On the Fine-Grained Parameterized Complexity of Partial Scheduling to Minimize the Makespan

Jesper Nederlof
Utrecht University, Algorithms and Complexity Group, The Netherlands
https://webspace.science.uu.nl/~neder003/
j.nederlof@uu.nl

Céline M. F. Swennenhuis
Eindhoven University of Technology, Combinatorial Optimization Group, The Netherlands
https://research.tue.nl/nl/persons/c%C3%A9line-swennenhuis
c.m.f.swennenhuis@tue.nl

─── **Abstract** ───

We study a natural variant of scheduling that we call *partial scheduling*: In this variant an instance of a scheduling problem along with an integer k is given and one seeks an optimal schedule where not all, but only k jobs, have to be processed.

Specifically, we aim to determine the fine-grained parameterized complexity of partial scheduling problems parameterized by k for all variants of scheduling problems that minimize the makespan and involve unit/arbitrary processing times, identical/unrelated parallel machines, release/due dates, and precedence constraints. That is, we investigate whether algorithms with runtimes of the type $f(k)n^{\mathcal{O}(1)}$ or $n^{\mathcal{O}(f(k))}$ exist for a function f that is as small as possible.

Our contribution is two-fold: First, we categorize each variant to be either in P, NP-complete and fixed-parameter tractable by k, or W[1]-hard parameterized by k. Second, for many interesting cases we further investigate the run time on a finer scale and obtain run times that are (almost) optimal assuming the Exponential Time Hypothesis. As one of our main technical contributions, we give an $\mathcal{O}(8^k k(|V| + |E|))$ time algorithm to solve instances of partial scheduling problems minimizing the makespan with unit length jobs, precedence constraints and release dates, where $G = (V, E)$ is the graph with precedence constraints.

1 Introduction

Scheduling is one of the most central application domains of combinatorial optimization. In the last decades, huge combined effort of many researchers led to major progress on understanding the worst-case computational complexity of almost all natural variants of scheduling: By now, for most of these variants it is known whether they are NP-complete or not. Scheduling problems provide the context of some of the most classic approximation algorithms. For example, in the standard textbook by Shmoys and Williamson on approximation algorithms [28] a wide variety of techniques are illustrated by applications to scheduling problems. See also the standard textbook on scheduling by Pinedo [23] for more background.

Instead of studying approximation algorithms, another natural way to deal with NP-completeness is *Parameterized Complexity* (PC). While the application of general PC theory to the area of scheduling has still received considerably less attention than the approximation point of view, recently its study has seen explosive growth, as witnessed by a plethora of publications (e.g. [2, 13, 16, 20, 26, 27]). Additionally, many recent results and open problems can be found in a survey by Mnich and van Bevern [19], and even an entire workshop on the subject was recently held [18].

In this paper we advance this vibrant research direction with a complete mapping of how several standard scheduling parameters influence the parameterized complexity of minimizing the makespan in a natural variant of scheduling problems that we call *partial scheduling*. Next to studying the classical question of whether parameterized problems are in P, FPT or W-hard, we also follow the well-established modern perspective of "fine-grained" PC and aim at run times of the type $f(k)n^{\mathcal{O}(1)}$ or $n^{f(k)}$ for the smallest function f of parameter k.

Partial Scheduling.   In many scheduling problems arising in practice, the set of jobs to be scheduled is not predetermined. We refer to this as *partial scheduling*. Partial scheduling is well-motivated from practice, as it arises naturally for example in the following scenarios:

1. Due to uncertainties a *close-horizon approach* may be employed and only few jobs out of a big set of jobs will be scheduled in a short but fixed time-window,
2. In freelance markets typically a large database of jobs is available and a freelancer is interested in selecting only a few of the jobs to work on,
3. The selection of the jobs to process may resemble other choices the scheduler should make, such as to outsource non-processed jobs to various external parties.

Partial scheduling has been previously studied in the equivalent forms of *maximum throughput scheduling* [24] (motivated by the first example setting above), *job rejection* [25], *scheduling with outliers* [12], *job selection* [8, 15, 29] and its special case *interval selection* [5].

In this paper, we conduct a rigorous study of the parameterized complexity of partial scheduling, parameterized by *the number of jobs to be scheduled*. We denote this number by k. While several isolated results concerning the parameterized complexity of partial scheduling do exist, this parameterization has (somewhat surprisingly) not been rigorously studied yet.[1] We address this and study the parameterized complexity of the (arguably) most natural variants of the problem. We fix as objective to minimize the makespan while scheduling at least k jobs, for a given integer k and study all variants with the following characteristics:

- 1 machine, identical parallel machines or unrelated parallel machines,
- release/due dates, unit/arbitrary processing times, and precedence constraints.

Note that a priori this amounts to $3 \times 2 \times 2 \times 2 \times 2 = 48$ variants.

1.1   Our Results

We give a classification of the parameterized complexity of these 48 variants. Additionally, for each variant that is not in P, we give algorithms solving them and lower bounds under ETH. To easily refer to a variant of the scheduling problem, we use the standard three-field notation by Graham et al. [11]. See Section 2 for an explanation of this notation. To accommodate our study of partial scheduling, we extend the $\alpha|\beta|\gamma$ notation as follows:

▶ **Definition 1.1.** *We let k-sched in the γ-field indicate that we only schedule k out of n jobs.*

---

[1] We compare the previous works and other relevant studied parameterization in the end of this section.

We study the fine-grained parameterized complexity of all problems $\alpha|\beta|\gamma$, where $\alpha \in \{1, P, R\}$, the options for β are all combinations for $r_j, prec, d_j, p_j = 1$, and γ is fixed to $\gamma = k\text{-sched}, C_{\max}$. Our results are explicitly enumerated in Table 1.

■ **Table 1** The fine-grained parameterized complexity of partial scheduling, where γ denotes k-sched, $C_{\max}$ and S.I. abbreviates SUBGRAPH ISOMORPHISM. Since $p_j = 1$ implies that the machines are identical, the mentioned number of 48 combinations reduces to 40 different scheduling problems. The $\mathcal{O}^*$ notation omits factors polynomial in the input size.

