sum of cost functions equal to $\texttt{OPT}(i, S^\dagger) + \texttt{OPT}(i-j, S \setminus S^\dagger)$ and this can not be less than $\texttt{OPT}(i, S)$. This gives us,

$$\texttt{OPT}(i, S) \leqslant \texttt{OPT}(j, S^\dagger) + \texttt{OPT}(i-j, S \setminus S^\dagger) = \min_{S' \subseteq S}\{\texttt{OPT}(j, S') + \texttt{OPT}(i-j, S \setminus S')\}. \tag{3}$$

Finally, since $S' \in \binom{S}{\leqslant j|S|/i}$ implies $S' \subseteq S$, we have

$$\min_{S' \in \binom{S}{\leqslant j|S|/i}}\{\texttt{OPT}(j, S') + \texttt{OPT}(i-j, S \setminus S')\} \geqslant \min_{S' \subseteq S}\{\texttt{OPT}(j, S') + \texttt{OPT}(i-j, S \setminus S')\}. \tag{4}$$

Combining Equations (2)–(4), we get

$$\begin{aligned}\texttt{OPT}(i, S) &\leqslant \min_{S' \subseteq S}\{\texttt{OPT}(j, S') + \texttt{OPT}(i-j, S \setminus S')\} \\ &\leqslant \min_{S' \in \binom{S}{\leqslant j|S|/i}}\{\texttt{OPT}(j, S') + \texttt{OPT}(i-j, S \setminus S')\} \leqslant \texttt{OPT}(i, S).\end{aligned}$$

Hence, equality must hold throughout, which proves the lemma. ◀

4 Weighted completion time

In this section we focus on the problem of minimizing the total weighted completion time $P \mid\mid \Sigma w_j C_j$ and prove Theorem 1. The special cases for a fixed number of machines (Theorem 3) are studied in Appendix A.

▶ **Theorem 1.** *$P \mid\mid \Sigma w_j C_j$ can be solved in time $\mathcal{O}(2.755^n)$.*

We prove Theorem 1 using dynamic programming. The key observation is that the base cases, computing the dynamic programming tables for $m \leqslant 3$ machines, can be handled efficiently. Therefore, we first show how to solve $Pm \mid\mid \Sigma w_j C_j$ on $m \in \{1, 2, 3\}$ machines, before presenting the dynamic program used to obtain Theorem 1. This is done by exploiting the meet-in-the-middle paradigm combined with an efficient data structure that supports linear programming queries in m dimensions.

Recall that for $m = 1$ machine, the following observation due to Smith [39] implies that the problem can be solved by simply sorting the jobs. Additionally, we assume without loss of generality that the given jobs are sorted by their ratio of processing time to weight.

▶ **Lemma 9** (Smith's rule [39]). *In an optimal schedule for an instance of $P \mid\mid \Sigma w_j C_j$, jobs on the same machine are scheduled in non-decreasing order of their ratio of processing time to weights.*

To efficiently solve $Pm \mid\mid \Sigma w_j C_j$ for $m \in \{2, 3\}$, we use the meet-in-the-middle paradigm. For $X \subseteq [n]$, let $\texttt{cost}(X)$ be the minimum weighted completion time for scheduling the jobs X on a single machine. In the following lemma, we show how to combine the cost of two disjoint subset of jobs.

▶ **Lemma 10.** *Consider an instance of $P \mid\mid \Sigma w_j C_j$ where the n jobs satisfy $\frac{p_1}{w_1} \leqslant \frac{p_2}{w_2} \leqslant \cdots \leqslant \frac{p_n}{w_n}$. Let $a \in [n]$, $A = [a]$, and $B = [n] \setminus A$. Consider partitions $X_1 \uplus \cdots \uplus X_m = A$ and $Y_1 \uplus \cdots \uplus Y_m = B$. Then, the minimum weighted completion time of a schedule where the jobs $X_i \cup Y_i$ are scheduled on the i-th machine for $i \in [m]$, is given by*

$$\sum_{i\in[m]} \texttt{cost}(X_i \cup Y_i) = \sum_{i\in[m]} \texttt{cost}(X_i) + z_0 + \sum_{i\in[m-1]} z_i \cdot p(X_i),$$

where $z_0 = \sum_{i\in[m]} \texttt{cost}(Y_i) + \left(w(B) - \sum_{r\in[m-1]} w(Y_r)\right) \cdot p(A)$, and for every $i \in [m-1]$, $z_i = w(Y_i) - w(B) + \sum_{r\in[m-1]} w(Y_r)$.

Proof. Consider the optimal schedule that processes jobs $X_i \cup Y_i$ on the i-th machine. Then by Smith's rule (Lemma 9), the i-th machine first processes jobs in X_i in order of their indices, and then processes jobs in Y_i in order of their indices.

Let C_j be the completion time of job $j \in [n]$ in the optimal schedule and denote by $C_j^{(i)} \coloneqq \sum_{k\in Y_i, k\leqslant j} p_k$ the completion time of job $j \in Y_i$ in an optimal schedule of Y_i on a single machine. In particular, $\texttt{cost}(Y_i) = \sum_{j\in Y_i} w_j C_j^{(i)}$. Since the jobs in X_i complete at time $p(X_i)$, we have $C_j = p(X_i) + C_j^{(i)}$ for every job $j \in Y_i$. Therefore, the total cost of scheduling $X_i \cup Y_i$ on a single machine amounts to

$$\begin{aligned}
\texttt{cost}(X_i \cup Y_i) &= \texttt{cost}(X_i) + \sum_{j\in Y_i} w_j \left(C_j^{(i)} + p(X_i)\right) \\
&= \texttt{cost}(X_i) + \sum_{j\in Y_i} w_j C_j^{(i)} + p(X_i) \cdot \sum_{j\in Y_i} w_j \\
&= \texttt{cost}(X_i) + \texttt{cost}(Y_i) + p(X_i) \cdot w(Y_i).
\end{aligned}$$

This implies that the minimum weighted completion time of the entire schedule is

$$\begin{aligned}
\sum_{i\in[m]} \texttt{cost}(X_i \cup Y_i) &= \sum_{i\in[m]} \left(\texttt{cost}(X_i) + \texttt{cost}(Y_i) + p(X_i) \cdot w(Y_i)\right) \\
&= \sum_{i\in[m]} \texttt{cost}(X_i) + \sum_{i\in[m]} \texttt{cost}(Y_i) + \sum_{i\in[m-1]} p(X_i) w(Y_i) \\
&\quad + \left(p(A) - \sum_{i\in[m-1]} p(X_i)\right) \cdot \left(w(B) - \sum_{r\in[m-1]} w(Y_r)\right) \\
&= \sum_{i\in[m]} \texttt{cost}(X_i) + \sum_{i\in[m-1]} \left(w(Y_i) - w(B) + \sum_{r\in[m-1]} w(Y_r)\right) \cdot p(X_i) \\
&\quad + \sum_{i\in[m]} \texttt{cost}(Y_i) + \left(w(B) - \sum_{r\in[m-1]} w(Y_r)\right) \cdot p(A) \\
&= \sum_{i\in[m]} \texttt{cost}(X_i) + z_0 + \sum_{i\in[m-1]} z_i \cdot p(X_i).
\end{aligned}$$

◀

Now, we are ready to prove Lemma 11.

