Scheduling with precedence constraints, time windows and bounded proper level

Maher Mallem[1,*]

Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, UMR 5668, Lyon cedex 07, 69342, France

Claire Hanen

Université Paris Nanterre, Nanterre, 92000, France

Sorbonne Université, CNRS, LIP6, Paris, 75005, France

Alix Munier-Kordon

Sorbonne Université, CNRS, LIP6, Paris, 75005, France

Abstract

In this paper we study scheduling problems with release times, deadlines and precedence constraints where the objective is to minimize the makespan. We study both the single machine setting with arbitrary processing times and the parallel identical machine setting with unit processing times. We analyze the problem from the parameterized complexity point of view and propose proper level q as a parameter, which is the maximum number of time windows $[r_j, d_j)$ that strictly include a time window $[r_i, d_i)$ on both ends.

Under a natural assumption which we call *prec*-consistency, we propose several variants to the well-known "earliest deadline" dominance rule on both settings. On the single machine setting we show that this leads to fixed-parameter tractable algorithm with respect to proper level q. On the other hand, we show that the parallel identical machine setting with unit processing

*Corresponding author.

Email addresses: maher.mallem@ens-lyon.fr (Maher Mallem), claire.hanen@lip6.fr (Claire Hanen), alix.munier@lip6.fr (Alix Munier-Kordon)

[1]Part of this research project was conducted when this co-author was a PhD student at LIP6, Sorbonne Université.

times is unlikely to have fixed-parameter tractable algorithms with respect to the proper level, even when restricted to instances with either a fixed makespan threshold or bounded precedence width.

Keywords: parameterized complexity, scheduling, single machine, parallel identical machines

1. Introduction

Parameterized complexity theory has recently been widely used to study scheduling problems [15, 16, 22]. Given a parameter k, a problem is called *fixed-parameter tractable* (FPT) parameterized by k if any of its instances I can be solved in time $\mathcal{O}(f(k)\cdot poly(|I|))$ with f an arbitrary computable function and $|I|$ the instance size [7]. Then such a problem is considered tractable in the sense that the superpolynomial dependency on the time complexity is independent from the instance size. When the studied problem is believed to not be FPT, many complexity classes are available as parameterized analogues to NP like para-NP or the W-hierarchy [8, 11].

When parameterized complexity was first applied to scheduling, most results went against the existence of FPT algorithms with the studied problems and parameters. For example Bodlaender and Fellows showed that $P|prec, p_j = 1|C_{max}$ is W[2]-hard parameterized by the number m of machines [3]. This result was later improved to XNLP-completeness when the number of machines is paired with precedence width w [4]. One way to respond to such negative results is to introduce job time windows to the considered scheduling problem. While the base problem usually becomes harder, this opens up a larger choice of parameters which are especially useful to control the enumeration of schedules.

Two parameters have been prominent when considering scheduling problems with time windows. First, slack σ - which is defined as the maximum difference between the processing time of a job and its time window length - was successfully used in the context of identical parallel machine scheduling [6, 29] and single machine scheduling with setup times, and rejection [1]. When tackling the Resource-Constrained Project Scheduling Problem (RCPSP) in [28], van Bevern et al. inferred release dates from the precedence graph and showed that RCPSP is FPT parameterized by precedence width w combined with slack σ. Second, pathwidth μ is the maximum number of overlapping job time windows at any given time. Several problems with

arbitrary processing times and/or precedence constraints have been recently proved FPT parameterized by μ [17, 21], mostly when paired with either slack σ [1, 29] or maximum processing time p_{max} [13, 27]. When pairing pathwidth μ with no other parameter, FPT algorithms with respect to have been found for several strongly NP-hard problems - notably $1|prec, r_j, d_j|C_{max}$ [14] and $P|prec, p_j = 1, r_j, d_j|C_{max}$ [23]. Paired with a recent para-NP-completeness result on problem $P2|r_j, d_j|C_{max}$ [13], this has led to a comprehensive parameterized classification of scheduling problems with respect to μ - see Figure 1, which is extracted from [20].

Despite the recent success with pathwidth μ, it is worth noting that this parameter can overestimate the instance difficulty at times. Indeed when all jobs have the same time window the instance would be expected to be among the easier ones - especially linear time solvable in the single-machine case - yet it has maximum pathwidth. So, instead of considering all time window overlaps, the proper level introduced in [22] tracks whenever a time window $[r_j, d_j)$ strictly includes another time window $[r_i, d_i)$ on both ends. And it only accounts for such time window overlaps. When this is the case we say that job j *surrounds* job i. Proper level q is the maximum number of jobs j which can surround a job i. This definition of q is inspired from the graph classification work by Proskurowski and Telle in [24] where the q-proper graph classes were defined. Instead of a graph, this is applied to the interval model given by the job time windows.

In this paper problems $1|prec, r_j, d_j|C_{max}$ and $P|prec, p_j = 1, r_j, d_j|C_{max}$, which were both shown to be FPT parameterized by pathwidth μ, are considered. We establish whether they are FPT parameterized by proper level q. This work is an extended version of the paper presented at conference ISCO 2024 [22], which focused on the single-machine setting and proposed a FPT algorithm for it. Here we also study problem $P|prec, p_j = 1, r_j, d_j|C_{max}$. We establish dominance rules for the general case and the case where the precedence graph is an outforest. Yet we manage to prove that this problem is para-NP-hard with respect to proper level q in the general case. This puts proper level q in stark contrast to pathwidth μ as a parameter.

The paper is organized as follows: in Section 2 we give the necessary preliminaries to our approach. In Section 3 we present our variants to the *earliest deadline* rule. We recall the *weak earliest deadline* introduced in [22] and recall the properties of the schedules which follow this dominance rule on problem $1|prec, r_j, d_j|C_{max}$. We take inspiration from this rule to define a parallel-machine variant which leads to a subset of dominant schedules on

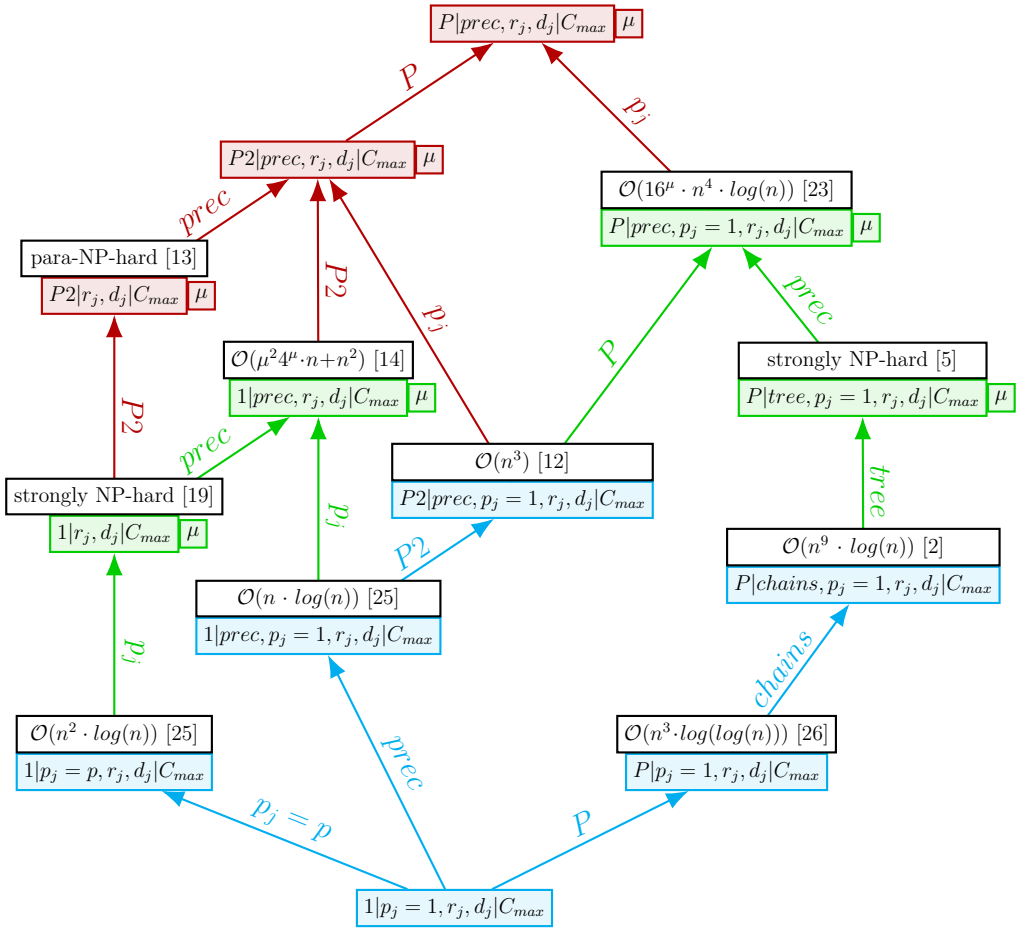


Figure 1: Parameterized complexity of scheduling problems with time windows with respect to pathwidth μ. Problems in blue can be solved in polynomial time, problems in green are FPT parameterized by μ and, unless P = NP, problems in red are *not* FPT with respect to μ. Arcs between problems represent inclusion relations, with each label indicating the changed property.

problem $P|prec, p_j = 1, r_j, d_j|C_{max}$. In Section 4 we focus on the single-machine setting. We present the dynamic programming algorithm proposed in [22], which computes the minimum makespan by only exploring the dominating scheduling prefixes. We show that this can be achieved in FPT time when parameterized by q. Then we present several variants of this algorithm, including one with a space complexity independent from the total number of jobs. Finally in Section 5 we consider problem $P|prec, p_j = 1, r_j, d_j|C_{max}$

and we show that it is unlikely to have fixed-parameter tractable algorithms with respect to the proper level, even when combined with either a fixed makespan threshold or a bounded precedence width.