| | | Problem Description | Parameterized Complexity in k | Result Type | Lower Bound under ETH | | Run Time |
					Excluded Run Time	Reduction from	
	1	$1|\text{prec}, p_j = 1|\gamma$	P	**[A]**			$n^{\mathcal{O}(1)}$
	2	$1|r_j, \text{prec}, p_j = 1|\gamma$	P	**[A]**			$n^{\mathcal{O}(1)}$
	3	$1|d_j, \text{prec}, p_j = 1|\gamma$	W[1]-hard	**[B]**	$n^{o(k/\log k)}$	3-COLORING	$n^{\mathcal{O}(k)}$
	4	$1|r_j, d_j, \text{prec}, p_j = 1|\gamma$	W[1]-hard	**[B]**	$n^{o(k/\log k)}$	3-COLORING	$n^{\mathcal{O}(k)}$
	5	$P|\text{prec}, p_j = 1|\gamma$	FPT	**[C]**	$\mathcal{O}^*(2^{o(\sqrt{k\log k})})$	$P|\text{prec}, p_j = 1|C_{\max}$	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	6	$P|r_j, \text{prec}, p_j = 1|\gamma$	FPT	**[C]**	$\mathcal{O}^*(2^{o(\sqrt{k\log k})})$	$P|\text{prec}, p_j = 1|C_{\max}$	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
Precedence Relations	7	$P|d_j, \text{prec}, p_j = 1|\gamma$	W[1]-hard	**[B]**	$n^{o(k/\log k)}$	3-COLORING	$n^{\mathcal{O}(k)}$
	8	$P|r_j, d_j, \text{prec}, p_j = 1|\gamma$	W[1]-hard	**[B]**	$n^{o(k/\log k)}$	3-COLORING	$n^{\mathcal{O}(k)}$
	9	$1|\text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(\sqrt{k})}$	k-CLIQUE	$n^{\mathcal{O}(k)}$
	10	$1|r_j, \text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	PARTITIONED S.I.	$n^{\mathcal{O}(k)}$
	11	$1|d_j, \text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	PARTITIONED S.I.	$n^{\mathcal{O}(k)}$
	12	$1|r_j, d_j, \text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	PARTITIONED S.I.	$n^{\mathcal{O}(k)}$
	13	$P|\text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	PARTITIONED S.I.	$n^{\mathcal{O}(k)}$
	14	$P|r_j, \text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	PARTITIONED S.I.	$n^{\mathcal{O}(k)}$
	15	$P|d_j, \text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	PARTITIONED S.I.	$n^{\mathcal{O}(k)}$
	16	$P|r_j, d_j, \text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	PARTITIONED S.I.	$n^{\mathcal{O}(k)}$
	17	$R|\text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	PARTITIONED S.I.	$n^{\mathcal{O}(k)}$
	18	$R|r_j, \text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	PARTITIONED S.I.	$n^{\mathcal{O}(k)}$
	19	$R|d_j, \text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	PARTITIONED S.I.	$n^{\mathcal{O}(k)}$
	20	$R|r_j, d_j, \text{prec}|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	PARTITIONED S.I.	$n^{\mathcal{O}(k)}$
	21	$1|p_j = 1|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	22	$1|r_j, p_j = 1|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	23	$1|d_j, p_j = 1|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	24	$1|r_j, d_j, p_j = 1|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	25	$P|p_j = 1|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	26	$P|r_j, p_j = 1|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	27	$P|d_j, p_j = 1|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
No Precedence Relations	28	$P|r_j, d_j, p_j = 1|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	29	$1||\gamma$	P	**[F]**			$n^{\mathcal{O}(1)}$
	30	$1|r_j|\gamma$	P	**[F]**			$n^{\mathcal{O}(1)}$
	31	$1|d_j|\gamma$	P	**[F]**			$n^{\mathcal{O}(1)}$
	32	$1|r_j, d_j|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	SUBSET SUM	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	33	$P||\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	SUBSET SUM	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	34	$P|r_j|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	SUBSET SUM	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	35	$P|d_j|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	SUBSET SUM	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	36	$P|r_j, d_j|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	SUBSET SUM	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	37	$R||\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	SUBSET SUM	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	38	$R|r_j|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	SUBSET SUM	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	39	$R|d_j|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	SUBSET SUM	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	40	$R|r_j, d_j|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	SUBSET SUM	$\mathcal{O}^*(2^{\mathcal{O}(k)})$

The rows of Table 1 are lexicographically sorted on (i) precedence relations / no precedence relations, (ii) a single machine, identical machines or unrelated machines (iii) release dates and/or deadlines. Because their presence has a major influence on the character of the problem we stress the distinction between variants with and without *precedence constraints*.[2]

---

[2] A precedence constraint $a \prec b$ enforces that job a needs to be finished before job b can start.
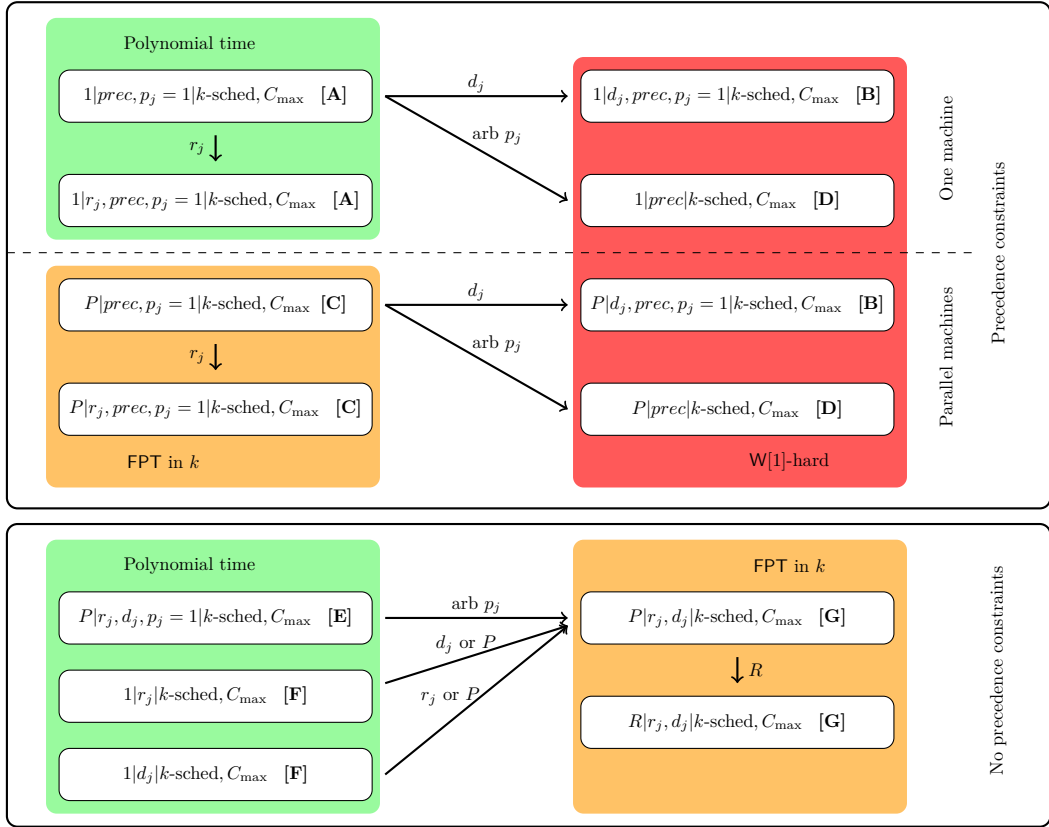
Figure 1 An illustration of the various result types as indicated in Table 1. Arrows indicate how a problem is generalized by another problem.

On a high abstraction level, our contribution is two-fold:

1. We present a *classification* of the complexity of all aforementioned variants of partial scheduling with the objective of minimizing the makespan. Specifically, we classify all variants to be either solvable in polynomial time, to be fixed-parameter tractable in k and NP-hard, or to be W[1]-hard.
2. For most of the studied variants we present both an algorithm and a lower bound that shows that our algorithm cannot be significantly improved unless the Exponential Time Hypothesis (ETH) fails.

Thus, while we completely answer a classical type of question in the field of Parameterized Complexity, we pursue in our second contribution a more modern and fine-grained understanding of the best possible run time with respect to the parameter k. For several of the studied variants, the lower bounds and algorithms listed in Table 1 follow relatively quickly. However, for many other cases we need substantial new insights to obtain (almost) matching upper and lower bounds on the runtime of the algorithms solving them. We have grouped the rows in *result types* [**A**]-[**G**] depending on our methods for determining their complexity.

1.2   Our new Methods

We now describe some of our most significant technical contributions for obtaining the various types (listed as [**A**]-[**G**] in Table 1) of results. Note that we skip some less interesting cases in this introduction; for a complete argumentation of all results from Table 1 we refer to

the full version of the paper. The main building blocks and logical implications to obtain the results from Table 1 are depicted in Figure 1. We now discuss these building blocks of Figure 1 in detail.