▶ **Lemma 11.** *$Pm \mid\mid \Sigma w_j C_j$ can be solved in $\mathcal{O}^\star(m^{n/2})$ time, for $m \in \{2, 3\}$.*

Proof. Consider an instance of $Pm \mid\mid \Sigma w_j C_j$ on $m \in \{2,3\}$ machines and n jobs with processing times $p_1, \ldots, p_n$ and weights $w_1, \ldots, w_n$. Split the jobs into two halves $A \coloneqq [\lfloor n/2 \rfloor]$ and $B \coloneqq [n] \setminus A$. Broadly, the algorithm will do the following: guess the optimal partition $X_1 \uplus \cdots \uplus X_m = A$ such that jobs in X_i are processed on the i-th machine in the optimal schedule and then use the data structure from Lemma 5 to compute the minimum weighted completion time when jobs in $X_i \cup Y_i$ are processed on the i-th machine, over all partitions $Y_1 \uplus \cdots \uplus Y_m = B$. The details in the remainder of the proof are presented in the following structure: (i) building a linear representation of the minimum weighted completion time for specific schedules using Lemma 10, (ii) designing an algorithm which leverages this linear representation, (iii) arguing the correctness of the algorithm and (iv) analyzing its running time.

Linear representation of the minimum weighted completion time. Consider partitions $X_1 \uplus \cdots \uplus X_m = A$ and $Y_1 \uplus \cdots \uplus Y_m = B$ such that $X_i \cup Y_i$ is optimally scheduled on the i-th machine for $i \in [m]$. By Lemma 10, the minimum weighted completion time of such a schedule is

$$\sum_{i \in [m]} \texttt{cost}(X_i \cup Y_i) = \sum_{i \in [m]} \texttt{cost}(X_i) + z_0 + \sum_{i \in [m-1]} z_i \cdot p(X_i),$$

where $z_0 = \sum_{i \in [m]} \texttt{cost}(Y_i) + \left(w(B) - \sum_{r \in [m-1]} w(Y_r) \right) \cdot p(A)$, and for every $i \in [m-1]$, $z_i = w(Y_i) - w(B) + \sum_{r \in [m-1]} w(Y_r)$. Notice that $z_0, z_1, \ldots, z_{m-1}$ depend only on Y_1, …, Y_m, A, B, but *not* on X_1, …, X_m. Define

$$\mathcal{R} \coloneqq \{(z_0, z_1, \ldots, z_{m-1}) \mid Y_1 \uplus \cdots \uplus Y_m = B \text{ and } z_0, z_1, \ldots, z_{m-1} \text{ are as defined above}\}. \quad (5)$$

Now, for a fixed partition $X_1 \uplus \cdots \uplus X_m = A$ corresponding to a schedule of jobs in A across the machines, we extend such a schedule optimally to also include jobs in B. The minimum weighted completion time of this is given by the minimum of $\sum_{i \in [m]} \texttt{cost}(X_i \cup Y_i)$ over all partitions $Y_1 \uplus \cdots \uplus Y_m = B$. This is equal to

$$\begin{aligned} &\min_{(z_0, z_1, \ldots, z_{m-1}) \in \mathcal{R}} \left\{ \sum_{i \in [m]} \texttt{cost}(X_i) + z_0 + \sum_{i \in [m-1]} z_i \cdot p(X_i) \right\} \\ &\quad = \sum_{i \in [m]} \texttt{cost}(X_i) + \min_{(z_0, z_1, \ldots, z_{m-1}) \in \mathcal{R}} \left\{ \sum_{i=0}^{m-1} z_i \cdot x_i \right\}, \end{aligned}$$

where $x_0 = 1$ and $x_i = p(X_i)$ for all $i \in [m-1]$.

Description of the algorithm. First, we compute $\mathcal{R}$ using Equation (5). Then, we use Lemma 5 to set up a data structure $\mathcal{D}$ for $m \in \{2,3\}$ dimensions, preprocessed with all elements $(z_0, z_1, \ldots, z_{m-1}) \in \mathcal{R}$. Recall that the data structure $\mathcal{D}$ is capable of answering linear programming queries of the form $\min_{(z_0, z_1, \ldots, z_{m-1}) \in \mathcal{R}} \left\{ \sum_{i=0}^{m-1} z_i \cdot x_i \right\}$ for any query point $(x_0, \ldots, x_{m-1})$. We then iterate over all partitions $X_1 \uplus \cdots \uplus X_m = A$, and for each partition we compute the value $\xi = \sum_{i \in [m]} \texttt{cost}(X_i)$. Next, we set $(x_0 = 1, x_1 = p(X_1), \ldots, x_{m-1} = p(X_{m-1}))$, and query $\mathcal{D}$ for the value $q = \min_{(z_0, z_1, \ldots, z_{m-1}) \in \mathcal{R}} \left\{ \sum_{i=0}^{m-1} z_i \cdot x_i \right\}$. Finally, we output the minimum value of $(\xi + q)$ over all partitions $X_1 \uplus \cdots \uplus X_m = A$. We present a pseudocode in Algorithm 1.

Algorithm 1 $Pm \mid\mid \Sigma w_j C_j$ for jobs $i \in [n]$ having weights w_i and processing times p_i; $m \in \{2, 3\}$.

Sort jobs $i \in [n]$ in non-decreasing order of $\frac{p_i}{w_i}$.
$A \leftarrow [\lfloor n/2 \rfloor]$, $B \leftarrow [n] \setminus A$
$\mathcal{R} \leftarrow \{(z_0, z_1, \ldots, z_{m-1}) \mid Y_1 \uplus \cdots \uplus Y_m = B\}$ ▷ as defined in Equation (5)
Use Lemma 5 to set up a data structure $\mathcal{D}$ with all $(z_0, z_1, \ldots, z_{m-1}) \in \mathcal{R}$
$\texttt{ans} \leftarrow \infty$
for $X_1 \uplus \cdots \uplus X_m = A$ **do**
    $\xi \leftarrow \sum_{i \in [m]} \texttt{cost}(X_i)$
    $x_0 \leftarrow 1$, $x_1 \leftarrow p(X_1)$, $\ldots$, $x_{m-1} \leftarrow p(X_{m-1})$
    Query $\mathcal{D}$ for $q \leftarrow \min_{(z_0, z_1, \ldots, z_{m-1}) \in \mathcal{R}} \left\{ \sum_{i=0}^{m-1} z_i \cdot x_i \right\}$
    $\texttt{ans} \leftarrow \min(\texttt{ans}, \xi + q)$
end for
return ans

Proof of correctness. Fix a partition $X_1 \uplus \cdots \uplus X_m = A$. The discussion on the mathematical representation of the weighted completion time implies that the computed value $(\xi + q) = \sum_{i \in [m]} \texttt{cost}(X_i) + \min_{(z_0, z_1, \ldots, z_{m-1}) \in \mathcal{R}} \left\{ \sum_{i=0}^{m-1} z_i \cdot x_i \right\}$ is equal to the minimum total weighted completion time of all input jobs, when X_i is scheduled completely on the i-th machine. The algorithm outputs the minimum value of $(\xi + q)$ over all choices of $X_1 \uplus \cdots \uplus X_m = A$, and hence correctly outputs the minimum weighted completion time for the input instance.