2. Preliminaries

First we define our two studied scheduling problems. We consider a set of n jobs $\mathcal{J} = \{1, \ldots, n\}$ to be scheduled on a set of machines. Each machine can only process one job at a time. Each job j has a processing time $p_j \in \mathbb{N}$ and a time window $[r_j, d_j)$ with $r_j, d_j \in \mathbb{N}_0$ and $r_j < d_j$. Job j cannot start earlier than its release date r_j and must be completed no later than its deadline d_j. Moreover we consider precedence constraints given by a partial order relation $\rightarrow$ on $\mathcal{J}$: "$i \rightarrow j$" implies that j cannot start earlier than the completion of i. A feasible schedule $\sigma : \mathcal{J} \rightarrow \mathbb{N}_0$ assigns a starting time $\sigma(i)$ to each job i, following the previous constraints. Our optimization criterion is the makespan of the schedule, i.e. $C_{\max}(\sigma) = \max_{i \in \mathcal{J}}[\sigma(i) + p_i]$. In the case of problem $1|prec, r_j, d_j|C_{max}$ there is a single machine and no additional restrictions on job processing times. In the case of problem $P|prec, p_j = 1, r_j, d_j|C_{max}$ there are m identical parallel machines and all jobs have a unit processing time.

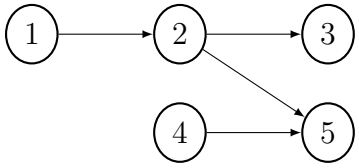
Since we have both time windows constraints and precedence relations in our problems, we must ensure that the time constraints are compatible with the partial order $\rightarrow$.

Definition 1. *A scheduling instance with release dates, deadlines and precedence constraints is called prec-consistent when:*
$\forall (i, j) \in [1, n]^2,\ (i \rightarrow j) \implies [(r_i + p_i \leq r_j) \wedge (d_i \leq d_j - p_j)]$.

Given a non *prec*-consistent instance, its release times and deadlines can be adjusted in time $\mathcal{O}(n^2)$ to fulfill this property by using path algorithms. Note that by doing so we only remove time window sections which are inaccessible according to precedence relations. So feasibility and the optimal makespan value are unchanged. While *prec*-consistency might look inconspicuous it is crucial to the dominance rule given in the next section.

Consider a *prec*-consistent instance. For the remainder of the paper we suppose that the jobs have been renamed in $[1, n]$ in lexicographic nondecreasing order of (deadline, release date). So when we compare two jobs i, j by writing "$i < j$" it means that: $(d_i < d_j) \vee [(d_i = d_j) \wedge (r_i \leq r_j)]$. We define proper sets and parameter q the following way:

i	1	2	3	4	5
p_i	2	2	3	4	1
r_i	0	3	6	0	5
d_i	6	8	11	13	14
Π_i	$\emptyset$	$\{4\}$	$\{4,5\}$	$\emptyset$	$\emptyset$
$\lvert\Pi_i\rvert$	0	1	2	0	0



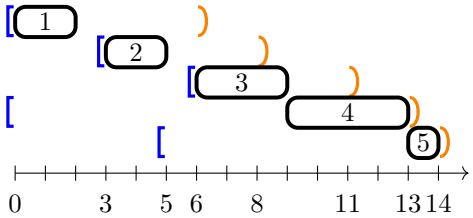


Figure 2: An instance of $1|prec, r_j, d_j|C_{max}$ with five jobs and proper level two.

Definition 2. *(Proper set Π_i) Let $i \in [1,n]$. We say that a job $j \in [1,n]$ surrounds job i when $r_j < r_i$ and $d_i < d_j$. In other words j surrounds i when time window $[r_i, d_i)$ is strictly included by time window $[r_j, d_j)$ on both ends. We define $\Pi_i = \{j \in [1,n], j$ surrounds $i\}$.*

(proper level) We define parameter $q = max_{i\in[1,n]}|\Pi_i|$. In other words q is the maximum number of jobs j which can surround a job i.

Finally like in [10] we highlight the following subset of jobs:

Definition 3. *We say that a job $j \in [1,n]$ is a summit when j surrounds no job. In other words: for all jobs i in $[1,n]$ j is not in Π_i. The summits are denoted $s_1 < \ldots < s_N$ with N the number of summits.*

An example of a *prec*-consistent instance is given in Figure 2. Jobs 1,2 and 3 surround no other job, so they are the summits of this instance. We conclude this section with the following properties:

Lemma 4. *(i) Every job either is a summit or surrounds a summit.*

(ii) If $s_i < s_k < j$ and $j \in \Pi_{s_i}$ then $j \in \Pi_{s_k}$.

Proof. (i) Let j be a job which is not a summit. Then j must surround at least one job j_1. Then either j_1 is a summit or it surrounds some job j_2. After k non-summit steps we would have $r_j < r_{j_1} < \ldots < r_{j_k}$ and

$d_{j_k} < \cdots < d_{j_1} < d_j$. Thus all these jobs are distinct from each other and it eventually ends with some job j_K which is a summit. Then we also have $r_j < r_{j_K}$ and $d_{j_K} < d_j$, so j surrounds summit j_K.

(ii) Let $s_i < s_k < j$ such that $j \in \Pi_{s_i}$. s_k is a summit so $s_k \notin \Pi_{s_i}$. And since $s_i < s_k$ we necessarily have $r_{s_i} \leq r_{s_k}$. So $r_j < r_{s_i} \leq r_{s_k}$. Now $s_k < j$ implies that $d_{s_k} \leq d_j$. If $d_{s_k} = d_j$ then we would have $j < s_k$ which leads to a contradiction. So $d_{s_k} < d_j$ and thus $j \in \Pi_{s_k}$. □

3. Earliest Deadline Rule Variants with Time Windows

In this section we propose several variants to the *earliest deadline first* rule which take into account job time windows. We consider problems $1|prec, r_j, d_j|C_{max}$ and $P|prec, p_j = 1, r_j, d_j|C_{max}$ and establish the dominance of schedules following these rules. Then we discuss whether such optimal schedules can be further modified to get optimal schedules with a stronger structure based on the summits of the instance.

Note that for any scheduling rule which is based on the order of release dates and/or deadlines, *prec*-consistency is essentially required to obtain a dominance result. The main goal of this restriction is to ensure that the time window-based job order used by the rule - like our lexicographic order $<$ here - does not contradict with some precedence relation. When this is not the case, as in the example given in Figure 3, said job order could recommend to schedule a job earlier than (or at the same time as) one of its predecessors.

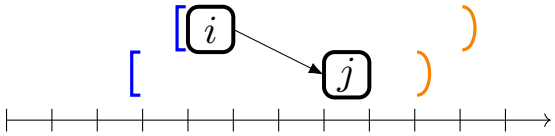


Figure 3: A non *prec*-consistent single-machine counterexample to all three dominance rules defined in Section 3.

3.1. Definition and dominance

First we present the (wED) rule:

Definition 5. *A feasible schedule σ on an instance of $1|prec, r_j, d_j|C_{max}$ follows the (wED) rule when: $\forall (i, j) \in [1, n]^2$, if $i < j$ then either:*

(*i*) *i is scheduled before j (i.e.* $\sigma(i) < \sigma(j)$*) or*

(*ii*) *j surrounds i (i.e.* $j \in \Pi_i$*) or*

(*iii*) *there exists* $h < i$ *such that* $\sigma(j) < \sigma(h) < \sigma(i)$.

Given a *prec*-consistent instance of $1|prec, r_j, d_j|C_{max}$ the schedules following the (wED) rule were shown to be dominant.

Property 6. *Given a feasible schedule* σ *on a prec-consistent instance of problem* $1|prec, r_j, d_j|C_{max}$*, one can build a feasible schedule with a makespan lower than or equal to* σ *and which follows the (wED) rule.*

Proof. Let σ be a feasible schedule. If σ follows the (wED) rule then we are done. If not then let (i, j) be a job couple such that $i < j$, $\sigma(j) < \sigma(i)$, $j \notin \Pi_i$ and all jobs between j and i in σ are higher than i. Then in σ we have "jSi" where j and all jobs in sequence S are higher than i. Within the same time interval we propose job reordering "ijS" the following way: push sequence "jS" to the right by p_i time units then insert i right before. Indeed $i < j$ and $j \notin \Pi_i$ so necessarily we have $r_i \leq r_j$. Plus j and all jobs in S are greater than i. Since our instance is *prec*-consistent it means that there is no precedence relation from any of these jobs to i. Thus i can be inserted as desired. Now again j and all jobs in sequence S are greater than i. So their deadlines are non lower than d_i and they can be pushed as desired. Note that job couple (i, j) and all couples (i, s) with s in sequence S are (wED)-valid, now that i is scheduled before them. Thus the resulting schedule is feasible, has the same makespan as σ, and has at least one less (wED)-invalidating job couple with (i, j) becoming (wED)-valid and no job couple becoming (wED)-invalidating. This procedure can be repeated a maximum of $\frac{n(n-1)}{2}$ times until we get a feasible schedule which follows the (wED) rule. □

Now consider problem $P|prec, p_j = 1, r_j, d_j|C_{max}$. If we have more than one machine, then multiple jobs can be scheduled at the same time unit. In the example given in Figure 4 jobs 1 and 2 cannot both start at time 0 at the same time according the (wED) rule, whereas this is the only way to get a schedule of minimum makespan for this instance.

In response we propose a first relaxation to the (wED) rule which aims to take into account this issue.

Figure 4: Counterexample to the (wED) rule on problem $P|prec, p_j = 1, r_j, d_j|C_{max}$ with $m = 2$ machines.

Definition 7. *Given an instance of $P|prec, p_j = 1, r_j, d_j|C_{max}$, a feasible schedule σ follows the (wED/P_1) rule when: $\forall (i,j) \in [1,n]^2$, if $i < j$ then either:*

(i) i is scheduled before or at the same time as j (i.e. $\sigma(i) \leq \sigma(j)$) or

(ii) j surrounds i (i.e. $j \in \Pi_i$) or

(iii) there exists $h < i$ such that $\sigma(j) \leq \sigma(h) < \sigma(i)$.