Precedence Constraints. Our main technical contribution concerns result type [**C**]. The simplest of the two cases, $P|\text{prec}, p_j = 1|k\text{-sched}, C_{\max}$, cannot be solved in $\mathcal{O}^*(2^{o(\sqrt{k \log k})})$ time assuming the Exponential Time Hypothesis and not in $2^{o(k)}$ unless sub-exponential time algorithms for the BICLIQUE problem exist, due to reductions by Jansen et al. [14]. Our contribution lies in the following theorem that gives an upper bound for the more general of the two problems that matches the latter lower bound:

▶ **Theorem 1.2.** $P|r_j, \text{prec}, p_j = 1|k\text{-sched}, C_{\max}$ *can be solved in* $\mathcal{O}(8^k k(|V| + |E|))$ *time,*[3] *where* $G = (V, E)$ *is the precedence graph given as input.*

Theorem 1.2 will be proved in Section 3. The first idea behind the proof is based on a natural[4] dynamic programming algorithm indexed by anti-chains of the partial order naturally associated with the precedence constraints. However, evaluating this dynamic program naïvely would lead to an $n^{\mathcal{O}(k)}$ time algorithm, where n is the number of jobs.

Our key idea is to only compute a subset of the table entries of this dynamic programming algorithm, guided by a new parameter of an antichain called the *depth*. Intuitively, the depth of an antichain A indicates the number of jobs that can be scheduled after A in a feasible schedule without violating the precedence constraints.

We prove Theorem 1.2 by showing we may restrict attention in the dynamic programming algorithm to antichains of depth at most k, and by bounding the number of antichains of depth at most k indirectly by bounding the number of *maximal* antichains of depth at most k. We believe this methodology should have more applications for scheduling problems with precedence constraints.

Surprisingly, the positive result of Theorem 1.2 is in *stark contrast* with the seemingly symmetric case where only deadlines are present: Our next result, indicated as [**B**] in Figure 1 shows it is much harder:

▶ **Theorem 1.3.** $P|d_j, \text{prec}, p_j = 1|k\text{-sched}, C_{\max}$ *is* W[1]*-hard, and cannot be solved in* $n^{o(k/\log k)}$ *time assuming the ETH.*

Theorem 1.3 is a consequence of a reduction outlined in Section 4. Note the W[1]-hardness follows from a natural reduction from the k-CLIQUE problem (presented originally by Fellows and McCartin [9]), but this reduction increases the parameter k to $\Omega(k^2)$ and would only exclude $n^{o(\sqrt{k})}$ time algorithms assuming the ETH. To obtain the tighter bound from Theorem 1.3, we instead provide a non-trivial reduction from the 3-COLORING problem based on a new selection gadget.

For result type [**D**], we give a lower bound by a (relatively simple) reduction from PARTITIONED SUBGRAPH ISOMORPHISM in Theorem 4.6 and Corollary 4.7. Since it is conjectured that PARTITIONED SUBGRAPH ISOMORPHISM cannot be solved in $n^{o(k)}$ time assuming the ETH, our reduction is a strong indication that the simple $n^{\mathcal{O}(k)}$ time algorithm (see [22]) cannot be improved significantly in this case.

---

[3] We assume basic arithmetic operations with the release dates take constant time.
[4] A similar dynamic programming approach was also present in for example [7].

No Precedence Constraints.   The second half of our classification concerns scheduling problems without precedence constraints, and is easier to obtain than the first half. Results [**E**], [**F**] are consequences of a greedy algorithm and Moore's algorithm [21] that solves the problem $1||\sum_j U_j$ in $\mathcal{O}(n \log n)$ time. Notice that this also solves the problem $1|r_j|k$-sched, $C_{\max}$, by reversing the schedule and viewing the release dates as the deadlines. For result type [**G**] we show that a standard technique in parameterized complexity, the color coding method, can be used to get a $2^{\mathcal{O}(k)}$ time algorithm for the most general problem of the class, being $R|r_j, d_j|k$-sched, $C_{\max}$. All lower bounds on the run time of algorithms for problems of type [**G**] are by a reduction from SUBSET SUM, but for $1|r_j, d_j|k$-sched, $C_{\max}$ this reduction is slightly different.

1.3   Related Work

The interest in parameterized complexity of scheduling problems recently witnessed an explosive growth, resulting in e.g. a workshop [18] and a survey by Mnich and van Bevern [19] with a wide variety of open problems.

The parameterized complexity of partial scheduling parameterized by the number of processed jobs, or equivalently, the number of jobs "on time" was studied before: Fellows et al. [9] studied a problem called k-TASKS ON TIME that is equivalent to $1|d_j, prec, p_j = 1|k$-sched, $C_{\max}$ and showed that it is W[1]-hard when parameterized by k,[5] and FPT parameterized by k and the width of the partially ordered set induced by the precedence constraints. Van Bevern et al. [27] showed that the JOB INTERVAL SELECTION problem, where each job is given a set of possible intervals to be processed on, is FPT in k. Bessy et al. [2] consider partial scheduling with a restriction on the jobs called "Coupled-Task", and also remarked the current parameterization is relatively understudied.

Another related parameter is the number of jobs that are *not scheduled*, that also has been studied in several previous works [4, 9, 20]. For example, Mnich and Wiese [20] studied the parameterized complexity of scheduling problems with respect to the number of rejected jobs in combination with other variables as parameter. If n denotes the number of given jobs, this parameter equals $n - k$. The two parameters are somewhat incomparable in terms of applications: In some settings only few jobs out of many alternatives need to be scheduled, but in other settings rejecting a job is very costly and thus will happen rarely. However, a strong advantage of using k as parameter is in terms of its computational complexity: If the version of the problem with all jobs mandatory is NP-complete it is trivially NP-complete for $n - k = 0$, but it may still be FPT in k.

1.4   Organization of this paper

This paper is organized as follows: We start with some preliminaries in Section 2. In Section 3 we present the proof of Theorem 1.2, and in Section 4 we describe the reductions for result types [**B**] and [**D**]. In Section 5 we give the algorithm for result type [**G**] and in Section 6 we present a conclusion. The proofs of some Theorems and Lemma's of Section 3 are omitted. These are indicated with a † and the full proofs can be found in the full version of the paper ([22]). In that version, we also motivate all cases from Table 1.

---

[5] Our results [**C**] and [**D**] build on and improve this result.

2 Preliminaries: The three-field notation by Graham et al.

Throughout this paper we denote scheduling problems using the three-field classification by Graham et al. [11]. Problems are classified by parameters $\alpha|\beta|\gamma$. The α describes the machine environment. This paper uses $\alpha \in \{1, P, R\}$, indicating whether there are one (1), identical (P) or unrelated (R) parallel machines available. Here identical refers to the fact that every job takes a fixed amount of time process independent of the machine, and unrelated means a job could take different time to process per machine. The β field describes the job characteristics, which in this paper can be a combination of the following values: *prec* (precedence constraints), r_j (release dates), d_j (deadlines) and $p_j = 1$ (all processing times are 1). We assume without loss of generality that all release dates and deadlines are integers.

The γ field concerns the optimization criteria. A given schedule determines C_j, the completion time of job j, and U_j, the unit penalty which is 1 if $C_j > d_j$, and 0 if $C_j \leq d_j$. In this paper we use the following optimization criteria

- $C_{\max}$: minimize the makespan (i.e. the maximum completion time C_j of any job),
- $\sum_j U_j$: minimize the number of jobs that finish after their deadline,
- k-sched: maximize the number of processed jobs; in particular, process at least k jobs.

A schedule is said to be *feasible* if no constraints (deadlines, release dates, precedence constraints) are violated.