Running-time analysis. Observe that $|\mathcal{R}| \leqslant m^{|B|} \leqslant m^{n/2+1}$, as this bounded by the number of partitions $Y_1 \uplus \cdots \uplus Y_m = B$. Moreover, $\mathcal{R}$ can be computed in time $\mathcal{O}^\star(m^{n/2})$. Preprocessing the data structure $\mathcal{D}$ with all $(z_0, \ldots, z_{m-1}) \in \mathcal{R}$ takes time $\mathcal{O}(|\mathcal{R}| \log |\mathcal{R}|) = \mathcal{O}^\star(m^{n/2})$, and any query can be answered in $\mathcal{O}(\log |\mathcal{R}|) = \mathcal{O}(n)$ time (Lemma 5). In total, the number of queries made to $\mathcal{D}$ is $\mathcal{O}(m^{|A|})$, one for each partition $X_1 \uplus \cdots \uplus X_m = A$. Hence, a total time of $\mathcal{O}(n) \cdot \mathcal{O}(m^{|A|}) = \mathcal{O}^\star(m^{n/2})$ is spent over all queries. Therefore, the algorithm takes a total time of $\mathcal{O}^\star(m^{n/2})$. ◀

We are now ready to prove Theorem 1.

Proof of Theorem 1. Consider an instance of $P \mid\mid \Sigma w_j C_j$, i.e. consider n jobs with processing times $p_1, \ldots, p_n$ and weights $w_1, \ldots, w_n$, and let m be the number of available identical parallel machines. Note that if $m \geqslant n$, then each job can be processed at time 0 on a distinct machine so that its completion time is precisely p_j. Thus, the optimal weighted completion time is simply $\sum_{i=1}^{n} w_i p_i$. Hence, suppose that $m < n$. We construct a DP table such that for any subset $S \subseteq [n]$ and $i \in [m]$, the table entry $\texttt{DP}[i, S]$ contains the minimum total weighted completion time when the jobs in S are scheduled on i machines.

For every $S \subseteq [n]$, initialize $\texttt{DP}[1, S]$ using Lemma 9, and initialize $\texttt{DP}[2, S]$ and $\texttt{DP}[3, S]$ using Lemma 11. By Lemma 8, used with $j = 1$, the following recurrence holds for any $i \geqslant 4$ and $S \subseteq [n]$.

$$\texttt{DP}[i, S] = \min_{S' \in \binom{S}{\leqslant |S|/i}} \{\texttt{DP}[1, S'] + \texttt{DP}[i-1, S \setminus S']\}.$$

We can therefore iterate over $i \in \{4, \ldots, m\}$ and $S \subseteq [n]$ in increasing order, and compute all table entries $\texttt{DP}[i, S]$ using the above recurrence. In the end, we output $\texttt{DP}[m, [n]]$.

The computed value of $\texttt{DP}[m,[n]]$ correctly contains the minimum weighted completion time of scheduling all jobs on m machines by Lemmas 8, 9, and 11 and thus our algorithm is correct. To analyze the running time, note that the initialization for $i=1$ takes time $\mathcal{O}^\star(2^n)$ and by Lemma 11, the initialization for $i=2$ and $i=3$ takes time at most

$$\sum_{S\subseteq[n]} \mathcal{O}^\star\left(3^{|S|/2}\right) = \mathcal{O}^\star\left(\sum_{r=0}^{n}\binom{n}{r}\cdot\left(\sqrt{3}\right)^r\right) = \mathcal{O}^\star\left(\left(1+\sqrt{3}\right)^n\right) = \mathcal{O}(2.733^n).$$

Each transition in the above recurrence takes time $\mathcal{O}(\binom{S}{\leqslant |S|/i})$ and thus the total time to compute $\texttt{DP}[i,S]$ for all $S\subseteq[n]$ and $i\in\{4,\ldots,m\}$ is at most

$$\begin{aligned}\sum_{i=4}^{m}\sum_{S\subseteq[n]}\mathcal{O}\left(\binom{S}{\leqslant |S|/i}\right) &\leqslant \sum_{S\subseteq[n]}\mathcal{O}^\star\left(\binom{S}{\leqslant |S|/4}\right)\\ &\leqslant \sum_{S\subseteq[n]}\mathcal{O}^\star\left(1.7548^{|S|}\right)\\ &= \sum_{r=0}^{n}\binom{n}{r}\cdot\mathcal{O}^\star\left(1.7548^r\right) = \mathcal{O}^\star\left(2.7548^n\right).\end{aligned}$$

The second inequality uses the bound $\binom{|S|}{\leqslant |S|/4} = \mathcal{O}(1.7548^{|S|})$ from Equation (1). In total, the algorithm runs in time $\mathcal{O}(2.733^n) + \mathcal{O}^\star(2.7548^n) \leqslant \mathcal{O}(2.755^n)$. ◀

5 Weighted number of tardy jobs

In this section, we focus on the problem of minimizing the total weight of tardy jobs $P\,||\,\Sigma w_jU_j$ defined in Section 2. We will use the Fast Subset Convolution algorithm due to Björklund et al. [3].

▶ **Lemma 12** (Fast Subset Convolution, follows from [3, Theorem 3]). *Given functions $f,g:2^{[n]}\to\{0\}\cup[n]$, the min-plus subset convolution $h(S)=\min_{S'\subseteq S}\{f(S')+g(S\setminus S')\}$ can be computed in time $\mathcal{O}^\star(2^n)$.*

We start by recalling an algorithm for solving the unweighted version of this problem on a single machine, this problem is denoted as $P1\,||\,\Sigma U_j$.

▶ **Lemma 13** (Moore's algorithm [32]). *$P1\,||\,\Sigma U_j$ can be solved in $\mathcal{O}(n\log n)$ time.*

We prove Theorem 2 via standard dynamic programming, where the base case is handled using Lemma 13, and the transitions are handled using Lemma 12.