Unfortunately, the schedules following the (wED|P_1) rule are not necessarily dominant on *prec*-consistent instances of $P|prec, p_j = 1, r_j, d_j|C_{max}$. As illustrated in Figure 5 this can happen when several jobs have the same predecessor. Here job couple $(1, 2)$ invalidates the (wED|P_1) rule, yet job 1 cannot be scheduled earlier than job 2 without scheduling job 4 at time 2 or later. So the (wED|P_1)-following schedules are not dominant in this example.

However we show that dominance can be obtained when every job has at most one predecessor - i.e. when the precedence graph is an outforest.

Property 8. *Given a feasible schedule σ on a prec-consistent instance of problem $P|outforest, p_j = 1, r_j, d_j|C_{max}$, one can build a feasible schedule with a makespan lower than or equal to σ and which follows the (wED/P_1) rule.*

Proof. Let σ be a feasible schedule. If σ follows the (wED|P_1) rule then we are done. If not then there exists a job couple (i, j) such that $i < j$, $\sigma(j) < \sigma(i)$, $j \notin \Pi_i$, and all jobs scheduled within interval $[\sigma(j), \sigma(i) - 1]$ in σ are higher than i. Among these couple take i minimum and j associated to a job couple with i and scheduled the earliest in σ.

Our goal is to have i scheduled at time $\sigma(j)$. Given that $i < j$ and $j \notin \Pi_i$, we indeed know that $r_i \leq r_j \leq \sigma(j)$. And since all jobs scheduled

i	1	2	3	4
r_i	0	0	0	1
d_i	2	3	4	5
Π_i	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

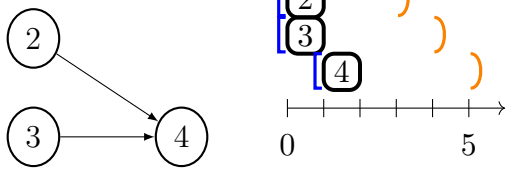


Figure 5: Counterexample to the (wED|P_1) rule on problem $P|prec, p_j = 1, r_j, d_j|C_{max}$ with $m = 2$ machines.

within interval $[\sigma(j), \sigma(i) - 1]$ in σ are higher than i, by *prec*-consistency there is no precedence relation from one of these jobs to i. Now recall that *prec* is an outforest, meaning that every job has at most one predecessor. So there is a job $j' > i$ scheduled at time $\sigma(i) - 1$ which has no successor scheduled at time $\sigma(i)$. So we can permute j' and i while keeping feasibility and the same makespan value. This argument can be applied to time units $\sigma(i) - 1, \ldots, \sigma(j)$ successively until i is scheduled at time $\sigma(j)$. At this point we know that we managed to schedule i earlier than or at the same time as j.

Now recall that j was one of the earliest jobs in σ which were part of a job couple with i. So, after this permutation, all other jobs j' which were part of a job couple are now scheduled later than or at the same time as i. This means that i is no longer part of a job couple invalidating the (wED|P_1) rule. Thus the resulting schedule is feasible, has the same makespan as σ and has a higher minimum job i among its (wED|P_1)-invalidating job couples. This procedure can be repeated a maximum of $(n - 1)$ times until we get a feasible schedule with the same makespan as σ which follows the (wED|P_1) rule. □

In the light of this proof and the given counterexample, clearly our main obstacles are the precedence relations between jobs scheduled within interval $[\sigma(j)+1, \sigma(i)]$. In particular i and j cannot be permuted if j is the predecessor to one of these jobs. As such we propose the following variant:

Definition 9. *Given an instance of problem $P|prec, p_j = 1, r_j, d_j|C_{max}$, the (wED/$P_2$) rule corresponds the (wED/P_1) rule but with the following extra case in the implication:*

(iv) there exists $k > j$ such that $\sigma(j) < \sigma(k) \leq \sigma(i)$ and $j \to k$.

Given a *prec*-consistent instance of $P|prec, p_j = 1, r_j, d_j|C_{max}$ we show that the schedules following the (wED|P_2) rule are dominant.

Property 10. *Given a feasible schedule σ on a prec-consistent instance of problem $P|prec, p_j = 1, r_j, d_j|C_{max}$, one can build a feasible schedule with a makespan lower than or equal to σ and which follows the (wED|P_2) rule.*

Proof. We proceed in a similar fashion as in the proof of Lemma 8. If σ does not follow the (wED|P_2) rule then there exists a job couple (i, j) such that $i < j$, $\sigma(j) < \sigma(i)$, $j \notin \Pi_i$, all jobs scheduled within interval $[\sigma(j), \sigma(i) - 1]$ in σ are higher than i, and there is no precedence relation from j to any job scheduled within interval $[\sigma(j) + 1, \sigma(i)]$ in σ. Among these couple take i minimum and j associated to a job couple with i and scheduled the earliest in σ.

Again we want to have i scheduled at time $\sigma(j)$. Like in the proof of Lemma 8 our hypotheses infer that $r_i \leq \sigma(j)$ and there is no precedence relation from any job scheduled within interval $[\sigma(j), \sigma(i) - 1]$ to i. However this time we also know that there is no precedence relation from j to any job scheduled within interval $[\sigma(j) + 1, \sigma(i)]$. So i and j can be directly permuted with no impact on feasibility and the value of the makespan. By earliness of j and minimality of i we conclude the same way as in the proof of Lemma 8. □

3.2. Summit-based decompositions

We present a summit-based decomposition on problem $1|prec, r_j, d_j|C_{max}$. This is based on the following key lemma, which establishes several summit-based properties of the schedules that follow the (wED) rule.

Lemma 11. *Given a prec-consistent instance of $1|prec, r_j, d_j|C_{max}$, every schedule which is feasible and which follows the (wED) rule is of the form $T_1 s_1 \dots T_N s_N T_{N+1}$ where:*

(i) $1 = s_1 < \dots < s_N$ *are the summits of the instance,*

(ii) for all k in $[1, N]$ and $i < s_k$ we have $\sigma(i) < \sigma(s_k)$,

(iii) for all k in $[1, N]$ if $j > s_k$ and $\sigma(j) < \sigma(s_k)$ then $j \in \Pi_{s_k}$,

(iv) a. every job in sequence T_1 is in set Π_{s_1},

b. *for all* k *in* $[2, N]$ *every job in sequence* T_k *is in set* $\Pi_{s_{k-1}} \cup \Pi_{s_k}$, *and*

c. *every job in sequence* T_{N+1} *is in set* Π_{s_N}.

Proof. Let σ be a feasible schedule which follows the (wED) rule. We start by proving point (ii). Let $k \in [1, N]$ and $i < s_k$. We show that $\sigma(i) < \sigma(s_k)$. By the (wED) rule either $\sigma(i) < \sigma(s_k)$, or $s_k \in \Pi_i$, or there is some $h < i$ such that $\sigma(s_k) < \sigma(h) < \sigma(i)$. s_k is a summit so by its definition the second case is not possible. Now by contradiction suppose that the third case is true. Take h as the lowest job such that $\sigma(s_k) < \sigma(h) < \sigma(i)$. Then couple (h, s_k) must follow the (wED) rule while $\sigma(s_k) < \sigma(h)$ and there is no $h' < h$ such that $\sigma(s_k) < \sigma(h') < \sigma(h)$ by minimality of h. Then it means that s_k is in Π_h, which contradicts that s_k is a summit. Thus by the (wED) rule this only leaves us with $\sigma(i) < \sigma(s_k)$.

Next we prove point (iii). Let $k \in [1, N]$ and $j > s_k$ such that $\sigma(j) < \sigma(s_k)$. Then by the (wED) rule either $j \in \Pi_{s_k}$ or there is $h < s_k$ such that $\sigma(j) < \sigma(h) < \sigma(s_k)$. In the first case we are done, so suppose we are in the second phase. Take h minimum. Then by the (wED) rule $j \in \Pi_h$. But according to Lemma 4 (i) either h is a summit or it surrounds a summit. In the first case Lemma 4 (ii) would show that $j \in \Pi_{s_k}$ and we are done. In the second case we have a summit $s_i < h$ such that $r_j < r_h < r_{s_i}$ and $d_{s_i} < d_h < d_j$. Then $j \in \Pi_{s_i}$ and again Lemma 4 (ii) can be used to show that $j \in \Pi_{s_k}$.

Now consider two consecutive summits s_k and s_{k+1}. s_k is lower than s_{k+1}. So by point (ii) we know that s_k is scheduled before s_{k+1} in σ. This means that σ is of the form $T_1 s_1 \dots T_N s_N T_{N+1}$ where all the jobs in sequences T_k are non-summits. Finally note job 1 has the lowest deadline and the lowest release date among the jobs with the same deadline. So job 1 is always a summit and $s_1 = 1$. This proves point (i).

Finally we prove point (iv). Let $k \in [1, N+1]$ and j be a job in sequence T_k.

a. If $k = 1$ then $s_1 = 1$, so $j > s_1$ and $\sigma(j) < \sigma(s_1)$. Thus by point (iii) $j \in \Pi_{s_1}$.

c. If $k = N+1$: job j is not a summit so by Lemma 4 (i) it surrounds one summit s_ℓ. But necessarily $s_\ell \leq s_N$. If $\ell = N$ then we are done. Else we know that $\sigma(s_N) < \sigma(j)$ so by point (ii) $j > s_N$. Then $s_\ell < s_N < j$ with j surrounding s_ℓ. By Lemma 4 (ii) j also surrounds s_N.

b. If $k \in [2, N]$: again job j is not a summit so by Lemma 4 (i) it surrounds one summit s_ℓ. If $\ell \leq k-1$ then the same reasoning as in point *c.* can be applied to show that $j \in \Pi_{s_{k-1}}$. Now suppose $\ell \geq k$. Then we have $s_k \leq s_\ell < j$ and $\sigma(j) < \sigma(s_k)$. Thus by point (iii) $j \in \Pi_{s_k}$.