3 Result Type C: Precedence Constraints, Release Dates and Unit Processing Times

In this section we provide a fast algorithm for partial scheduling with release dates and unit processing times parameterized by the number k of scheduled jobs (Theorem 1.2). There exists a simple, but slow, algorithm with runtime $\mathcal{O}^*(2^{k^2})$ that already proves that this problem is FPT in k: This algorithm branches k times on jobs that can be processed next. If more than k jobs are available at a step, then processing these jobs greedily is optimal. Otherwise, we can recursively try to schedule all non-empty subsets of jobs to schedule next, and a $\mathcal{O}^*(2^{k^2})$ time algorithm is obtained via a standard (bounded search-tree) analysis. To improve on this algorithm, we present a dynamic programming algorithm based on table entries indexed by antichains in the precedence graph G describing the precedence relations. Such an antichain describes the maximal jobs already scheduled in a partial schedule. Our key idea is that, to find an optimal solution, it is sufficient to restrict our attention to a subset of all antichains. This subset will be defined in terms of the *depth* of an antichain. With this algorithm we improve the runtime to $\mathcal{O}(8^k k(|V| + |E|))$.

By binary search, we can restrict attention to a variant of the problem that asks whether there is a feasible schedule with makespan at most $C_{\max}$, for a fixed universal deadline $C_{\max}$.

Notation for Posets. Any precedence graph G is a directed acyclic graph and therefore induces a partial order $\prec$ on $V(G)$. Indeed, if there is a path from x to y, we let $x \preceq y$. An *antichain* is a set $A \subseteq V(G)$ of mutually incomparable elements. We say A is *maximal* if there is no antichain A' with $A \subset A'$. The set of *predecessors* of A is $\mathrm{pred}(A) = \{x \in V(G) : \exists a \in A : x \preceq a\}$, and the the set of *comparables* of A is $\mathrm{comp}(A) = \{x \in V(G) : \exists a \in A : x \preceq a \text{ or } x \succeq a\}$. Note $\mathrm{comp}(A) = V(G)$ if and only if A is maximal.

An element $x \in V(G)$ is a *minimal* element if $x \preceq y$ for all $y \in \mathrm{comp}(\{x\})$. An element $x \in V(G)$ is a *maximal* element if $x \succeq y$ for all $y \in \mathrm{comp}(\{x\})$. Furthermore $\min(G) = \{x \mid x \text{ is a minimal element in } G\}$ and $\max(G) = \{x \mid x \text{ is a maximal element in } G\}$.

Notice that $\max(G)$ is exactly the antichain A such that $\mathrm{pred}(A) = V(G)$. We denote the subgraph of G induced by S with $G[S]$. We may assume that $r_j < r_{j'}$ if $j \prec j'$ since job j' will be processed later than r_j in any schedule. To handle release dates we use the following:

▶ **Definition 3.1.** *Let G be a precedence graph. Then G^t is the precedence graph restricted to all jobs that can be scheduled on or before time t, i.e. all jobs with release date at most t.*

We assume $G = G^{C_{\max}}$, since all jobs with release date greater than $C_{\max}$ can be ignored.

The Algorithm.   We now introduce our dynamic programming algorithm for $P|r_j, prec, p_j = 1|k\text{-sched}, C_{\max}$. Let m be the number of machines available. We start with defining the table entries. For a given antichain $A \subseteq V(G)$ and integer t we define

$$S(A, t) = \begin{cases} 1, & \text{if there exists a feasible schedule of makespan } t \text{ that processes } \mathrm{pred}(A), \\ 0, & \text{otherwise.} \end{cases}$$

Computing the values of $S(A, t)$ can be done by trying all combinations of scheduling at most m jobs of A at time t and then checking whether all remaining jobs of $\mathrm{pred}(A)$ can be scheduled in makespan $t - 1$. To do so, we also verify that all the jobs in A actually have a release date at or before t. Formally, we have the following recurrence for $S(A, t)$:

▶ **Lemma 3.2.**

$$S(A, t) = (A \subseteq V(G^t)) \wedge \bigvee_{X \subseteq A : |X| \leq m} S(A', t - 1) : A' = \max(\mathrm{pred}(A) \setminus X).$$

Proof. If $A \not\subseteq V(G^t)$, then there is a job $j \in A$ with $r_j > t$. And thus $S(A, t) = 0$.

For any $X \subseteq A$, X is a set of maximal elements with respect to $G[\mathrm{pred}(A)]$, and consists of pair-wise incomparable jobs, since A is an antichain. So, we can schedule all jobs from X at time t without violating any precedence constraints. Define $A' = \max(\mathrm{pred}(A) \setminus X)$ as the unique antichain such that $\mathrm{pred}(A) \setminus X = \mathrm{pred}(A')$. If $S(A', t - 1) = 1$ and $|X| \leq m$, we can extend the schedule of $S(A', t - 1)$ by scheduling all X at time t. In this way we get a feasible schedule processing all jobs of $\mathrm{pred}(A)$ before or at time t. So if we find such an X with $|X| \leq m$ and $S(A', t - 1) = 1$, we must have $S(A, t) = 1$.

For the other direction, if for all $X \subseteq A$ with $|X| \leq m$, $S(A', t - 1) = 0$, then no matter which set $X \subseteq A$ we try to schedule at time t, the remaining jobs cannot be scheduled before t. Note that only jobs from A can be scheduled at time t, since those are the maximal jobs. Hence, there is no feasible schedule and $S(A, t) = 0$. ◀

The above recurrence cannot be directly evaluated, since the number of different antichains of a graph can be big: there can be as many as $\binom{n}{k}$ different antichains with $|\mathrm{pred}(A)| \leq k$, for example in the extreme case of an independent set. Even when we restrict our precedence graph to have out degree k, there could be k^k different antichains, for example in k-ary trees. To circumvent this issue, we restrict our dynamic programming algorithm only to a specific subset of antichains. To do this, we use the following new notion of the *depth* of an antichain.

▶ **Definition 3.3.** *Let A be an antichain. Define the depth (with respect to t) of A as*

$$d^t(A) = |\mathrm{pred}(A)| + |\min(G^t - \mathrm{comp}(A))|.$$

We also denote $d(A) = d^{C_{\max}}(A)$.

$$d(A) = |\mathrm{pred}(A)| + |\min(G - \mathrm{comp}(A))| = 2 + 2$$
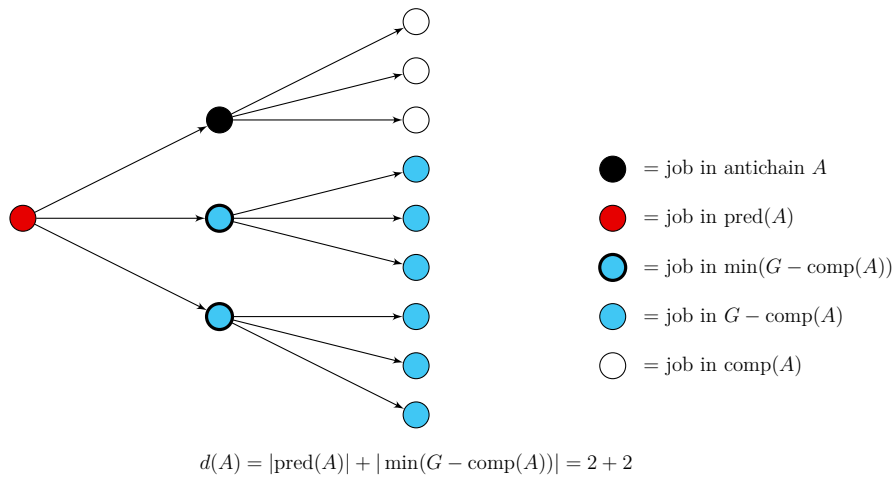
Figure 2 Example of an antichain and its depth in a perfect 3-ary tree. We see that $|\mathrm{pred}(A)| = 2$, but $d(A) = 4$. If $k = 2$, the dynamic programming algorithm will not compute $S(A, t)$ since $d(A) > k$. The only antichains with depth ≤ 2 are the empty set and the root node r on its own as a set. Indeed $d(\emptyset) = d(\{r\}) = 1$. Note that for instances with $k = 2$, a feasible schedule may exist. If so, we will find that $R(\{r\}, 1) = 1$, which will be defined later. In this way, we can still find the antichain A as a solution.