▶ **Theorem 2.** *$P\,||\,\Sigma w_jU_j$ can be solved in time $\mathcal{O}^\star(2^n)$.*

Proof. Consider an instance of $P\,||\,\Sigma w_jU_j$, i.e., consider n jobs with processing times $p_1,\ldots,p_n$, weights $w_1,\ldots,w_n$ and due dates $d_1,\ldots,d_n$, and let m be the number of available identical parallel machines. Note that if $m\geqslant n$, then each job can be processed at time 0 on a distinct machine so that it is tardy if and only if $p_j>d_j$. Thus, the optimal weighted number of tardy jobs is simply $\sum_{i=1}^{n} w_i\cdot[\![p_i>d_i]\!]$. Hence, suppose that $m<n$. We construct a DP table such that for any subset $S\subseteq[n]$ and $i\in[m]$, the table entry $\texttt{DP}[i,S]$ contains the minimum total number of tardy jobs when the jobs in S are scheduled on i machines. In other words, $\texttt{DP}[i,S]$ stores the optimum for the *unweighted version* of the problem on i machines with jobs S. We prove the following claim.

▷ Claim 14. The minimum total weight of tardy jobs is equal to

$$\min\{w([n] \setminus S) \mid S \subseteq [n], \texttt{DP}[m, S] = 0\}.$$

Proof. Let the minimum total weight of tardy jobs be $\texttt{OPT}$. Fix an optimal schedule. Let $S^* \subseteq [n]$ be the set of jobs which are not tardy in this schedule. Since the jobs S^* can be scheduled on m machines with every job meeting its deadline, we have $\texttt{DP}[m, S^*] = 0$. Moreover, the total weight of tardy jobs is $w([n] \setminus S^*)$. Therefore, we have

$$\texttt{OPT} = w([n] \setminus S^*) \geqslant \min\{w([n] \setminus S) \mid S \subseteq [n], \texttt{DP}[m, S] = 0\}.$$

On the other direction, let $S^\dagger \subseteq [n]$ be a subset with $\texttt{DP}[m, S^\dagger] = 0$ which minimizes $w([n] \setminus S^\dagger)$. Since $\texttt{DP}[m, S^\dagger] = 0$, the jobs $S^\dagger$ can be scheduled on m machines with every job meeting its deadline. Extending this schedule by scheduling the remaining jobs $[n] \setminus S^\dagger$ arbitrarily after the already scheduled jobs, we get the total weight of tardy jobs in this schedule to be at most $w([n] \setminus S^\dagger)$, and this must be at least $\texttt{OPT}$. Therefore,

$$\texttt{OPT} \leqslant w([n] \setminus S^\dagger) = \min\{w([n] \setminus S) \mid S \subseteq [n], \texttt{DP}[m, S] = 0\}.$$

The above inequalities together imply $\texttt{OPT} = \min\{w([n] \setminus S) \mid S \subseteq [n], \texttt{DP}[m, S] = 0\}$. ◁

We now describe the algorithm. For every $S \subseteq [n]$, initialize $\texttt{DP}[1, S]$ using Lemma 13. By Lemma 8, the following recurrence holds for any $i \geqslant 2$ and $S \subseteq [n]$.

$$\texttt{DP}[i, S] = \min_{S' \subseteq S} \{\texttt{DP}[1, S'] + \texttt{DP}[i - 1, S \setminus S']\}.$$

We can therefore iterate over $i \in \{2, \ldots, m\}$ in increasing order, and compute all table entries $\texttt{DP}[i, S]$ using Fast Subset Convolution (Lemma 12) with the above recurrence. In the end, we output $\min\{w([n] \setminus S) \mid S \subseteq [n], \texttt{DP}[m, S] = 0\}$.

The correctness of the algorithm follows from Claim 14 and Lemmas 8 and 12. To analyze the running time, note that the initialization for $i = 1$ takes time at most $\mathcal{O}(2^n) \cdot \mathcal{O}(n \log n) = \mathcal{O}^\star(2^n)$. For every $i \in \{2, \ldots, m\}$, Fast Subset Convolution using Lemma 12 takes time $\mathcal{O}^\star(2^n)$, as all entries in $\texttt{DP}$ are at most n. Finally, computing $\min\{w([n] \setminus S) \mid S \subseteq [n], \texttt{DP}[m, S] = 0\}$ takes a total time of $\mathcal{O}^\star(2^n)$. The total running time adds up to $\mathcal{O}^\star(2^n + m \cdot 2^n + 2^n) = \mathcal{O}^\star(2^n)$. ◀

References

1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Scheduling lower bounds via AND subset sum. *J. Comput. Syst. Sci.*, 127:29–40, 2022. doi:10.1016/J.JCSS.2022.01.005.

2 Andreas Björklund, Radu Curticapean, Thore Husfeldt, Petteri Kaski, and Kevin Pratt. Fast deterministic chromatic number under the asymptotic rank conjecture. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2804–2818. SIAM, 2025. doi:10.1137/1.9781611978322.91.

3 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.

4 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. doi:10.1137/070683933.

5 Andreas Björklund and Petteri Kaski. The asymptotic rank conjecture and the set cover conjecture are not both true. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 859–870. ACM, 2024. doi:10.1145/3618260.3649656.

6 Andreas Björklund, Petteri Kaski, Tomohiro Koana, and Jesper Nederlof. Kronecker Scaling of Tensors with Applications to Arithmetic Circuits and Algorithms. In Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis, editors, *53rd International Colloquium on Automata, Languages, and Programming (ICALP 2026)*, volume 374 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:24, Dagstuhl, Germany, 2026. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2026.36.

7 Cornelius Brand, Radu Curticapean, Baitian Li, and Kevin Pratt. Faster convolutions: Yates and strassen revisited. In Sepehr Assadi and Eva Rotenberg, editors, *2026 Symposium on Simplicity in Algorithms, SOSA 2026, Vancouver, BC, Canada, January 12-14, 2026*, pages 328–339. SIAM, 2026. doi:10.1137/1.9781611978964.25.

8 Karl Bringmann, Anita Dürr, and Karol Węgrzycki. Tight (S)ETH-Based Lower Bounds for Pseudopolynomial Algorithms for Bin Packing and Multi-machine Scheduling. In Aditya Bhaskara and Artur Czumaj, editors, *Proceedings of the 58th Annual ACM Symposium on Theory of Computing, STOC 2026, Salt Lake City, UT, USA, June 22-26, 2026*, pages 2094–2105. ACM, 2026. doi:10.1145/3798129.3800913.

9 Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. Faster minimization of tardy processing time on a single machine. *Algorithmica*, 84(5):1341–1356, 2022. doi:10.1007/S00453-022-00928-W.

10 John L. Bruno, Edward G. Coffman Jr., and Ravi Sethi. Scheduling independent tasks to reduce mean finishing time. *Commun. ACM*, 17(7):382–387, 1974. doi:10.1145/361011.361064.