□

Then in any (wED)-following schedule of our single-machine problem every job lower than a summit s_k must be scheduled before s_k. And among the jobs higher than s_k only those in Π_{s_k} can be scheduled before s_k. With the next section methodology it would already lead to an FPT dynamic programming algorithm with $\mathcal{O}((2q)! \cdot 4^q \cdot N)$ states. However this number can be decreased to $\mathcal{O}(2^q \cdot N)$ by restricting further the set of dominant schedules. This was done by fixing the job order in each T_k.

Definition 12. *Consider a prec-consistent instance of* $1|prec, r_j, d_j|C_{max}$. *A schedule of this instance is called summit-ordered when it is feasible and of the form* $T_1 s_1 \dots T_N s_N T_{N+1}$ *where properties* $(i), (ii), (iii)$ *of Lemma 11 are satisfied and the following property holds:*

(iv)
- *a. all jobs in sequence* T_1 *are in* Π_{s_1} *and scheduled in nondecreasing order of their release date.*
- *b. for all* k *in* $[1, N-1]$ $T_{k+1} = C_k A_{k+1} B_{k+1}$ *where:*
  - $*$ *all jobs in sequence* C_k *are in* Π_{s_k}, *not in* $\Pi_{s_{k+1}}$ *and scheduled in nondecreasing order of their deadline,*
  - $*$ *all jobs in sequence* A_{k+1} *are in* $\Pi_{s_k} \cap \Pi_{s_{k+1}}$ *and scheduled in any order (say in their lexicographic order),*
  - $*$ *all jobs in sequence* B_{k+1} *are in* $\Pi_{s_{k+1}}$, *not in* Π_{s_k} *and scheduled in nondecreasing order of their release date.*
- *c. all jobs in sequence* T_{N+1} *are in* Π_{s_N} *and scheduled in nondecreasing order of their deadline.*

For example consider the instance given in Figure 2. A feasible schedule with job order "12453" (if it exists) would be *summit-ordered* with $C_2 = \emptyset$, $A_3 = \{4\}$ and $B_3 = \{5\}$, while one with job order "12543" would not be. *summit-ordered* were shown to be dominant on problem $1|prec, r_j, d_j|C_{max}$.

Property 13. *On problem* $1|prec, r_j, d_j|C_{max}$ *the summit-ordered schedules are dominant.*

Proof. Let σ be a feasible schedule. From Lemma 6 and Lemma 11 one can build $\bar{\sigma}$ of the form $T_1 s_1 \dots T_N s_N T_{N+1}$ with a makespan lower or equal to σ which follows the (wED) rule and verify points (i), (ii) and (iii). In order to prove point (iv) we propose a reordering of each sequence T_k with $k \in [1, N+1]$.

Let $k \in [1, N-1]$. Consider sequence "$s_k T_{k+1} s_{k+1}$". First we want to reorder each sequence T_{k+1} into a form $C_k A_{k+1} B_{k+1}$. Given the jobs in T_{k+1}, C_k groups those in Π_{s_k} and not in $\Pi_{s_{k+1}}$, A_{k+1} groups those common to both proper sets, and B_{k+1} groups those in $\Pi_{s_{k+1}}$ and not in Π_{s_k}. If T_{k+1} is not of this form then we first check the jobs from C_k. If there is a job from C_k scheduled after a job of $A_{k+1} \cup B_{k+1}$, let c be the leftmost of them. Then we have "$s_k C S c$" in schedule $\bar{\sigma}$ where the jobs in C are from C_k and the jobs in S are from $A_{k+1} \cup B_{k+1}$. We propose to reorder into "$s_k C c S$" the following way: push sequence S to the right by p_c units then insert c right before S. Indeed all jobs from S are in $\Pi_{s_{k+1}}$ so their deadlines are higher than $\sigma(s_{k+1})$ and they could be pushed to the right as desired. Plus c is in Π_{s_k} so its release date is lower than $\sigma(s_k)$ and it could be inserted as desired. So the only obstacle would be a precedence relation between some job α in S to job c. On the one hand α is in $\Pi_{s_{k+1}}$ so it is greater than s_{k+1}. On the other hand c is not in $\Pi_{s_{k+1}}$ and is scheduled before s_{k+1} in $\bar{\sigma}$. By Lemma 11 (iii) this means that c is lower than s_{k+1} and thus $c < \alpha$. Recall that our instance is prec-consistent, so it means that there cannot be a precedence relation from α to c. Thus the proposed reordering "$s_k C c S$" is allowed and does not increase the makespan of our schedule. Repeating this at most $|C_k|$ times allows us to get all jobs from C_k on the left side of interval $[\sigma(s_k)+p_{s_k}, \sigma(s_{k+1}))$. Then one can show that all jobs from B_{k+1} can be grouped on the right side of interval $[\sigma(s_k)+p_{s_k}, \sigma(s_{k+1}))$ with symmetric arguments.

Now let $k \in [1, N]$. We show that the jobs in C_k can be reordered in nondecreasing order of their deadline. Notice that jobs in $C_k \subset \Pi_{s_k}$ have release time lower than r_{s_k}. So they can be scheduled without idle time after s_k. If two consecutive jobs i, j of C_k satisfy $d_i > d_j$, we cannot have $i \to j$ as our instance is prec-consistent. So i, j can be swapped without modifying the feasibility nor the makespan of the schedule. Iterating such swaps leads to the desired order. Similarly one can show that all jobs from B_k can be reordered in nondecreasing order of their release dates with symmetric arguments. □

On the other hand, on problem $P|prec, r_j, d_j|C_{max}$ it must be noted that

i	1	2	3	4
r_i	0	0	0	0
d_i	2	3	4	5
Π_i	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

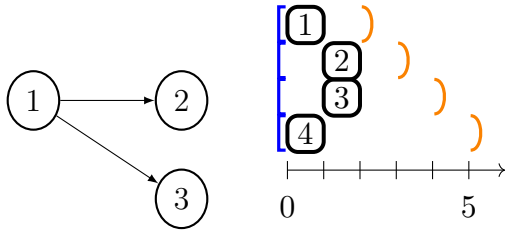


Figure 6: Counterexample to properties (ii) and (iii) of Lemma 11 with the (wED|P_1) rule on problem $P|outtree, p_j = 1, r_j, d_j|C_{max}$ with $m = 2$ machines.

we cannot rely on a similar summit-based decomposition with either the (wED|P_1) rule or the (wED|P_2) rule, even when restricting ourselves to tree-like precedence relations. In the example given in Figure 6 the only schedule of optimal makespan 2 starts job 4 at time 0. With respect to Lemma 11 this contradicts both property (ii) - job 4 is a summit yet jobs 2 and 3 are scheduled at a later time - and property (iii) - job 3 is a summit and job 4 is scheduled at an earlier time, yet job 4 is not in the proper set of job 3. While this does not rule out the existence of a different kind of summit-based decomposition, the latter would need to allow some summits to be scheduled earlier than prior summits.

4. A Single-Machine FPT Algorithm

In this section we focus on problem $1|prec, r_j, d_j|C_{max}$. We present a dynamic programming algorithm which explores active *summit-ordered* schedules - "active" meaning that there is no unnecessary waiting time.

4.1. Core concept

Assume that we have built a partial *summit-ordered* schedule until a summit s_k. To extend this schedule to other jobs, we need to keep track of the information which ensures that each job is scheduled exactly once. Using the definition of *summit-ordered* schedules we show that we only need to recall the set L_k of jobs greater than summit s_k and scheduled before s_k. To this purpose we define the following set:

Definition 14. *(Lingering set Λ_k) We define* $\Lambda_k = (\bigcup_{1 \leq h \leq k} \Pi_{s_h}) \cap [s_k + 1, n]$.

Then a *summit-ordered* schedule $T_1 s_1 \dots T_N s_N T_{N+1}$ can be represented as a state path $(s_1, L_1) \to (\dots) \to (s_N, L_N)$ where every L_k is a subset of Λ_k. Such a path decomposition is motivated by the following lemma:

Lemma 15. *Let $\sigma = T_1 s_1 \dots T_N s_N T_{N+1}$ be an active summit-ordered schedule on an instance I of $1|prec, r_j, d_j|C_{max}$. Then for every k in $[1, N]$ schedule $T_1 s_1 \dots T_k s_k$ is a feasible active schedule on job subset $[1, s_k] \cup L_k$ where $L_k = (\bigcup_{1 \leq h \leq k} T_h) \cap [s_k + 1, n]$.*

Proof. Feasibility and activeness are stable properties by prefix operation. What is left to show is that the set of jobs featured in partial schedule $T_1 s_1 \dots T_k s_k$ is indeed $[1, s_k] \cup L_k$.

($\subseteq$) Since we have a *summit-ordered* schedule, by Definition 12 (i) summits $s_1, \dots, s_k$ are all lower or equal to s_k, so they are all included in $[1, s_k]$. Now given $\ell \in [1, k]$ and a job $j \in T_\ell$: if $j \leq s_k$ then $j \in [1, s_k]$, else we have $j \in [s_k + 1, n]$ and $j \in \bigcup_{1 \leq h \leq k} T_h$, so $j \in L_k$.

($\supseteq$) By definition L_k is included in $\bigcup_{1 \leq h \leq k} T_h$. Now let $j \in [1, s_k]$. Since we have a *summit-ordered* schedule, by Definition 12 (ii) all jobs lower than s_k are scheduled before s_k in σ. Thus either $j = s_\ell$ or $j \in T_\ell$ for some ℓ in $[1, k]$. □

Then the corresponding schedule can be retrieved solely from the sets L_k and the pre-processed proper sets of the summits:

Lemma 16. *Let $\sigma = T_1 s_1 \dots T_N s_N T_{N+1}$ be an active summit-ordered schedule on an instance I of $1|prec, r_j, d_j|C_{max}$. Let $(s_1, L_1) \to (\dots) \to (s_N, L_N)$ be the corresponding state path. Then for every k in $[1, N]$:*

a. $B_1 = L_1$,

b. for every k in $[2, N-1]$:

- $C_k = \{j \in \Pi_{s_k}, j \notin L_k \cup \Pi_{s_{k+1}}\}$,
- $A_{k+1} = \{j \in \Pi_{s_k} \cap \Pi_{s_{k+1}} \cap L_{k+1}, j \notin L_k\}$,
- $B_{k+1} = \{j \in \Pi_{s_{k+1}} \cap L_{k+1}, j \notin \Pi_{s_k} \cup L_k\}$,

c. $C_N = \{j \in \Pi_{s_N}, j \notin L_N\}$.