The intuition behind this definition is that it quantifies the number of jobs that can be scheduled before (and including) A without violating precedence constraints. See Figure 2 for an example of an antichain and its depth. We restrict the dynamic programming algorithm to only compute $S(A, t)$ for A satisfying $d^t(A) \leq k$. This ensures that we do not go "too deep" into the precedence graph unnecessarily at the cost of a slow runtime.

Because of this restriction in the depth, it could happen that we check no antichains with k or more predecessors, while there are corresponding feasible schedules. It is therefore possible that for some antichains A with $d^t(A) > k$, there is a feasible schedule for all $\geq k$ jobs in $\mathrm{pred}(A)$ before time $C_{\max}$, but the value $S(A, C_{\max})$ will not be computed. To make sure we still find an optimal schedule, we also compute the following condition $R(A, t)$ for all $t \leq C_{\max}$ and antichains A with $d^t(A) \leq k$:

$$R(A, t) = \begin{cases} 1, & \text{if there exists a feasible schedule with makespan at most } C_{\max} \text{ that} \\ & \text{processes } \mathrm{pred}(A) \text{ on or before } t \text{ and processes jobs from} \\ & \min(G - \mathrm{pred}(A)) \text{ after } t, \text{ with a total of } k \text{ jobs processed,} \\ 0, & \text{otherwise.} \end{cases}$$

By definition of $R(A, t)$, if $R(A, t) = 1$ for any A and $t \leq C_{\max}$, then we find a feasible schedule that processes k jobs on time.[6] We show in [22] that $R(A, t)$ can be quickly computed:

▶ **Lemma 3.4** (†). *There is an $\mathcal{O}(|V|k + |E|)$ time algorithm* `fill(A,t)` *that, given an antichain A, integer t, and value $S(A, t)$, computes $R(A, t)$.*

The algorithm `fill(A,t)` checks if $S(A, t) = 1$ and if so, greedily schedules jobs from $\min(G - \mathrm{pred}(A))$ after t in order of smallest release date. If $k - |\mathrm{pred}(A)|$ jobs can be scheduled before $C_{\max}$, it returns "true" ($R(A, t) = 1$). Otherwise, it returns "false" ($R(A, t) = 0$).

---

[6] The reverse direction is more difficult and postponed to Lemma 3.6.

Combining all steps gives us the algorithm as described in Algorithm 1. It remains to bound its runtime and argue its correctness.

■ **Algorithm 1** Algorithm for $P|\text{pred}, p_j = 1|k\text{-sched}, C_{\max}$.

---
1 foreach $t = 1, ..., C_{\max}$ **do**
2 ⎢ Enumerate all antichains A in G^t with $d^t(A) \leq k$ using Lemma 3.5
3 ⎢ **foreach** *antichain A in G^t with $d^t(A) \leq k$* **do**
4 ⎢ ⎢ Compute $S(A,t)$ using Lemma 3.2
5 ⎢ ⎢ **if** `fill`$(S(A,t), A, t)$ **then**
6 ⎢ ⎢ ⎢ **return** *TRUE*
7 return *FALSE*

Runtime.   To analyze the runtime of the dynamic programming algorithm, we need to bound the number of checked antichains. Recall that we only check antichains A with $d^t(A) \leq k$ for each time $t \leq C_{\max}$. We first analyze the number of antichains A with $d(A) \leq k$ in any graph and use this to upper bound the number of antichains checked at time t.

▶ **Lemma 3.5** (†). *For any t, there are at most 4^k antichains A with $d^t(A) \leq k$ in any precedence graph $G = (V, E)$, and they can be enumerated within $\mathcal{O}(4^k(|V| + |E|))$ time.*

Notice that to compute each $S(A, t)$, we look at a maximum of $\binom{k}{m} \leq 2^k$ different sets X. Computing the antichain A' such that $A' = \max(\text{pred}(A) \setminus X)$ takes $\mathcal{O}(|V| + |E|)$ time. After this computation, $R(A, t)$ is directly computed in $\mathcal{O}(|V|k + |E|)$ time. For each time $t \in \{1, ..., C_{\max}\}$, there are at most 4^k different antichains A for which we compute $S(A, t)$ and $R(A, t)$. Since $C_{\max} \leq k$, we therefore have total runtime of $\mathcal{O}(4^k k(2^k(|V| + |E|) + (|V|k + |E|)))$. Hence, Algorithm 1 runs in time $\mathcal{O}(8^k k(|V| + |E|))$.

Correctness of algorithm.   To show that the algorithm described in Algorithm 1 indeed returns the correct answer, the following lemma is clearly sufficient:

▶ **Lemma 3.6** (†). *A feasible schedule for k jobs with makespan at most $C_{\max}$ exists if and only if $R(A, t) = 1$ for some $t \leq C_{\max}$ and antichain A with $d^t(A) \leq k$.*

To prove Lemma 3.6, we consider the schedule which corresponds to an antichain which has minimal depth. We then conclude that it either should be witnessed by some $R(A, t)$ or that there is another antichain with even smaller depth, which contradicts the assumption. The proof heavily relies on the intricacies of the definition of depth.

4   Result Types B and D: One Machine and Precedence Constraints

In this section we show that Algorithm 1 cannot be even slightly generalized further: if we allow job-dependent deadlines or non-unit processing times, the problem becomes W[1]-hard parameterized by k and cannot be solved in $n^{o(k/\log k)}$ time unless the ETH fails.

Job-dependent deadlines.   The fact that combining precedence constraints with job-dependent deadlines makes the problem W[1]-hard, is a direct consequence from the fact that $1|prec, p_j = 1| \sum_j U_j$ is W[1]-hard, parameterized by $n - \sum_j U_j = k$ where n is the number of jobs [9]. It is important to notice that the notation of these problems implies that each job

can have its own deadline. Hence, we conclude from this that $1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$ is W[1]-hard parameterized by k. This is a reduction from k-CLIQUE and therefore we get a lower bound on algorithms for the problem of $n^{\Omega(\sqrt{k})}$. Based on the Exponential Time Hypothesis, we now sharpen this lower bound with a reduction from 3-COLORING:

▶ **Theorem 4.1.** $1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$ *is* W[1]-*hard parameterized by* k. *Furthermore, there is no algorithm solving* $1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$ *in* $2^{o(n)}$ *time where* n *is the number of jobs, assuming ETH.*

Proof. The proof will be a reduction from 3-COLORING, for which no $2^{o(|V|+|E|)}$ algorithm exists under the Exponential Time Hypothesis [6, pages 471-473]. Let the graph $G = (V, E)$ be the instance of 3-COLORING with $|V| = n'$ and $|E| = m'$. We then create the following instance for $1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$.

- For each vertex $v_i \in V$, create 6 jobs:
  - v_i^1, v_i^2 and v_i^3 with deadline $d_{v_i} = i$,
  - w_i^1, w_i^2 and w_i^3 with deadline $d_{w_i} = n' + 2m' + 1 - i$,
  add precedence constraints $v_i^1 \prec w_i^1$, $v_i^2 \prec w_i^2$ and $v_i^3 \prec w_i^3$. These jobs represent which color for each vertex will be chosen (if v_i^1 and w_i^1 are processed, vertex i gets color 1).
- For each edge $e_j \in E$, create 12 jobs:
  - e_j^{12}, e_j^{13}, e_j^{21}, e_j^{23}, e_j^{31} and e_j^{32} with deadline $d_{e_j} = n' + j$,
  - f_j^{12}, f_j^{13}, f_j^{21}, f_j^{23}, f_j^{31} and f_j^{32} with deadline $d_{f_j} = n' + m' + 1 - j$,
  add precedence constraints $e_j^{ab} \prec f_j^{ab}$. These jobs represent what the colors of the endpoints of an edge will be. So if the jobs e_j^{ab} and f_j^{ab} are processed for $e = \{u, v\}$, then vertex u has color a and vertex v has color b. Since the endpoints should have different colors, the jobs e_j^{aa} and f_j^{aa} do not exist.
- For each e_j^{ab} with $e = \{u, v\}$ add the precedence constraints $u^a \prec e_j^{ab}$ and $v^b \prec e_j^{ab}$.
- Set $C_{\max} = k = 2n' + 2m'$.