11 Timothy M. Chan. Fixed-dimensional linear programming queries made easy. In Sue Whitesides, editor, *Proceedings of the Twelfth Annual Symposium on Computational Geometry, Philadelphia, PA, USA, May 24-26, 1996*, pages 284–290. ACM, 1996. doi:10.1145/237218.237397.

12 Timothy M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discret. Comput. Geom.*, 16(4):361–368, 1996. doi:10.1007/BF02712873.

13 Thomas M. Cover and Joy A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006. doi:10.1002/047174882X.

14 Jianzhong Du and Joseph Y.-T. Leung. Minimizing total tardiness on one machine is np-hard. *Math. Oper. Res.*, 15(3):483–495, 1990. doi:10.1287/MOOR.15.3.483.

15 Anita Dürr, Evangelos Kipouridis, Michael Lampis, and Karol Węgrzycki. Faster Algorithms for k-Orthogonal Vectors in Low Dimension. In *53rd International Colloquium on Automata, Languages, and Programming, ICALP 2026, Royal Holloway, University of London, Egham, United Kingdom, July 7-10, 2026*, volume 374 of *LIPIcs*, pages 85:1–85:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPIcs.ICALP.2026.85.

16 Nick Fischer and Leo Wennmann. Minimizing tardy processing time on a single machine in near-linear time. *TheoretiCS*, 4, 2025. doi:10.46298/THEORETICS.25.14.

17 George Gens and Eugene Levner. Fast approximation algorithm for job sequencing with deadlines. *Discret. Appl. Math.*, 3(4):313–318, 1981. doi:10.1016/0166-218X(81)90008-1.

18 Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5(2):287–326, 1979. doi:10.1016/S0167-5060(08)70356-X.

19 Leonidas J. Guibas, Jorge Stolfi, and Kenneth L. Clarkson. Solving related two-and three-dimensional linear programming problems in logarithmic time. *Theor. Comput. Sci.*, 49:81–84, 1987. doi:10.1016/0304-3975(87)90101-0.

20 Klaus Heeger and Danny Hermelin. Minimizing the Weighted Number of Tardy Jobs Is W[1]-Hard. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, Royal Holloway, London, United Kingdom, September 2-4, 2024*, volume 308 of *LIPIcs*, pages 68:1–68:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ESA.2024.68.

21 Danny Hermelin, Tomohiro Koana, and Dvir Shabtay. Faster minimization of total weighted completion time on parallel machines. *CoRR*, abs/2502.13631, 2025. doi:10.48550/arXiv.2502.13631.

22 Danny Hermelin, Hendrik Molter, and Dvir Shabtay. Minimizing the weighted number of tardy jobs via (max,+)-convolutions. *INFORMS J. Comput.*, 36(3):836–848, 2024. doi:10.1287/IJOC.2022.0307.

23 Klaus Jansen, Felix Land, and Kati Land. Bounding the running time of algorithms for scheduling and packing problems. *SIAM J. Discret. Math.*, 30(1):343–366, 2016. doi:10.1137/140952636.

24 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

25 Kim-Manuel Klein, Adam Polak, and Lars Rohwedder. On Minimizing Tardy Processing Time, Max-Min Skewed Convolution, and Triangular Structured ILPs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2947–2960. SIAM, 2023. doi:10.1137/1.9781611977554.CH112.

26 Dusan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *J. Sched.*, 21(5):493–503, 2018. doi:10.1007/S10951-017-0550-0.

27 Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys. Chapter 9 Sequencing and scheduling: Algorithms and complexity. In Stephen C. Graves, Alexander H. G. Rinnooy Kan, and Paul Herbert Zipkin, editors, *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445–522. North-Holland, 1993. doi:10.1016/S0927-0507(05)80189-6.

28 Eugene L. Lawler and J. Michael Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969. doi:10.1287/mnsc.16.1.77.

29 Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, pages 343–362. Elsevier, 1977. doi:10.1016/S0167-5060(08)70743-X.

30 Christophe Lenté, Mathieu Liedloff, Ameur Soukhal, and Vincent T'kindt. Exponential algorithms for scheduling problems. Technical report, Ecole Polytechnique de l'Université de Tours, 2014. URL: https://hal.science/hal-00944382v1.

31 William L. Maxwell. On sequencing n jobs on one machine to minimize the number of late jobs. *Management Science*, 16(5):295–297, 1970. doi:10.1287/mnsc.16.5.295.

32 J. Michael Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109, 1968. doi:10.1287/mnsc.15.1.102.

33 Jesper Nederlof. Finding large set covers faster via the representation method. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, Aarhus, Denmark, August 22-24, 2016*, volume 57 of *LIPIcs*, pages 69:1–69:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.69.

34 Jesper Nederlof. An invitation to "Fine-grained complexity of NP-complete problems". *Comput. Sci. Rev.*, 61:100919, 2026. doi:10.1016/J.COSREV.2026.100919.

35 Jesper Nederlof, Jakub Pawlewicz, Céline M. F. Swennenhuis, and Karol Węgrzycki. A Faster Exponential Time Algorithm for Bin Packing With a Constant Number of Bins via Additive Combinatorics. *SIAM J. Comput.*, 52(6):1369–1412, 2023. doi:10.1137/22M1478112.

36 F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20(2):87–93, February 1977. doi:10.1145/359423.359430.

37 Michael H. Rothkopf. Scheduling independent tasks on parallel processors. *Management Science*, 12(5):437–447, 1966. doi:10.1287/mnsc.12.5.437.

38 Sartaj Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 23(1):116–127, 1976. doi:10.1145/321921.321934.

39 Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. doi:10.1002/nav.3800030106.

A Proof of Theorem 3

We restate Theorem 3 for convenience.

▶ **Theorem 3.** *$Pm \mid\mid \Sigma w_j C_j$ can be solved in time:*

- $\mathcal{O}^\star(m^{n/2})$ *if* $m \leqslant 3$,
- $\mathcal{O}(2.389^n)$ *if* $m = 4$,
- $\mathcal{O}(2.726^n)$ *if* $m = 5$,
- $\mathcal{O}(2.733^n)$ *if* $m = 6$.

Note that Lemma 11 proves one of the cases of Theorem 3. We now prove the remaining cases separately as Lemmas 15–17.