Proof. This directly comes from Definition 12 (iv) and Lemma 15. □

Finally, in order to find the optimal makespan of our instance, we show that we only need to consider *summit-ordered* schedules which are optimal on every prefix $T_1 s_1 \dots T_k s_k$:

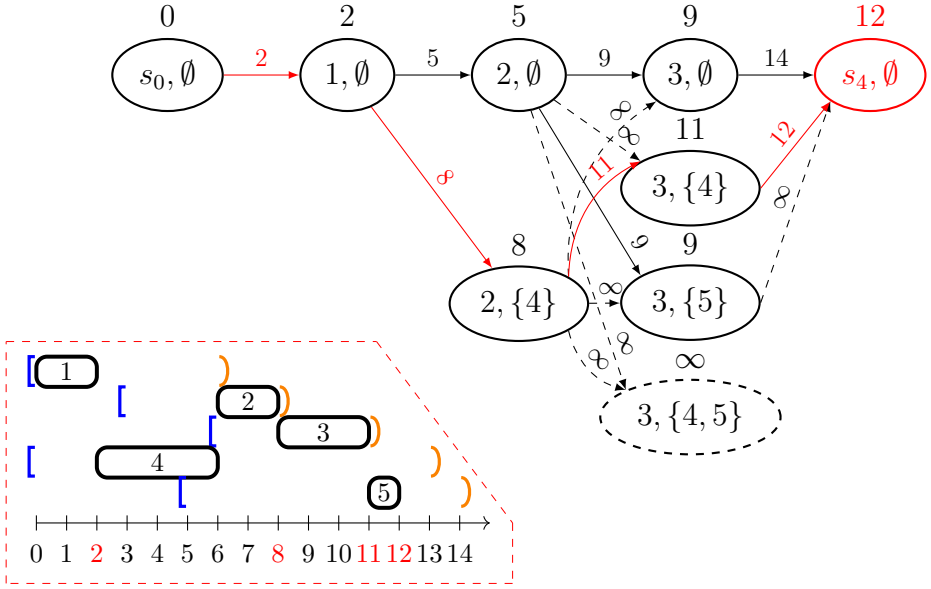


Figure 7: State graph associated to the example given in Figure 2. The value of each arc is the makespan of the scheduling candidate computed by extend(k, L_k, L_{k+1}). The red path corresponds to active schedule "14235", which gives the optimal makespan.

Lemma 17. *Let $\sigma = T_1 s_1 \dots T_N s_N T_{N+1}$ be an active summit-ordered schedule with optimal makespan on an instance I of $1|prec, r_j, d_j|C_{max}$. Then one can build an active summit-ordered schedule $\sigma' = T'_1 s_1 \dots T'_N s_N T'_{N+1}$ also with optimal makespan and such that for all $k \in [1, N]$ schedule prefix $T'_1 s_1 \dots T'_k s_k$ has optimal makespan among the feasible schedules of state $(s_k, \bigcup_{1 \leq h \leq k} T'_h \cap [s_k + 1, n])$.*

Proof. This is proved by downward induction on k. Suppose that for all $j > k$ schedule $T_1 s_1 \dots T_j s_j$ is optimal among the schedules associated to the same state - which is true in base case $k = N$. Let τ be a schedule associated to the same state as schedule $T_1 s_1 \dots T_k s_k$ and with optimal makespan. Then by Proposition 13 one can build $\tau' = T'_1 s_1 \dots T'_k s_k$ with the corresponding properties and the same makespan as τ. We propose schedule $\sigma' = \tau \cdot T_{k+1} s_{k+1} \dots T_N s_N T_{N+1}$, for which the result holds for all $j \geq k$. This concludes the induction. □

This motivates the procedure given in Algorithm 1. Set dummy summits s_0, s_{N+1} with processing time 0, an initial state $(s_0, \emptyset)$ and a final state $(s_{N+1}, \emptyset)$. For $k = 0$ to N use the optimal makespan from states (s_k, L_k) and extend them to the states (s_{k+1}, L_{k+1}) accordingly to Lemma 16. For

Algorithm 1 main()

preprocessing() ▷ Check *prec*-consistency. Compute proper sets and lingering sets.
$r_{s_0}, p_{s_0}, d_{s_0}, p_{s_{N+1}} \leftarrow 0$; $r_{s_{N+1}} \leftarrow max_{i\in[1,n]}(r_i)$; $d_{s_{N+1}} \leftarrow \infty$
$\Pi_{s_0}, \Lambda_0, \Pi_{s_{N+1}}, \Lambda_{N+1} \leftarrow \emptyset$
Create table T with $N+2$ columns and $2^{|\Lambda_k|}$ slots in each column k.
Initialize T with ∞ everywhere.
$T[0][0] \leftarrow 0$
for $k = 0$ to N **do**
    for each $(L_k, L_{k+1}) \subseteq \Lambda_k \times \Lambda_{k+1}$ **do**
        $T[k+1, L_{k+1}] \leftarrow \min(T[k+1, L_{k+1}], \text{extend}(k, L_k, L_{k+1}))$
return $T[N+1, \emptyset]$

each state pair either the resulting schedule is invalid or it is an optimal makespan candidate for the successor. At the end of the procedure the optimal makespan is given by the value of state $(s_{N+1}, \emptyset)$.

The state graph corresponding to the Figure 2 example is given in Figure 7. We have $\Lambda_1 = \emptyset, \Lambda_2 = \{4\}$ and $\Lambda_3 = \{4, 5\}$. The optimal makespan is obtained with *summit-ordered* active schedule "14235", which corresponds to state path $(s_0, \emptyset) \rightarrow (1, \emptyset) \rightarrow (2, \{4\}) \rightarrow (3, \{4\}) \rightarrow (s_4, \emptyset)$.

4.2. State bounding

In this subsection we bound the size of lingering sets Λ_k. It turns out that only the proper set of summit s_k is needed to obtain Λ_k:

Lemma 18. $\forall k \in [1, N]$, $\Lambda_k = \Pi_{s_k}$.

Proof. Let $k \in [1, n]$. By definition every job j in Π_{s_k} is greater than s_k. So j is in Λ_k. Now let $\ell \in \Lambda_k$. We show that $\ell \in \Pi_{s_k}$. By the definition of Λ_k there is $j \in [1, k]$ such that $\ell \in \Pi_{s_j}$. If $j = k$ then we are done. Else we have $s_j < s_k < \ell$ with $\ell \in \Pi_{s_j}$. Then by Lemma 4 (ii) $\ell \in \Pi_{s_k}$. □

This bounds the size of Λ_k and the number of successors (s_{k+1}, L_{k+1}) for each state by parameter q:

Corollary 19. $\forall k \in [1, n]$, $|\Lambda_k| \leq q$. *Plus every state has at most* 2^q *successors.*

Algorithm 2 extend(k, L_k, L_{k+1})

▷ Input: $k \in [0, N], L_k \subseteq \Lambda_k, L_{k+1} \subseteq \Lambda_{k+1}$.
▷ Goal: start from an optimal schedule on job subset $[1, s_k] \cup L_k$ and attempt to extend it to job subset $[1, s_{k+1}] \cup L_{k+1}$.
$Cmax \leftarrow T[k, L_k]$
if $[1, s_k] \cup L_k \not\subseteq [1, s_{k+1}] \cup L_{k+1}$ or $Cmax = \infty$ **then return** ∞
▷ Add jobs between summits s_k and s_{k+1} accordingly to Lemma 16.
$C_k \leftarrow list(\{j \in \Pi_{s_k}, j \notin L_k \cup \Pi_{s_{k+1}}\})$
sort(C_k, nondecreasing d_j)
$A_{k+1} \leftarrow list(\{j \in \Pi_{s_k} \cap \Pi_{s_{k+1}} \cap L_{k+1}, j \notin L_k\})$
$B_{k+1} \leftarrow list(\{j \in \Pi_{s_{k+1}} \cap L_{k+1}, j \notin \Pi_{s_k} \cup L_k\})$
sort(B_{k+1}, nondecreasing r_j)
$add \leftarrow C_k \cdot A_{k+1} \cdot B_{k+1} \cdot [s_{k+1}]$
if prec_invalid(L_k, add) **then return** ∞
for j in add (in order) **do**
    $Cmax \leftarrow \max(Cmax, r_j) + p_j$
    if $Cmax > d_j$ **then return** ∞
return $Cmax$

4.3. Complexity

Now that the number of states and successors have been bounded we compute the time and space complexity of the proposed dynamic programming algorithm. In the preprocessing phase *prec*-consistency is checked then proper sets and lingering sets are computed. This takes time $\mathcal{O}(n^2)$ and space $\mathcal{O}(q \cdot N)$. Now by Corollary 19 each column has at most 2^q slots and for each one at most 2^q candidate successors are explored. For each couple $((s_k, L_k), (s_{k+1}, L_{k+1}))$ at most $2q + 1$ jobs are added to the schedule. By Lemma 16 they can be retrieved from sets L_k and L_{k+1} and appended accordingly to Definition 12 in time $\mathcal{O}(q \cdot log(q))$.