We now prove that the created instance is a yes instance if and only if the original 3-COLORING instance is a yes instance. Assume that there is a 3-coloring of the graph $G = (V, E)$. Then there is also a feasible schedule: For each vertex v_i with color a, process the jobs v_i^a and w_i^a at their respective deadlines. For each edge $e_j = \{u, v\}$ with u colored a and v colored b, process the jobs e_j^{ab} and f_j^{ab} exactly at their respective deadlines. Notice that because it is a 3-coloring, each edge has endpoints of different colors, so these jobs exist. Also note that no two jobs were processed at the same time. Exactly $2n' + 2m'$ jobs were processed before time $2n' + 2m'$. Furthermore, no precedence constraints were violated.

For the other direction, assume that we have a feasible schedule in our created instance of $1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$. Let $\mathcal{V}_i = \{v_i^1, v_i^2, v_i^3\}$, $\mathcal{W}_i = \{w_i^1, w_i^2, w_i^3\}$, and let $\mathcal{E}_j = \{e_j^{12}, e_j^{13}, e_j^{21}, e_j^{23}, e_j^{31}, e_j^{32}\}$ and $\mathcal{F}_j = \{f_j^{12}, f_j^{13}, f_j^{21}, f_j^{23}, f_j^{31}, f_j^{32}\}$. We show by induction on i that out of each of the sets $\mathcal{V}_i$, $\mathcal{W}_i$, $\mathcal{E}_j$ and $\mathcal{F}_j$, exactly one job was scheduled at its deadline.

Since we have a feasible schedule, at time $2m' + 2n'$ one of the jobs of $\mathcal{W}_1$ must be scheduled, since they are the only jobs with a deadline greater than $2n + 2m - 1$. However, if w_1^a was scheduled at time $2m' + 2n'$, then the job v_1^a must be processed at time 1 because of precedence constraints and since its deadline is 1. Note, that no other jobs from $\mathcal{V}_1$ and $\mathcal{W}_1$ can be processed, due to their deadlines and precedence constraints.

Now assume that all sets $\mathcal{V}_1, ..., \mathcal{V}_{i-1}, \mathcal{W}_1, ..., \mathcal{W}_{i-1}$ have exactly one job scheduled at their respective deadline, and no more can be processed. Since we have a feasible schedule, one job should be scheduled at time $2n' + 2m' - (i - 1)$. However, since no more jobs from $\mathcal{W}_1, ..., \mathcal{W}_{i-1}$ can be scheduled, the only possible jobs are from $\mathcal{W}_i$ since they are the only other jobs with a deadline greater than $2n' + 2m' - i$. However, if w_i^a was scheduled at

time $2n' + 2m' - (i - 1)$, then the job v_i^a must be processed at time i because of precedence constraints, its deadline at i and because at times $1, ..., i - 1$ other jobs had to be processed. Also, no other job from $\mathcal{V}_i$ can be processed in the schedule, since they all have deadline i. As a consequence, no other jobs from $\mathcal{W}_1$ can be processed, as they are restricted to precedence constraints. So the statement holds for all set $\mathcal{V}_i$ and $\mathcal{W}_i$. In the exact same way, one can conclude the same about all sets $\mathcal{E}_j$ and $\mathcal{F}_j$.

Because of this, we see that each job and each vertex have received a color from the schedule. They must form a 3-coloring, because a job from $\mathcal{E}_j$ could only be processed if the two endpoints got two different colors. Hence the 3-COLORING instance is a yes instance.

As $k = 2n' + 2m'$ we therefore conclude there is no $2^{o(n)}$ algorithm under the ETH.   ◄

Note that this bound significantly improves the old lower bound of $2^{\Omega(\sqrt{n})}$ implied by the the reduction from k-CLIQUE reduction: Since $k \leq n$, Theorem 4.1 implies that

▶ **Corollary 4.2.** *Assuming ETH, there is no algorithm solving* $1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$ *in* $n^{o(k/\log(k))}$ *where* n *is the number of jobs.*

Non-unit processing times.   We show that having non-unit processing times combined with precedence constraints make the problem $\mathsf{W}[1]$-hard even on one machine. The proof of Theorem 4.3 heavily builds on the reduction from k-CLIQUE to k-TASKS ON TIME by Fellows and McCartin [9].

▶ **Theorem 4.3.** $1|prec|k\text{-sched}, C_{\max}$ *is* $\mathsf{W}[1]$*-hard, parameterized by* k.

Proof. The proof is a reduction from k-CLIQUE. We start with $G = (V, E)$, an instance of k-CLIQUE. For each vertex $v \in V$, create a job j_v with $p_{j_v} = 2$. For each edge $e \in E$, create a job j_e with $p_{j_e} = 1$. Now for each edge (u, v), add the following two precedence relations: $j_u \prec j_e$ and $j_v \prec j_e$, so before one can process a job associated with an edge, both jobs associated with the endpoints of that edge need to be finished. Now let $k' = k + \frac{1}{2}k(k - 1)$ and $C_{\max} = 2k + \frac{1}{2}k(k - 1)$. We will now prove that $1|prec|k'\text{-sched}, C_{\max}$ is a yes instance if and only of k-CLIQUE is a yes instance.

Assume that the k-CLIQUE instance is a yes instance, then process first the k jobs associated with the vertices of the k-clique. Next process the $\frac{1}{2}k(k - 1)$ jobs associated with the edges of the k-clique. In total, $k + \frac{1}{2}k(k - 1) = k'$ jobs are now processed with a makespan of $2k + \frac{1}{2}k(k - 1)$. Hence, the instance of $1|prec|k'\text{-sched}, C_{\max}$ is a yes instance.

For the other direction, assume $1|prec|k'\text{-sched}, C_{\max}$ to be a yes instance, so we have found a feasible schedule. For any feasible schedule, if one schedules l jobs associated with vertices, then at most $\frac{1}{2}l(l - 1)$ jobs associated with edges can be processed, because of the precedence constraints. However, because $k' = k + \frac{1}{2}k(k - 1)$ jobs were done in the feasible schedule before $C_{\max} = 2k + \frac{1}{2}k(k - 1)$, at most k jobs associated with vertices can be processed, because they have processing time of size 2. Hence, we can conclude that exactly k vertex-jobs and $\frac{1}{2}k(k - 1)$ edge-jobs were processed. Hence, there were k vertices connected through $\frac{1}{2}k(k - 1)$ edges, which is a k-clique.   ◄

The proofs of Theorem 4.6 and Corollary 4.7 are reductions from PARTITIONED SUBGRAPH ISOMORPHISM. Let $P = (V', E')$ be a "pattern" graph, $G = (V, E)$ be a "target" graph, and $\chi : V \to V'$ a "coloring" of the vertices of G with elements from P. A χ-*colorful P-subgraph of* G is a mapping $\varphi : V' \to V$ such that (1) for each $\{u, v\} \in E'$ it holds that $\{\varphi(u), \varphi(v)\} \in E$ and (2) for each $u \in V'$ it holds that $\chi(\varphi(u)) = u$. If χ and G are clear from the context they may be omitted in this definition.