▶ **Lemma 15.** *$P4 \mid\mid \Sigma w_j C_j$ can be solved in $\mathcal{O}(2.389^n)$ time.*

Proof. Consider an instance of $P4 \mid\mid \Sigma w_j C_j$, where the input consists of n jobs with processing times $p_1, \ldots, p_n$ and weights $w_1, \ldots, w_n$. Let $\alpha := 0.628$, the choice of α will become clear later in the proof, and let $A = [\lfloor \alpha n \rfloor]$, and $B = [n] \setminus A$. Broadly, the algorithm does the following. Without loss of generality, assume that the first machine schedules the smallest number of jobs from A. Let X_1 be the set of jobs scheduled on the first machine. This implies $|X_1| \leqslant |A|/4$. For every $t \in \{p(X_1) \mid X_1 \subseteq A, |X_1| \leqslant |A|/4\}$, we use Lemma 5 to set up a data structure $\mathcal{D}_t$ which is preprocessed to contain the relevant information for every partition $Y_1 \uplus Y_2 \uplus Y_3 \uplus Y_4 = B$. Next, we iterate over all partitions $X_1 \uplus X_2 \uplus X_3 \uplus X_4$ with $|X_1| \leqslant |X_2| \leqslant |X_4| \leqslant |X_4|$, and query $\mathcal{D}_{p(X_1)}$ for the minimum weighted completion time when $X_i \cup Y_i$ are the jobs scheduled on the i-th machine, over all partitions $Y_1 \uplus Y_2 \uplus Y_3 \uplus Y_4 = B$. Finally, we output the minimum weighted completion time over all choices of $X_1 \uplus X_2 \uplus X_3 \uplus X_4 = A$. The details in the remainder of the proof are presented in the following structure: (i) building a linear representation of the minimum weighted completion time for specific schedules using Lemma 10, (ii) designing an algorithm which leverages this linear representation, (iii) arguing the correctness of the algorithm and (iv) analyzing its running time.

Linear representation of the minimum weighted completion time. Consider partitions $X_1 \uplus X_2 \uplus X_3 \uplus X_4 = A$ and $Y_1 \uplus Y_2 \uplus Y_3 \uplus Y_4 = B$ such that $X_i \cup Y_i$ is optimally scheduled on the i-th machine for $i \in [4]$. By Lemma 10, the minimum weighted completion time of such a schedule amounts to

$$\begin{aligned}\sum_{i\in[4]} \texttt{cost}(X_i \cup Y_i) &= \sum_{i\in[4]} \texttt{cost}(X_i) + z_0 + \sum_{i\in[3]} z_i \cdot p(X_i) \\ &= \sum_{i\in[4]} \texttt{cost}(X_i) + ((z_0 + z_1 \cdot p(X_1)) + z_2 \cdot p(X_2) + z_3 \cdot p(X_3)),\end{aligned}$$

where $z_0 = \sum_{i\in[4]} \texttt{cost}(Y_i) + \left(w(B) - \sum_{r\in[3]} w(Y_r)\right) \cdot p(A)$, and for every $i \in [m-1]$, $z_i = w(Y_i) - w(B) + \sum_{r\in[3]} w(Y_r)$. Let $t = p(X_1)$. This gives us

$$\sum_{i\in[4]} \texttt{cost}(X_i \cup Y_i) = \sum_{i\in[4]} \texttt{cost}(X_i) + ((z_0 + z_1 \cdot t) + z_2 \cdot p(X_2) + z_3 \cdot p(X_3))$$
$$= \sum_{i\in[4]} \texttt{cost}(X_i) + \left(\sum_{i=1}^{3} z'_i \cdot x'_i\right),$$

where $z'_1 = z_0 + z_1 \cdot t$, $z'_2 = z_2$, $z'_3 = z_3$ and $x'_1 = 1$, $x'_2 = p(X_2)$, $x'_3 = p(X_3)$. Notice that z'_1, z'_2, z'_3 depend on Y_1, Y_2, Y_3, Y_4, $t = p(X_1)$, A, B; but *not* on X_2, X_3, X_4.

Without loss of generality, assume $|X_1| \leqslant |X_2| \leqslant |X_3| \leqslant |X_4|$. Therefore, $|X_1| \leqslant |A|/4$. Let $\mathcal{Z} = \{p(X) \mid X \subseteq A, |X| \leqslant |A|/4\}$ and thus $t = p(X_1) \in \mathcal{Z}$. For every $t \in \mathcal{Z}$, define

$$\mathcal{R}_t = \{(z'_1, z'_2, z'_3) \mid Y_1 \uplus Y_2 \uplus Y_3 \uplus Y_4 = B, \text{ and } z'_1, z'_2, z'_3 \text{ as defined above}\}. \tag{6}$$

Now, for a fixed partition $X_1 \uplus X_2 \uplus X_3 \uplus X_4 = A$ with $|X_1| \leqslant |X_2| \leqslant |X_3| \leqslant |X_4|$, corresponding to a schedule of jobs in A across the machines, we extend such a schedule optimally to also include jobs in B. Let $t = p(X_1)$. The minimum total weighted completion time of this is given by the minimum of $\sum_{i\in[4]} \texttt{cost}(X_i \cup Y_i)$ over all partitions $Y_1 \uplus Y_2 \uplus Y_3 \uplus Y_4 = B$. This is equal to

$$\min_{(z'_1,z'_2,z'_3)\in\mathcal{R}_t} \left\{ \sum_{i\in[4]} \texttt{cost}(X_i) + \sum_{i=1}^{3} z'_i \cdot x'_i \right\} = \sum_{i\in[4]} \texttt{cost}(X_i) + \min_{(z'_1,z'_2,z'_3)\in\mathcal{R}_t} \left\{ \sum_{i=1}^{3} z'_i \cdot x'_i \right\}$$

where $x'_1 = 1$, $x'_2 = p(X_2)$ and $x'_3 = p(X_3)$.

Description of the algorithm. First, we compute $\mathcal{Z} = \{p(X) \mid X \subseteq A, |X| \leqslant |A|/4\}$ and then compute $\mathcal{R}_t$ using Equation (6) for all $t \in \mathcal{Z}$. For every $t \in \mathcal{Z}$ do the following: using Lemma 5, set up a data structure $\mathcal{D}_t$ supporting linear programming queries on 3 dimensions, preprocessed with the elements $(z'_1, z'_2, z'_3) \in \mathcal{R}_t$. Note that $\mathcal{D}_t$ is capable of answering linear programming queries of the form $\min_{(z'_1,z'_2,z'_3)\in\mathcal{R}_t} \left\{\sum_{i=1}^{3} z'_i \cdot x'_i\right\}$ for any queried point (x'_1, x'_2, x'_3). We now iterate over all partitions $X_1 \uplus X_2 \uplus X_3 \uplus X_4 = A$ with $|X_1| \leqslant |X_2| \leqslant |X_3| \leqslant |X_4|$, and for each partition do the following. Let $t = p(X_1)$. We compute the value $\xi = \sum_{i\in[4]} \texttt{cost}(X_i)$. Next, we set $(x'_1 = 1, x'_2 = p(X_2), x'_3 = p(X_3))$, and query $\mathcal{D}_t$ for the value $q = \min_{(z'_1,z'_2,z'_3)\in\mathcal{R}_t} \left\{\sum_{i=1}^{3} z'_i \cdot x'_i\right\}$. Finally, we output the minimum value of $(\xi + q)$ over all partitions $X_1 \uplus X_2 \uplus X_3 \uplus X_4 = A$. We present pseudocode of the approach in Algorithm 2.