Now only the precedence checks related to the added jobs in $T_{k+1} \cup \{s_{k+1}\}$ are left. Since we build a *summit-ordered* schedule, following Definition 12 (iv), there is no invalidating precedence constraint from one job in $T_{k+1} \cup \{s_{k+1}\}$ to another. So an invalidating one would necessarily go from a job i in $T_{k+1} \cup \{s_{k+1}\}$ to a job j in $\bigcup_{1 \leq h \leq k} T_h \cup \{s_h\}$. Since our instance is *prec*-consistent j must be greater than i, which is itself greater than or equal to s_{k+1} and thus greater than s_k. So $j \in L_k$. By Corollary 19 this leaves at most $2q + 1$ possible jobs at the start of the potentially invalidating precedence constraint and at most q possible jobs at the end of it. So the precedence checks take time $\mathcal{O}(q^2)$ for any couple $((s_k, L_k), (s_{k+1}, L_{k+1}))$.

Then either some deadline/precedence constraint is invalidated or the jobs are successfully added and a new candidate makespan value is proposed to the successor. So the main loop of the algorithm takes time $\mathcal{O}(q^2 \cdot 4^q \cdot N)$. Now we consider space complexity. For each state (i, L) we must remember index i, some encoding of set L and the minimum makespan currently found. By Corollary 19 this requires space $\mathcal{O}(q + log(D))$ where $D = max_{i \in [1,n]}(d_i)$. Finally once the whole state graph has been computed, a feasible schedule with optimal makespan can be retrieved by starting from the final state and going backwards in the state graph.

Theorem 20. *Any prec-consistent instance of problem* $1|prec, r_j, d_j|C_{max}$ *can be solved in time* $\mathcal{O}(max(1, q^2 \cdot 4^q) \cdot N + n^2)$ *and space* $\mathcal{O}((q + log(D)) \cdot 2^q \cdot N)$.

Note that precedence relations only intervene during the precedence relation checks between each couple $((s_k, L_k), (s_{k+1}, L_{k+1}))$. So if our instance has no precedence constraints then each of these couples take time $\mathcal{O}(q \cdot log(q))$ to process instead of time $\mathcal{O}(q^2)$.

Corollary 21. *Any instance of problem* $1|r_j, d_j|C_{max}$ *can be solved in time* $\mathcal{O}(max(1, q \cdot log(q) \cdot 4^q) \cdot N + n^2)$ *and space* $\mathcal{O}((q + log(D)) \cdot 2^q \cdot N)$.

4.4. Space Optimization

Note that it is possible to reduce the space complexity of our algorithm to $\mathcal{O}((q + log(D)) \cdot 2^q)$ at the cost of a slightly increased time complexity. Indeed only the optimal makespan of the states associated to summit s_h must be remembered at any time in order to find the optimal makespan of all states associated to summit s_{h+1}. And when all the states associated to s_h have interacted with their successors, the space they take can be freed and used for the states associated to s_{h+2}. And, instead of computing proper sets and lingering sets in the preprocessing phase, each one of them can be determined in time $\mathcal{O}(n)$ whenever needed and forgotten once all the states associated to the corresponding job have interacted with all their successors. This algorithm variant computes the minimum makespan in space $\mathcal{O}((q + log(D)) \cdot 2^q)$ by only adding $\mathcal{O}(N \cdot n)$ operations.

Corollary 22. *The minimum makespan of any prec-consistent instance of problem* $1|prec, r_j, d_j|C_{max}$ *can be found in time* $\mathcal{O}(max(1, q^2 \cdot 4^q) \cdot N + n^2)$ *and space* $\mathcal{O}((q + log(D)) \cdot 2^q)$.

A schedule of corresponding makespan can then be found with $(N-1)$ additional iterations of the previous algorithm variant. Let $i \in [1, N-1]$. Right before the i^{th} iteration suppose that all states associated to summit s_{N-i+1} are in memory. Then at least one of them encodes part of a schedule of minimum makespan. Suppose that such a state S is annotated. We use an i^{th} iteration of the algorithm to retrieve all states associated to previous summit s_{N-i} and their relations to those associated to summit s_{N-i+1}. During this i^{th} iteration, we will identify at least one state S' linked to S with a relation which has a matching minimum makespan value. By comparing both states we can retrieve the set of jobs scheduled between summits s_{N-i} and s_{N-i+1} in some schedule of optimal makespan. And by annotating S' we can repeat this process with the states associated to the previous summit. So little by little a schedule of minimum makespan can be determined in a backwards manner.

Corollary 23. *Any prec-consistent instance of* $1|prec, r_j, d_j|C_{max}$ *can be solved in time* $\mathcal{O}(max(1, q^2 \cdot 4^q) \cdot N^2 + N \cdot n^2)$ *and space* $\mathcal{O}((q + log(D)) \cdot 2^q)$.

4.5. Minimizing L_{max}

In this subsection we consider instances of problem $1|prec, r_j|L_{max}$. Deadlines are replaced with due dates and we aim to minimize the maximum lateness of the instance. We run the dynamic programming algorithm for the makespan in order to get an initial value for L_{max}. This could be as far as $D' = [max_{i\in[1,n]}(r_i) + \sum_{i\in[1,n]} p_i]$ from the optimal value. Then we perform a binary search by solving a series of decision problems. We use deadlines instead of due dates and update them accordingly to the current step of the binary search. Each of these steps is solved by one run of our makespan algorithm.

Note that the value of parameter q is the same in every step of the search. Indeed when changing the L_{max} threshold all deadlines are shifted by the same amount of the time, so no relation of the form "$d_i < d_j$" is ever changed. This also means that *prec*-consistency, proper sets and lingering sets remain unchanged and thus do not need to be computed again at every step.

Corollary 24. *Any prec-consistent instance of* $1|prec, r_j|L_{max}$ *can be solved in time* $\mathcal{O}(max(1, q^2 \cdot 4^q) \cdot N \cdot log(D') + n^2)$ *and space* $\mathcal{O}((q + log(D')) \cdot 2^q \cdot N)$.

Corollary 25. *Any instance of problem* $1|r_j|L_{max}$ *can be solved in time* $\mathcal{O}(max(1, q \cdot log(q) \cdot 4^q) \cdot N \cdot log(D') + n^2)$ *and space* $\mathcal{O}((q + log(D')) \cdot 2^q \cdot N)$.

5. Negative Results on Identical Parallel Machines

In this section we show that decision problem $P|prec, p_j = 1, r_j, d_j|\star$ is para-NP-hard parameterized by q. In fact we show that the problem is still para-NP-hard if parameter q is paired with maximum deadline $D = \max_j d_j$, and it is W[2]-hard if q is paired with the width of the precedence graph w. Both proofs adapt previous reductions from Lenstra and Rinnooy-Kan [18] and Bodlaender and Fellows [3] respectively. This highlights the fact that unlike parameter pathwitdh μ, parameter q is not powerful enough to get a FPT algorithm on parallel processors.

5.1. With Parameter $q + D$

First we show that $P|prec, p_j = 1, r_j, d_j|\star$ is para-NP-hard parameterized by $q + D$.

Theorem 26. *$P|prec, p_j = 1, r_j, d_j|\star$ is NP-hard when $q + D = 3$.*

We adapt the reduction from CLIQUE proposed by Lenstra and Rinnooy-Kan in [18] to prove that $P|prec, p_j = 1|C_{max} \leq D = 3$ is strongly NP-hard. Let $G = (V, E)$ be a graph with $v = |V|$ and $e = |E|$. Let k be the clique size objective. Then Lenstra and Rinnooy-Kan defined an instance of problem $P|prec, p_j = 1|C_{max} \leq D$ the following way:

Definition 27. *[18] Set $a = k(k-1)/2$, $a' = e - a$ and $k' = v - k$. Then scheduling instance I has $m = (1+max(k, a+k', a'))$ machines, $n = 3m$ jobs and makespan threshold $D = 3$. Jobs are split into v vertex jobs $J_i, i \in V$, e edge jobs $J_{\{i,j\}}, \{i, j\} \in E$ and $3m - v - e$ fill jobs. Precedence relations in the form of a multipartite graph between fill jobs are used to ensure that in any schedule of makespan 3, $m - k$ of them will be scheduled at time 0, $m - (a + k')$ of them at time 1 and $m - a'$ of them at time 2. Plus for each edge $\{i, j\} \in E$ we have precedence from vertex jobs J_i, J_j to edge job $J_{\{i,j\}}$.*

Then if there is a clique of size k in G we schedule the corresponding k vertex jobs at time 0, the a edge jobs associated to this clique and the k' remaining vertex jobs at time 1, and the l' remaining edge jobs at time 2. The resulting schedule is feasible. Conversely if there is no clique of size k in G at most $a - 1$ edge jobs can be scheduled before time 2. So at least $a' + 1$ edge jobs have to be scheduled at time 2 which, combined with the $m - a'$ fill jobs set at that time, exceeds the number of machines. Thus there

is no feasible schedule for I. This concludes the reduction and the proof that $P|prec, p_j = 1|C_{max} < D$ is strongly NP-hard when $D = 3$.

Note that in any feasible schedule considered in the proof, vertex jobs are scheduled at time 0 or 1 while edge jobs are scheduled at time 1 or 2. So job time windows can be added accordingly without hindering feasibility.

Definition 28. *Instance I' of problem $P|prec, p_j = 1, r_j, d_j|\star$ is defined with help from the instance I of Definition 27. We have the set of jobs, we keep the precedence relations between vertex jobs and vertex jobs, but we remove the those between fill jobs. Instead we use time windows to set the fill jobs: $m-k$ of them with time window $[0, 1)$, $m-(a+k')$ of them with time window $[1, 2)$ and $m - a'$ of them with time window $[2, 3)$. All vertex jobs have time window $[0, 2)$ and all edge jobs have time window $[1, 3)$.*

The resulting instance I' is *prec*-consistent and equivalent to base instance I. Note that all time windows have length 1 or 2, so no time window can strictly include another on both ends. Thus in instance I' we have $q = 0$. This concludes the reduction and the proof that $P|prec, p_j = 1, r_j, d_j|\star$ is strongly NP-hard when $q + D = 3$.

5.2. With Parameter $q + w$

Here we consider parameter $q+w$ and we show that $P|prec, p_j = 1, r_j, d_j|\star$ is W[2]-hard.