▶ **Definition 4.4** (PARTITIONED SUBGRAPH ISOMORPHISM). *Given graphs $G = (V, E)$ and $P = (V', E')$, $\chi : V \to V'$. Determine whether there is a χ-colorful P-subgraph of G.*

▶ **Theorem 4.5** (Marx [17]). PARTITIONED SUBGRAPH ISOMORPHISM *cannot be solved in $n^{o(|E'|/\log |E'|)}$ time assuming the Exponential Time Hypothesis (ETH).*

We will now reduce PARTITIONED SUBGRAPH ISOMORPHISM to $1|\text{prec}, r_j|k\text{-sched}, C_{\max}$.

▶ **Theorem 4.6.** $1|prec, r_j|k\text{-sched}, C_{\max}$ *cannot be solved in $n^{o(k/\log k)}$ time assuming the Exponential Time Hypothesis (ETH).*

Proof. Let $G = (V, E)$, $P = (V', E')$ and $\chi : V \to V'$. We will write $V' = \{1, \ldots, s\}$. Define for $i = 0, \ldots, s$ the following important time stamps:

$$t_i := \sum_{j=1}^{i} 3^{s+1-j}.$$

Construct the following jobs for the instance of the $1|\text{prec}, r_j|k\text{-sched}, C_{\max}$ problem:
- For $i = 1, \ldots, s$:
  - For each vertex $v \in V$ such that $\chi(v) = i$, create a job j_v with processing time $p(j_v) = 3^{s+1-i}$ and release date t_{i-1}.
- For each $(v, w) \in E$ such that $(\chi(v), \chi(w)) \in E'$, create a job $j_{v,w}$ with $p(j_{v,w}) = 1$ and release date t_s. Add precedence constraints $j_v \prec j_{v,w}$ and $j_w \prec j_{v,w}$.

Then ask whether there exists a solution to the scheduling problem for $k = s + |E'|$ with makespan $C_{\max} \leq t_s + |E'|$.

Let the PARTITIONED SUBGRAPH ISOMORPHISM instance be a yes-instance and let $\varphi : V(P) \to V(G)$ be a colorful P-subgraph. We claim the following schedule is feasible:

- For $i = 1, \ldots, s$:
  - Process $j_{\varphi(i)}$ at its release date t_{i-1}.
- Process for each $(i, i') \in E'$ the job $j_{\varphi(i), \varphi(i')}$ somewhere in the interval $[t_s, t_s + |E'|]$.

Notice that all jobs are indeed processed after their release date and that in total there are $k = s + |E'|$ processed before $C_{\max} \leq t_s + |E'|$. Furthermore, all precedence constraints are respected as any edge job is processed after both its predecessors. Also, the edge jobs $e^{\varphi(i), \varphi(i')}$ must exist, as $\varphi(P)$ is a properly colored P-subgraph. Therefore, we can conclude that indeed this schedule is feasible.

For the other direction, assume that there is a solution to the created instance of $1|\text{prec}, r_j|k\text{-sched}, C_{\max}$. Define $J_i = \{j_v : \chi(v) = i\}$. We will first prove that at most 1 job from each set J_i can be processed in a feasible schedule. To do this, we first prove that at most 1 job from each set J_i can be processed before t_s. Any job in J_i has release date $t_{i-1} = \sum_{j=1}^{i-1} 3^{s+1-j}$. Therefore, there is only $t_s - t_{i-1} = \sum_{j=i}^{s} 3^{s+1-j}$ time left to process the jobs from J_i before time t_s. However, the processing time of any job in J_i is 3^{s+1-i}, and since $2 \cdot 3^{s+1-i} > \sum_{j=i}^{s} 3^{s+1-j}$, at most 1 job from J_i can be processed before t_s. Since all jobs not in some J_i have their release date at t_s, at most s jobs are processed at time t_s. Thus at time t_s, there are $|E'|$ time unit left to process $|E'|$ jobs, because of the choice of k and makespan. Hence the only way to get a feasible schedule is to process exactly one job from each set J_i at its respective release date and process exactly $|E'|$ edge jobs after t_s.

Let v^i be the vertex, such that j_v was processed in the feasible schedule with color i. We will show that $\varphi : V(P) \to V(G)$, defined as $\varphi(i) = v^i$, is a function such that $\varphi(P)$ is a properly colored P-subgraph of G. Hence, we are left to prove that for each $(i, i') \in E'$,

the edge $(\varphi(i), \varphi(i')) \in E$, i.e. that for each $(i, i') \in E'$, the job $j_{\varphi(i),\varphi(i')}$ was processed. Because only the vertex jobs $j_{\varphi(1)}, j_{\varphi(2)}, \ldots, j_{\varphi(s)}$ were processed, the precedence constraints only allow for edge jobs $j_{\varphi(i),\varphi(i')}$ to be processed. We created edge job $j_{v,w}$ if and only if $(v, w) \in E$ and $(\chi(v), \chi(w)) \in E'$, hence the $|E'|$ edge jobs have to be exactly the edge jobs $j_{\varphi(i),\varphi(i')}$ for $(i, i') \in E'$. Therefore, we proved indeed that $\varphi(P)$ is a colorful P-subgraph of G.

Notice that $k = s + |E'| \leq 3|E'|$ as we may assume the number of vertices in P is at most $2|E'|$. Hence the given bound follows. ◄

▶ **Corollary 4.7.** $2|prec|k\text{-}sched, C_{\max}$ *cannot be solved in $n^{o(k/\log k)}$ time assuming the Exponential Time Hypothesis (ETH).*

Proof. We can use the same idea for the reduction from PARTITIONED SUBGRAPH ISO-MORPHISM as in the proof of Theorem 4.6, except for the release dates, as they are not allowed in this type of scheduling problem. To simulate the release dates, we use the second machine as a release date machine, meaning that we will create a job for each upcoming release date and will require these new jobs to be processed. More formally: For $i = 1, \ldots, s$, create a job j_{r_i} with processing time 3^{s+1-i} and precedence constraints $j_{r_i} \prec j$ for any job j that had release date t_i in the original reduction. Furthermore let $j_{r_i} \prec j_{r_{i+1}}$. Then we add $|E'|$ jobs j' with processing time 1 and with precedence relations $j_{r_s} \prec j'$. We then ask whether there exists a feasible schedule with $k = 2s + 2|E'|$ and with makespan $t_s + |E'|$. All newly added jobs are required in any feasible schedule and therefore, all other arguments from the previous reduction also hold. Finally, note that k is again linear in $|E'|$. ◄

5   Result Type G: k-scheduling without Precedence Constraints

The problem $P|k\text{-}sched|C_{\max}$, cannot be solved in $2^{o(k)}$ time assuming the ETH by a reduction to SUBSET SUM. We show that the problem is fixed-parameter tractable with a matching run time in k, even in the case of unrelated machines, release dates and deadlines, denoted by $R|r_j, d_j, k\text{-}sched|C_{\max}$.

▶ **Theorem 5.1.** $R|r_j, d_j, k\text{-}sched|C_{\max}$ *is fixed-parameter tractable in k and can be solved in $\mathcal{O}^*((2e)^k k^{\mathcal{O}(\log k)})$ time.*

Proof. We give an algorithm that solves any instance of $R|r_j, d_j, k\text{-}sched|C_{\max}$ within $\mathcal{O}^*((2e)^k k^{\mathcal{O}(\log k)})$ time. The algorithm is a randomized algorithm that can be de-randomized using the color coding method, as described by Alon et al. [1]. The algorithm first (randomly) picks a coloring $c : \{1, ..., n\} \to \{1, ..., k\}$, so each job is given one of the k available colors. We then compute whether there is a feasible colorful schedule, i.e. a feasible schedule that processes exactly one job of each color. If this colorful schedule can be found, then it is possible to schedule at least k jobs before $C_{\max}$.