Proof of correctness. Fix a partition $X_1 \uplus X_2 \uplus X_3 \uplus X_4 = A$ and set $t = p(X_1)$. The discussion on the mathematical representation of the weighted completion time implies that the computed value $(\xi + q) = \sum_{i\in[4]} \texttt{cost}(X_i) + \min_{(z'_1,z'_2,z'_3)\in\mathcal{R}_t} \left\{\sum_{i=1}^{3} z'_i \cdot x'_i\right\}$ is equal to the minimum total weighted completion time when X_i is scheduled completely on the i-th machine. The algorithm outputs the minimum value of $(\xi + q)$ over all choices of $X_1 \uplus X_2 \uplus X_3 \uplus X_4 = A$, and hence correctly outputs the minimum weighted completion time for the input instance.

Algorithm 2 $P4 \mid\mid \Sigma w_j C_j$ for jobs $i \in [n]$ having weights w_i and processing times p_i.

Sort jobs $i \in [n]$ in a non-decreasing order of $\frac{p_i}{w_i}$.
$\alpha \leftarrow 0.628$, $A \leftarrow [\lfloor \alpha n \rfloor]$, $B \leftarrow [n] \setminus A$
$\mathcal{Z} \leftarrow \{p(X) \mid X \subseteq A, |X| \leqslant |A|/4\}$
for $t \in \mathcal{Z}$ **do**
    $\mathcal{R}_t = \{(z'_1, z'_2, z'_3) \mid Y_1 \uplus Y_2 \uplus Y_3 \uplus Y_4 = B\}$ ▷ as defined in Equation (6)
    Use Lemma 5 to set up a data structure $\mathcal{D}_t$ from $\mathcal{R}_t$.
end for
$\texttt{ans} \leftarrow \infty$
for $X_1 \uplus X_2 \uplus X_3 \uplus X_4 = A$ with $|X_1| \leqslant |X_2| \leqslant |X_3| \leqslant |X_4|$ **do**
    $t \leftarrow p(X_1)$
    $\xi \leftarrow \sum_{i \in [4]} \texttt{cost}(X_i)$
    $x'_1 \leftarrow 1$, $x'_2 \leftarrow p(X_2)$, $x'_3 \leftarrow p(X_3)$
    Query $\mathcal{D}_t$ for $q \leftarrow \min_{(z'_1, z'_2, z'_3) \in \mathcal{R}_t} \left\{ \sum_{i=1}^{3} z'_i \cdot x'_i \right\}$
    $\texttt{ans} \leftarrow \min(\texttt{ans}, \xi + q)$
end for
return ans

Running-time analysis. First, observe that $|\mathcal{Z}| \leqslant \binom{|A|}{\leqslant |A|/4} \leqslant \mathcal{O}^\star(1.7548^{|A|})$ by Equation (1), and hence $\mathcal{Z}$ can be computed in time $\mathcal{O}^\star(1.7548^{|A|})$. For every $t \in \mathcal{Z}$, $|\mathcal{R}_t| \leqslant 4^{|B|}$ and $\mathcal{R}_t$ can be computed in time $\mathcal{O}^\star(4^{|B|})$. Moreover, for every $t \in \mathcal{Z}$, preprocessing $\mathcal{D}_t$ takes time $\mathcal{O}(|\mathcal{R}_t| \log |\mathcal{R}_t|) = \mathcal{O}^\star(4^{|B|})$ (Lemma 5). The total preprocessing time across all $t \in \mathcal{Z}$ is $|\mathcal{Z}| \cdot \mathcal{O}^\star(4^{|B|}) = \mathcal{O}^\star(1.7548^{|A|} \cdot 4^{|B|})$. For any $t \in \mathcal{Z}$, a linear programming query to $\mathcal{D}_t$ takes time $\mathcal{O}(\log |\mathcal{R}_t|) = \mathcal{O}(n)$ (Lemma 5). In total, the algorithm makes a total of $\mathcal{O}^\star(4^{|A|})$ queries to the data structures $\mathcal{D}_t$, over all $t \in \mathcal{Z}$. Hence these queries take a total time of $\mathcal{O}(n) \cdot \mathcal{O}^\star(4^{|A|}) = \mathcal{O}^\star(4^{|A|})$. The total running time of the entire algorithm is therefore $\mathcal{O}^\star\left(1.7548^{|A|} \cdot 4^{|B|} + 4^{|A|}\right) = \mathcal{O}^\star\left(1.7548^{\alpha n} \cdot 4^{(1-\alpha)n} + 4^{\alpha n}\right)$. This is minimized when $1.7548^{\alpha n} \cdot 4^{(1-\alpha)n} = 4^{\alpha n}$, i.e. when $\alpha \approx 0.628$. By our choice of $\alpha = 0.628$ this gives us a running time of $\mathcal{O}^\star\left(1.7548^{\alpha n} \cdot 4^{(1-\alpha)n} + 4^{\alpha n}\right) \leqslant \mathcal{O}(2.389^n)$. ◀

To obtain algorithms solving $Pm \mid\mid \Sigma w_j C_j$ for $m \in \{5, 6\}$ in time faster than $\mathcal{O}(2.755^n)$, we crucially use the equalities in Lemma 8 with $j = 3$ as follows. We use a dynamic program similar to the one in the proof of Theorem 3. For any subset $S \subseteq [n]$ and $i \in [m]$, let the table entry $\texttt{DP}[i, S]$ contains the minimum total weighted completion time when the jobs in S are scheduled on i machines.

▶ **Lemma 16.** *$P5 \mid\mid \Sigma w_j C_j$ can be solved in $\mathcal{O}(2.726^n)$ time.*

Proof. For every $S \subseteq [n]$, initialize $\texttt{DP}[2, S]$ using Lemma 11. Moreover, for every $S \subseteq [n]$ with $|S| \leqslant 0.6n$, initialize $\texttt{DP}[3, S]$ using Lemma 11. By Lemma 8 with $i = 5$, $j = 3$ and $S = [n]$ the following recurrence holds.

$$\texttt{DP}[5, [n]] = \min_{S \in \binom{[n]}{\leqslant 3n/5}} \{\texttt{DP}[3, S] + \texttt{DP}[2, [n] \setminus S]\} = \min_{S \in \binom{[n]}{\leqslant 0.6n}} \{\texttt{DP}[3, S] + \texttt{DP}[2, [n] \setminus S]\}.$$

With the values of $\texttt{DP}[2, S]$ for $S \subseteq [n]$ and $\texttt{DP}[3, S]$ for $S \in \binom{[n]}{\leqslant 0.6n}$ already available, compute the value of $\texttt{DP}[5, [n]]$ by enumerating all $S \in \binom{n}{\leqslant 0.6n}$ and taking the minimum value of $\texttt{DP}[3, S] + \texttt{DP}[2, [n] \setminus S]$. Finally, output this computed value of $\texttt{DP}[5, [n]]$.