Theorem 29. *$P|prec, p_j = 1, r_j, d_j|\star$ is W[2]-hard parameterized by $q + w$.*

We adapt the reduction from k-DOMINATING SET which was proposed by Bodlaender and Fellows in [3] to prove that $P|prec, p_j = 1|C_{max} \leq D$ parameterized by the number of machines is W[2] hard. Let $G = (V, E)$ be a graph with $V = \{u_0, \ldots, u_{v-1}\}$, $v = |V|$ and $e = |E|$. Let k be the dominating set size objective.

Before introducing the reduction, we first define some values and state their properties that will be used later. We define $c = 2v^2 + 1$. For any $0 \leq \alpha \leq (k + 1)v$ and $0 \leq l_1, l_2 \leq v - 1$ we define:

$$\lambda(\alpha, l_1, l_2) = (v - 1 + \alpha \cdot c + l_1 \cdot (2v) - l_2) \tag{1}$$

Then the following properties can be stated:

Lemma 30. $\forall l_1, l_2, l'_1, l'_2 \in \{0, \ldots, v - 1\}$:

- $\lambda(\alpha+1, l_1, l_2) - \lambda(\alpha, l_1', l_2') \geq v+2$ *for any* $\alpha \in \{0, \ldots, (k+1)v-1\}$,
- $\lambda(\alpha, l_1+1, l_2') - \lambda(\alpha, l_1, l_2) \geq v+1$ *for any* $\alpha \in \{0, \ldots, (k+1)v\}$.

Proof. For a fixed α, the minimum value of $\lambda(\alpha+1, l_1, l_2)$ is obtained with $l_1 = 0$ and $l_2 = v-1$, and the maximum value of $\lambda(\alpha, l_1', l_2')$ is obtained with $l_1' = v-1$ and $l_2' = 0$. So we get:

$$\begin{aligned}\lambda(\alpha+1, l_1, l_2) - \lambda(\alpha, l_1', l_2') &\geq [v-1+(\alpha+1)\cdot c - (v-1)] \\ &\quad - [v-1+\alpha\cdot c + (v-1)\cdot(2v)] \\ &= c - (2v^2 - v - 1) \\ &= v+2\end{aligned}$$

Similarly for fixed α, l_1:

$$\begin{aligned}\lambda(\alpha, l_1+1, l_2) - \lambda(\alpha, l_1, l_2') &\geq [v-1+\alpha\cdot c + (l_1+1)(2v) - (v-1)] \\ &\quad - [v-1+\alpha\cdot c + l_1(2v)] \\ &= 2v - (v-1) \\ &= v+1\end{aligned}$$

□

Then we define an instance I of $P|prec, p_j = 1, r_j, d_j|\star$ the following way:

Definition 31. *Instance* $I = \langle m, prec, r_j, d_j, D\rangle$ *has* $m = 2k+1$ *machines and makespan threshold* $D = \lambda((k+1)\cdot v, 1, v-1) = (k+1)\cdot v\cdot c + 2v$. *Let* $D' = D - v$.

We have $k+1$ *job chains of length* D'*: one floor chain* $(f_j)_j$ *and* k *selector chains* $(s_{i,j})_{i,j}$ *with* $1 \leq i \leq k$ *and* $0 \leq j \leq D'-1$. *The floor chain aims at ticking time with not much flexibility. The starting time of the first job in each selector chain will be used to select a node in the dominating set. The jobs constituting these chains are respectively called floor jobs* f_j *and selector jobs* $s_{i,j}$*, and they have time window* $[j, j+v)$.

On top of these chains we add the following jobs:

- ***floor gadgets:*** *one job* f_j' *"parallel" to each floor job* f_j *with* $j = \lambda(\alpha, l, 0)$ *or* $j = \lambda(\alpha, l, 0) + v$ *for some* $0 \leq l \leq v-1$ *and* $0 \leq \alpha \leq (k+1)v-1$. *Precedence relations* (f_{j-1}, f_j') *and* (f_j', f_{j+1}) *are in prec. Plus floor gadget* f_j' *has the same time window as floor job* f_j.
- ***selector gadgets:*** *for vertex indices* $0 \leq l_1, l_2 \leq v-1$ *such that* $l_1 \neq l_2$ *and* $(u_{l_1}, u_{l_2}) \notin E$*: two jobs* $s_{i,j}', s_{i,j+v}'$ *associated to each selector job* $s_{i,j}'$ *with* $1 \leq i \leq k$ *and* j *of the form* $\lambda(\alpha, l_1, l_2)$ *where* $1 \leq \alpha \leq (k+1)v$. *Precedence relations* $(s_{i,j-1}, s_{i,j}')$, $(s_{i,j}', s_{i,j+1})$, $(s_{i,j+v-1}, s_{i,j+v}')$

and $(s'_{i,j+v}, s_{i,j+v+1})$ *are in prec. Plus selector gadget* $s'_{i,j}$ *has the same time window* $[j, j+v)$ *as selector job* $s_{i,j}$*, while selector gadget* $s'_{i,j+v}$ *has the time window* $[j+v, j+2v)$*.*

Note that according to Lemma 30 the time windows of two different floor gadgets never intersect (i.e. $\lambda(\alpha, l+1, 0) > \lambda(\alpha, l, 0) + v$).

Lemma 32. *If graph* G *has a dominating set of size* k *then instance* I *has a feasible schedule.*

Proof. Given $\{u_{\gamma_1}, \ldots, u_{\gamma_k}\}$ a dominating set of graph G, we propose schedule τ where floor jobs/gadgets f_j, f'_j are scheduled at time j and for each $i \in [1, k]$ selector jobs/gadgets $s_{i,j}, s'_{i,j}$ are scheduled at time $\gamma_i + j$. Hence here the nodes in the dominating set define the starting time of the first job of the selector chains and the other jobs are performed in sequence without idle time.

We show that schedule τ is feasible. Clearly all precedence constraints are met in τ. Now we show that no more than $m = 2k+1$ jobs are scheduled at any time T. When T is *not* of the form $\lambda(\alpha, l, 0)$ nor $\lambda(\alpha, l, 0) + v$ where $0 \leq l \leq v-1$ and $0 \leq \alpha \leq (k+1)v-1$, there is at most one floor job, no floor gadget, at most k selector jobs and at most k selector gadgets scheduled at the same time, due to precedence constraints on the chains and with gadgets, so no issue there.

Now consider any time $T = \lambda(\alpha, l, 0)$, or $T = \lambda(\alpha, l, 0) + v$ of this form. Then the floor gadget f'_T is scheduled at that time. Plus for each $i \in [1, k]$ at most one selector gadget associated to the i^{th} selector chain can be scheduled at time T and, if it exists, it is $s'_{i,T-\gamma_i}$. We show that at least one these k selector gadget possibilities does not exist.

- If u_l is in the dominating set then $l = \gamma_{i_0}$ for some $i_0 \in [1, k]$. If $T = \lambda(\alpha, l, 0)$, then by definition the selector gadget $s'_{i_0,\lambda(\alpha,l,\gamma_{i_0})}$, which can be rewritten as $s'_{i_0,T-\gamma_{i_0}}$, cannot exist (since $l = \gamma_{i_0}$). if $T = \lambda(\alpha, l, 0) + v$, similarly, selector gadget $s'_{i_0,\lambda(\alpha,l,\gamma_{i_0})+v}$ cannot exist.

- Otherwise vertex u_l is adjacent to some vertex u_{γ_i} from the dominating set, with $i \in [1, k]$. If $T = \lambda(\alpha, l, 0)$, then selector gadget $s'_{i,T-\gamma_i} = s'_{i,\lambda(\alpha,l,\gamma_i)}$ where $(u_l, u_{\gamma_i}) \in E$, and thus it cannot exist according to the definition. If $T = \lambda(\alpha, l, 0) + v$, then selector gadget $s'_{i,T-\gamma_i} = s'_{i,\lambda(\alpha,l,\gamma_i)+v}$ would exist although $(u_l, u_{\gamma_i}) \in E$, a contradiction. So no selector gadget from the i^{th} chain is scheduled at time T.

Thus in both cases at most $k-1$ selector gadgets can be scheduled at the considered time T. So the number of machines is never exceeded anywhere and schedule τ is feasible. □

Conversely, given a feasible schedule of instance I, we show how to infer a dominating set of graph G. We would like to use the positions of the selector chain jobs in an interval of length c where all jobs are scheduled in an active manner - i.e. with no waiting time between consecutive jobs in any chain. The existence of such an interval was ensured by setting a sufficiently large time horizon.

Lemma 33. *If instance I has a feasible schedule then graph G has a dominating set of size k.*

Proof. Let τ be a feasible schedule of instance I. Given $\alpha \in [1, (k+1)v-1]$ the α^{th} range refers to time interval $[(-1+\alpha\cdot c), (-1+(\alpha+1)\cdot c))$, which corresponds to $[\lambda(\alpha,0,0)-v, \lambda(\alpha+1,0,0)-v)$. We say that the α^{th} range is *polluted* by a chain when there exists a time unit in this range to which no job from this chain is scheduled. For instance the α^{th} range is polluted by i^{th} selector chain, when there exists a time unit T in this range such that for all $0 \leq j \leq D'-1$ selector job $s_{i,j}$ is not scheduled at time T.

Since each of the $k+1$ chains in I have length $D' = D-v$, they can only pollute at most $v-1$ ranges each, and so at most $(k+1)v-(k+1)$ ranges total. can be polluted. Because the total number of ranges is $(k+1)v-1$, at least one of them is not polluted, say the α_0^{th} range $[(-1+\alpha_0\cdot c), (-1+(\alpha_0+1)\cdot c))$.

Due to precedence relations all floor gadgets f'_j (resp. selector gadgets $s'_{i,j}, s'_{i,j+v}$) executed in range α_0 are executed at the same time as their associated floor job f_j (resp. selector job $s_{i,j}$). Then by the feasibility of τ and the precedence relations, exactly one job from each chain is scheduled at time $T_0 = (-1+\alpha_0\cdot c)$. Let $j_0, \ldots, j_k$ be indices such that $f_{T_0-j_0}$ is the floor job scheduled at time T_0 and, given $i \in [1,k]$, s_{i,T_0-j_i} is the selector job of the i^{th} job which is scheduled at time T_0. Then we set $\gamma_i = (j_i - j_0)$ if $j_i \geq j_0$ and $(v + j_i - j_0)$ otherwise. Since j_i and j_0 can differ by at most $v-1$, we have $\gamma_i \in [0, v-1]$.