Given a coloring c, we compute whether there exists a colorful schedule in the following way. Define for $1 \leq i \leq m$ and $X \subseteq \{1, ..., k\}$:

$B_i(X) =$ minimum makespan of all schedules on machine i processing $|X|$ jobs,
each from a different color in X.

Clearly $B_i(\emptyset) = 0$, and all values $B_i(X)$ can be computed in $\mathcal{O}(2^k n)$ time using the following:

▶ **Lemma 5.2.** *Let* $\min\{\emptyset\} = \infty$. *Then*

$$B_i(X) = \min_{l \in X} \min_{j:c(j)=l} \{C_j = \max\{r_j, B_i(X \setminus \{l\})\} + p_{ij} : C_j \leq d_j\}.$$

Proof. In a schedule on one machine with $|X|$ jobs using all colors from X, one job should be scheduled as last, defining the makespan. So for all possible jobs j, we compute what the minimal end time would be if j was scheduled at the end of the schedule. This j cannot start before its release date or before all other colors are scheduled. ◄

Next, define for $1 \leq i \leq m$ and $X \subseteq [k]$, $A_i(X)$ to be 1 if $B_i(X) \leq C_{\max}$, and to be 0 otherwise. So $A_i(X) = 1$ if and only if $|X|$ jobs, each from a different color of X, can be scheduled on machine i before $C_{\max}$. A colorful feasible schedule exists if and only if there is some partition $X_1, ..., X_m$ of $\{1, .., k\}$ such that $\Pi_{i=1}^m A_i(X_i) = 1$. The *subset convolution* of two functions is defined as $(A_i * A_{i'})(X) = \sum_{Y \subseteq X} A_i(Y) A_{i'}(X \setminus Y)$. Then $\Pi_{i=1}^m A_i(X_i) = 1$ if and only if $(A_1 * \cdots * A_m)(\{1, ..., k\}) > 0$. The value of $(A_1 * \cdots * A_m)(\{1, ..., k\}) > 0$ can be computed in $2^k k^{\mathcal{O}(1)}$ time using fast subset convolution [3].

An overview of the randomized algorithm is given in Algorithm 2. If the k jobs that are processed in an optimal solution are all in different colors, the algorithm outputs true. By standard analysis, k jobs are all assigned different colors with probability at least $1/e^k$, and thus e^k independent trials to boost the error probability of the algorithm to at most $1/2$.

■ **Algorithm 2** Algorithm for solving $R|r_j, d_j, k\text{-sched}|C_{\max}$.

---

1 For a given coloring c:
2 **foreach** $i = 1, ..., m$ **do**
3 $\quad$ **foreach** $X \subseteq \{1, .., k\}$ *in order of increasing size* **do**
4 $\quad\quad$ Compute $B_i(X)$ using Lemma 5.2.
5 $\quad\quad$ Set $A_i(X) = 1$ if $B_i(X) \leq C_{\max}$, set $A_i(X) = 0$ otherwise.
6 Compute $(A_1 * \cdots * A_m)(\{1, ..., k\})$ using fast subset convolution [3].
7 **if** $(A_1 * \cdots * A_m)(\{1, ..., k\}) > 0$ **then**
8 $\quad$ **return** *TRUE*

By using the standard methods by Alon et al. [1], Algorithm 2 can be derandomized. ◄

6 Concluding Remarks

We classify all studied variants of partial scheduling parameterized by the number of jobs to be scheduled to be either in P, NP-complete and fixed-parameter tractable by k, or W[1]-hard parameterized by k. Our main technical contribution is an $\mathcal{O}(8^k k(|V| + |E|))$ time algorithm for $P|r_j, \text{prec}, p_j = 1|k\text{-sched}, C_{\max}$.

In a fine-grained sense, the cases we left open are cases 3-20 from Table 1. We believe in fact algorithms in rows 5-6 and 10-20 are optimal: An $n^{o(k)}$ time algorithm for any case from result type [**C**] or [**D**] would imply either a $2^{o(n)}$ time algorithm for BICLIQUE or an $n^{o(k)}$ time algorithm for PARTITIONED SUBGRAPH ISOMORPHISM, which both would be surprising. It would be interesting to see whether for any of the remaining cases with precedence constraints and unit processing times a "sub-exponential" time algorithm exists.

A related case is $P3|\text{prec}, p_j = 1|C_{max}$ (where $P3$ denotes three machines). It is a famously hard open question (see e.g. [10]) whether this can be solved in polynomial time, but maybe it is doable to try to solve this question in sub-exponential time, e.g. $2^{o(n)}$?

──── **References** ────

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.
2 Stéphane Bessy and Rodolphe Giroudeau. Parameterized complexity of a coupled-task scheduling problem. *Journal of Scheduling*, 22(3):305–313, 2019.

3   Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 67–74. ACM, 2007.

4   Hans L. Bodlaender and Michael R. Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995.

5   Julia Chuzhoy, Rafail Ostrovsky, and Yuval Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Mathematics of Operations Research*, 31(4):730–738, 2006.

6   Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.

7   Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. Scheduling partially ordered jobs faster than 2^n. *Algorithmica*, 68(3):692–714, 2014. `doi:10.1007/s00453-012-9694-7`.

8   Joonyup Eun, Chang Sup Sung, and Eun-Seok Kim. Maximizing total job value on a single machine with job selection. *Journal of the Operational Research Society*, 68(9):998–1005, 2017.

9   Michael R. Fellows and Catherine McCartin. On the parametric complexity of schedules to minimize tardy tasks. *Theoretical Computer Science*, 298(2):317–324, 2003.

10   Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

11   Ron 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.

12   Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Danny Segev. Scheduling with outliers. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 149–162. Springer, 2009.

13   Danny Hermelin, Matthias Mnich, and Simon Omlor. Single machine batch scheduling to minimize the weighted number of tardy jobs. *CoRR*, 2019. `arXiv:1911.12350`.

14   Klaus Jansen, Felix Land, and Maren Kaluza. Precedence scheduling with unit execution time is equivalent to parametrized biclique. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 329–343. Springer, 2016.

15   Christos Koulamas and Shrikant S. Panwalkar. A note on combined job selection and sequencing problems. *Naval Research Logistics*, 60(6):449–453, 2013.

16   Christophe Lenté, Matthieu Liedloff, Ameur Soukhal, and Vincent T'kindt. Exponential algorithms for scheduling problems, 2014.

17   Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6:85–112, 2010.

18   Nicole Megow, Matthias Mnich, and Gerhard Woeginger. Lorentz Workshop 'Scheduling Meets Fixed-Parameter Tractability', 2019.

19   Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 2018.

20   Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1-2):533–562, 2015.

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

22   Jesper Nederlof and Céline Swennenhuis. Parameterized complexity of partial scheduling. *arXiv preprint*, 2019. `arXiv:1912.03185`.

23   Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer Publishing Company, Incorporated, 3rd edition, 2008.

24 Jirí Sgall. Open problems in throughput scheduling. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 2–11, 2012.

25 Dvir Shabtay, Nufar Gaspar, and Moshe Kaspi. A survey on offline scheduling with rejection. *Journal of Scheduling*, 16(1):3–28, 2013.

26 René van Bevern, Robert Bredereck, Laurent Bulteau, Christian Komusiewicz, Nimrod Talmon, and Gerhard J. Woeginger. Precedence-constrained scheduling problems parameterized by partial order width. In *Discrete Optimization and Operations Research - 9th International Conference, DOOR 2016, Vladivostok, Russia, September 19-23, 2016, Proceedings*, pages 105–120, 2016.

27 René van Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. *Journal of Scheduling*, 18(5):449–469, 2015.

28 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

29 Bibo Yang and Joseph Geunes. A single resource scheduling problem with job-selection flexibility, tardiness costs and controllable processing times. *Computers & Industrial Engineering*, 53(3):420–432, 2007.