The correctness of the algorithm follows from Lemmas 8 and 11. To analyze the running time, note that the initialization for $i = 2$ and all $S \subseteq [n]$ takes time at most

$$\sum_{S\subseteq[n]} \mathcal{O}^\star\left(2^{|S|/2}\right) = \mathcal{O}^\star\left(\sum_{r=0}^{n} \binom{n}{r} \cdot \left(\sqrt{2}\right)^r\right) = \mathcal{O}^\star\left(\left(1+\sqrt{2}\right)^n\right) \leqslant \mathcal{O}(2.4143^n).$$

The initialization for $i = 3$ and all $S \in \binom{[n]}{\leqslant 0.6n}$ takes time at most

$$\sum_{S\in\binom{[n]}{\leqslant 0.6n}} \mathcal{O}^\star\left(3^{|S|/2}\right) = \mathcal{O}^\star\left(\sum_{r=0}^{\lfloor 0.6n\rfloor} \binom{n}{r} \cdot \left(\sqrt{3}\right)^r\right) \leqslant \mathcal{O}(2.726^n).$$

The last inequality is due to Lemma 18. Finally, the computation of $\mathtt{DP}[5, [n]]$ using the above recurrence takes time $\mathcal{O}^\star(2^n)$. In total, the algorithm runs in time $\mathcal{O}(2.4143^n) + \mathcal{O}(2.726^n) + \mathcal{O}^\star(2^n) = \mathcal{O}(2.726^n)$. ◀

▶ **Lemma 17.** *$P6 \mid\mid \Sigma w_j C_j$ can be solved in $\mathcal{O}(2.733^n)$ time.*

Proof. For every $S \subseteq [n]$, initialize $\mathtt{DP}[3, S]$ using Lemma 11. By Lemma 8 with $i = 6$, $j = 3$ and $S = [n]$ the following recurrence holds.

$$\mathtt{DP}[6, [n]] = \min_{S\subseteq[n]} \{\mathtt{DP}[3, S] + \mathtt{DP}[3, [n] \setminus S]\}.$$

With the values of $\mathtt{DP}[3, S]$ for $S \subseteq [n]$ already available, compute the value of $\mathtt{DP}[6, [n]]$ by enumerating all $S \subseteq [n]$ and taking the minimum value of $\mathtt{DP}[3, S] + \mathtt{DP}[3, [n] \setminus S]$. Finally, output this computed value of $\mathtt{DP}[6, [n]]$.

The correctness of the algorithm follows from Lemmas 8 and 11. To analyze the running time, note that the initialization for $i = 3$ and all $S \subseteq [n]$ takes time at most

$$\sum_{S\subseteq[n]} \mathcal{O}^\star\left(3^{|S|/2}\right) = \mathcal{O}^\star\left(\sum_{r=0}^{n} \binom{n}{r} \cdot \left(\sqrt{3}\right)^r\right) = \mathcal{O}^\star\left(\left(1+\sqrt{3}\right)^n\right) \leqslant \mathcal{O}(2.733^n).$$

Finally, the computation of $\mathtt{DP}[6, [n]]$ using the above recurrence takes time $\mathcal{O}^\star(2^n)$. In total, the algorithm runs in time $\mathcal{O}(2.733^n) + \mathcal{O}^\star(2^n) = \mathcal{O}(2.733^n)$. ◀

B Technical Lemma

We prove the following technical lemma.

▶ **Lemma 18.** $\sum_{r=0}^{\lfloor 0.6n\rfloor} \binom{n}{r} \cdot \left(\sqrt{3}\right)^r \leqslant \mathcal{O}(2.726^n)$.

Proof. Consider the function $f(x) : (0, 0.6] \to \mathbb{R}$ defined as

$$f(x) = x(\ln\sqrt{3} - \ln x) - (1-x)\ln(1-x).$$

Therefore, for $r \in [\lfloor 0.6n\rfloor]$, we have

$$\begin{aligned}
\binom{n}{r} \cdot \left(\sqrt{3}\right)^r &\leqslant 2^{H(r/n)n} \cdot 2^{r\log\sqrt{3}} \\
&= 2^{((r/n)(\log\sqrt{3} - \log(r/n)) - (1-(r/n))\log(1-(r/n)))n} \\
&= \left(\exp\left((r/n)(\ln\sqrt{3} - \ln(r/n)) - (1-(r/n))\ln(1-(r/n))\right)\right)^n \\
&= \left(e^{f(r/n)}\right)^n.
\end{aligned}$$

Consider the first derivative $f'(x)$ of $f(x)$,

$$\begin{aligned} f'(x) &= \frac{d}{dx}\Big(x(\ln\sqrt{3} - \ln x) - (1-x)\ln(1-x)\Big) \\ &= (\ln\sqrt{3} - \ln x) + x\left(-\frac{1}{x}\right) - (-\ln(1-x)) - \left(-\frac{1-x}{1-x}\right) \\ &= \ln\sqrt{3} - \ln x + \ln(1-x). \end{aligned}$$

Now, consider the second derivative $f''(x)$ of $f(x)$,

$$f''(x) = \frac{d}{dx}\Big(\ln\sqrt{3} - \ln x + \ln(1-x)\Big) = -\frac{1}{x} - \frac{1}{1-x} < 0 \qquad \text{for } x \in (0, 0.6).$$

Thus $f'(x)$ is decreasing in the range $(0, 0.6]$. In particular, for all $x \in (0, 0.6)$, we have,

$$f'(x) > f'(0.6) = \ln\sqrt{3} - \ln 0.6 + \ln 0.4 = \ln\left(\frac{\sqrt{3}\cdot 0.4}{0.6}\right) = \ln\left(\frac{2\sqrt{3}}{3}\right) > 0.$$

This in turn implies $f(x)$ is increasing in the range $(0, 0.6]$. Plugging in the exact values gives us $f(0.6) \leqslant 1.0026$ and $e^{f(0.6)} \leqslant 2.7254$. Therefore, for all $r \in [\lfloor 0.6n \rfloor]$, we have,

$$\binom{n}{r} \cdot \left(\sqrt{3}\right)^r \leqslant \left(e^{f(r/n)}\right)^n \leqslant \left(e^{f(0.6)}\right)^n \leqslant 2.7254^n.$$

This implies,

$$\sum_{r=0}^{\lfloor 0.6n \rfloor} \binom{n}{r} \cdot \left(\sqrt{3}\right)^r = 1 + \sum_{r=1}^{\lfloor 0.6n \rfloor} \binom{n}{r} \left(\sqrt{3}\right)^r \leqslant 1 + \lfloor 0.6n \rfloor \cdot 2.7254^n = \mathcal{O}(2.726^n).$$

This completes the proof. ◀