We claim that $\{u_{\gamma_1}, \ldots, u_{\gamma_k}\}$ is a dominating set of graph G. Let $l \in [0, v-1]$. We show that either vertex u_l is part of the proposed set, or it is adjacent to one of the elements in the proposed set. Consider time units $(T_0 + l\cdot(2v) + j_0 + v) = \lambda(\alpha_0, l, 0) + j_0$ and $\lambda(\alpha_0, l, 0) + j_0 - v$ which are both in range α_0. Among the jobs scheduled at these times we have

job floor $f_{\lambda(\alpha_0,l,0)}$, floor gadget $f'_{\lambda(\alpha_0,l,0)}$ (resp. job floor $f_{\lambda(\alpha_0,l-1,0)+v}$, floor gadget $f'_{\lambda(\alpha_0,l-1,0)+v}$) and k selector jobs, one per selector chain. So at most $k-1$ selector gadgets can be scheduled at each of these times due to the $2k+1$ machines. Consequently there is $i \in [1,k]$ such that selector gadget $s'_{i,\lambda(\alpha_0,l,0)+j_0-j_i}$ does not exist. And similarly there is $i' \in [1,k]$ such that $s'_{i',\lambda(\alpha_0,l,0)+j_0-j_{i'}-v}$ does not exist either.

- If $j_i \geq j_0$ then $s'_{i,\lambda(\alpha_0,l,0)+j_0-j_i} = s'_{i,\lambda(\alpha_0,l,\gamma_i)}$. This selector gadget does not exist, so either $l = \gamma_i$ or $(v_l, v_{\gamma_i}) \in E$.
- If $j_i < j_0$ then $s'_{i',\lambda(\alpha_0,l,0)+j_0-j_{i'}-v} = s'_{i',\lambda(\alpha_0,l,\gamma_{i'})}$. This selector gadget does not exist, so either $l = \gamma_{i'}$ or $(v_l, v_{\gamma_{i'}}) \in E$.

So in both cases we confirm that vertex v_l is either part of $\{u_{\gamma_1}, \ldots, u_{\gamma_k}\}$ or adjacent to one of these vertices. Therefore the proposed set is indeed a dominating set of size k in graph G. □

This concludes the reduction and the proof of Theorem 29. This confirms that contrary to pathwidth μ a FPT algorithm with respect to proper level q is unlikely to exist on problem $P|prec, p_j = 1, r_j, d_j|\star$.

6. Conclusion

In this paper we considered a new parameter on scheduling problems with release times and deadlines. This parameter, which was named proper level q, was defined as the maximum number of job time windows which can strictly include a job time window on both ends. Regarding problem $1|prec, r_j, d_j|C_{max}$ we proposed a FPT algorithm parameterized by q. Under some assumption of *prec*-consistency we established a new dominance rule, which we called the weak earliest deadline rule (wED), to reduce the space of candidate schedules. This led to the same summit-based dominance as in [9], which we proved to still work in the presence of precedence constraints under *prec*-consistency. Pairing this with a dynamic programming algorithm on a restricted selection of scheduling prefixes allowed us to solve the problem in FPT time. In contrast, we showed that decision problem $P|prec, p_j = 1, r_j, \bar{d}_j|\star$ is para-NP-hard parameterized by proper level q. Given that this problem is FPT with respect to pathwidth μ [23], this confirms both parameters to be distinct when it comes to studying scheduling problems.

Further research would take the proposed FPT algorithm as a baseline and look to optimize the dynamic programming phase. On the one hand the state graph is currently generated in a width-oriented way. So nearly the whole state graph must be generated before the first schedule candidates are found. On the other hand, a depth-oriented variant could find such schedule candidates quicker and use their makespan value as an upper bound for the rest of the state graph generation.

On another note, while we showed that parallel machine scheduling with unit-time jobs and general precedence constraints is para-NP-hard parameterized by the proper level, it is worth noting that the problem becomes polynomial-time solvable when precedence relations are restricted to chains - see e.g. [2]. On the contrary, the special case of tree-like precedence constraints is known to be strongly NP-hard [5]. Although we managed to establish a dominance rule specific to the case of outforests, the parameterized complexity of the latter with respect to the proper level is still open and should be studied. We believe that our dominance rule could serve as the basis for a FPT algorithm with respect to the proper level in the future.

References

[1] R. Baart, M. de Weerdt, and L. He. Single-machine scheduling with release times, deadlines, setup times, and rejection. *European Journal of Operational Research*, 291(2):629–639, 2021. Number: 2 Publisher: Elsevier.

[2] P. Baptiste, P. Brucker, S. Knust, and V. Timkovsky. Ten notes on equal-execution-time scheduling. *4OR*, 2:111–127, Jan. 2004.

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

[4] H. L. Bodlaender, C. Groenland, J. Nederlof, and C. M. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 193–204. IEEE, 2022.

[5] P. Brucker, M. Garey, and D. Johnson. Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness. *Math. Oper. Res.*, 2(3):275–284, 1977.

[6] M. Cieliebak, T. Erlebach, F. Hennecke, B. Weber, and P. Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In J.-J. Levy, E. W. Mayr, and J. C. Mitchell, editors, *Exploring New Frontiers of Theoretical Informatics*, pages 209–222, Boston, MA, 2004. Springer US. ISBN 978-1-4020-8141-5.

[7] R. Downey and M. Fellows. *Parameterized Complexity*. Springer, 1999.

[8] R. G. Downey, M. R. Fellows, and K. W. Regan. Descriptive complexity and the W hierarchy. In *Proof Complexity and Feasible Arithmetics*, pages 119–134, 1996.

[9] J. Erschler, G. Fontan, C. Mercé, and F. Roubellat. A new dominance concept in scheduling n jobs on a single machine with ready times and due dates. *Oper. Res.*, 31(1):114–127, 1983. doi: 10.1287/OPRE.31.1.114.

[10] J. Erschler, G. Fontan, and C. Merce. Un nouveau concept de dominance pour l'ordonnancement de travaux sur une machine. *RAIRO-Operations Research*, 19(1):15–26, 1985.

[11] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 1998.

[12] M. R. Garey and D. S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM Journal on Computing*, 6(3):416–426, Sept. 1977. ISSN 0097-5397, 1095-7111.

[13] C. Hanen and A. Munier Kordon. Fixed-parameter tractability of scheduling dependent typed tasks subject to release times and deadlines. *Journal of Scheduling*, pages 1–15, 2023.

[14] C. Hanen, M. Mallem, and A. Munier-Kordon. Parameterized complexity of single-machine scheduling with precedence, release dates and deadlines. In *15th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP)*, 2022.

[15] K. Heeger and D. Hermelin. Minimizing the weighted number of tardy jobs is W[1]-hard. *arXiv preprint arXiv:2401.01740*, 2024.

[16] K. Jansen, K. Kahler, and E. Zwanger. Exact and approximate high-multiplicity scheduling on identical machines. *arXiv preprint arXiv:2404.17274*, 2024.

[17] A. M. Kordon and N. Tang. A fixed-parameter algorithm for scheduling unit dependent tasks with unit communication delays. In *European Conference on Parallel Processing*, pages 105–119. Springer, 2021.

[18] J. Lenstra and A. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Oper. Res.*, 26(1):22–35, 1978.

[19] J. Lenstra, A. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Ann. of Discrete Math.*, 1:343–362, 1977.

[20] M. Mallem. *Parameterized Complexity and new Efficient Enumerative Schemes for RCPSP*. PhD thesis, Sorbonne Université, 2024.

[21] M. Mallem, C. Hanen, and A. Munier-Kordon. Parameterized complexity of a parallel machine scheduling problem. In *17th International Symposium on Parameterized and Exact Computation (IPEC)*, 2022.

[22] M. Mallem, C. Hanen, and A. Munier-Kordon. A new structural parameter on single machine scheduling with release dates and deadlines. In A. Basu, A. R. Mahjoub, and J. J. Salazar González, editors, *Combinatorial Optimization*, pages 205–219, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-60924-4.

[23] A. Munier Kordon. A fixed-parameter algorithm for scheduling unit dependent tasks on parallel machines with time windows. *Discrete Applied Mathematics*, 290:1–6, 2021.

[24] A. Proskurowski and J. A. Telle. Classes of graphs with restricted interval models. *Discrete Mathematics & Theoretical Computer Science*, 3, 1999.

[25] B. Simons. A fast algorithm for single processor scheduling. In *19th Annual Symposium on Foundations of Computer Science (SFCS 1978)*, pages 246–252, Oct. 1978. ISSN: 0272-5428.

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

[27] I. Tarhan, J. Carlier, C. Hanen, A. Jouglet, and A. Munier Kordon. Parameterized analysis of a dynamic programming algorithm for a parallel machine scheduling problem. In *European Conference on Parallel Processing*, pages 139–153. Springer, 2023.

[28] R. van Bevern, R. Bredereck, L. Bulteau, C. Komusiewicz, N. Talmon, and G. J. Woeginger. Precedence-constrained scheduling problems parameterized by partial order width. In Y. Kochetov, M. Khachay, V. Beresnev, E. Nurminski, and P. Pardalos, editors, *Discrete Optimization and Operations Research*, pages 105–120, Cham, 2016. Springer International Publishing. ISBN 978-3-319-44914-2.

[29] R. van Bevern, R. Niedermeier, and O. Suchỳ. A parameterized complexity view on non-preemptively scheduling interval-constrained jobs: few machines, small looseness, and small slack. *Journal of Scheduling*, 20:255–265, 2017. doi: https://doi.org/10.1007/s10951-016-0478-9.