A new structural parameter on single machine scheduling with release dates and deadlines

Maher Mallem[1][0000−0001−5654−1090], Claire Hanen[1,2][0000−0003−2482−5042], and Alix Munier-Kordon[1][0000−0002−2170−6366]

[1] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
{Maher.Mallem,Claire.Hanen,Alix.Munier}@lip6.fr
http://www.lip6.fr
[2] UPL, Université Paris Nanterre, F-92000 Nanterre, France

Abstract. In this paper we study the single machine scheduling problem with release dates, deadlines and precedence relations where the objective is to minimize the makespan. This is a well-known strongly NP-hard scheduling problem [20]. We analyze the problem from the parameterized complexity point of view. We propose parameter q which is the maximum number of time windows $[r_j, d_j)$ that can strictly include a time window $[r_i, d_i)$ on both ends. We show that problems $1|prec, r_j, d_j|C_{max}$ and $1|prec, r_j|L_{max}$ are fixed-parameter tractable parameterized by q. We use a dynamic programming approach and define a new dominance rule, which we call the weak earliest deadline rule. This rule narrows down the number of relevant scheduling prefixes enough to complete the search via a fixed-parameter tractable number of dynamic programming states.

Keywords: parameterized complexity · scheduling · single machine

1 Introduction

Many scheduling problems, even seemingly basic ones such as $1|r_j|L_{max}$, have been proved strongly NP-hard [20] over the years. Upon reaching NP-hardness this quickly, it becomes difficult to establish anything deeper about these scheduling problems within the scope of classical complexity theory. To answer this, parameterized complexity theory gives additional tools for a refined analysis of such hard scheduling problems. Given a parameter k and denoting n the input size, a problem is called $fixed-parameter\ tractable$ (FPT) parameterized by k if it can be solved in time $\mathcal{O}(f(k)\cdot poly(n))$ with f an arbitrary computable function [7]. The idea is to identify k as the limiting property and give a polynomial time algorithm for all instances with a bounded value of k. 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 [8,12].

While parameterized complexity theory has been successfully applied to a lot of computer science areas, it has started to be studied extensively on scheduling only quite recently. One of the few older results was from [3] and showed

that $P|prec, p_j = 1|C_{max}$ is W[2]-hard parameterized by the number m of machines. When time windows are involved, two parameters have been considered with great success. First slack σ - which is defined as the maximum number of starting times allowed for any job - was successfully used in the context of single machine scheduling with setup times, and rejection [1]. 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 were proved FPT parameterized by μ [1,16,18,21], for some of them even in the context of parallel machines.

Despite their success, such parameters can overestimate the instance difficulty at times. Indeed a single job with large time window length is enough to get an unbounded slack. In the case of pathwidth μ we give an example in Figure 1 with two instances of decision problem $1|r_j, d_j|\star$. In the left instance all jobs have the same release date and deadline, so this is an easy particular case which can be solved in linear time by adding up the processing times. Yet this instance has the same pathwidth as the right one, which simulates a PARTITION problem by using a fill job right in the middle. Note that both instances yield the same time window overlap graph - the complete graph K_n. So in order to distinguish them one would have to track time window interactions beyond their overlaps.

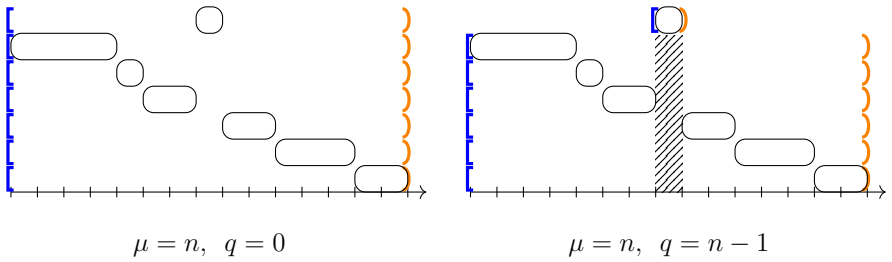


Fig. 1. Two instances of $1|r_j, d_j|\star$ with the same pathwidth μ but different q-proper levels q.

Instead of time window overlaps like with pathwidth μ, we propose to track whenever a time window $[r_j, d_j)$ strictly includes another time window $[r_i, d_i)$ on both ends. When this is the case we say that job j *surrounds* job i. Such time window interactions were considered in the past [10,11,13], albeit never in the context of parameterized complexity. We define parameter q as the maximum number of jobs j which can surround a job i. This definition of q is inspired by Proskurowski and Telle in [23] where the q-proper level of an interval graph was defined. This is applied to the interval graph given by the job time windows.

Going back to Figure 1 the q-proper level effectively distinguishes between both instances: a low value for the easy instance on the left and a high value for the hard instance on the right. As the right instance suggests parameter q can be interpreted as the maximum size of any PARTITION subproblem which is encoded in the scheduling instance.

Now it is worth noting that scheduling rules have already been successfully used in some subproblems of $1|prec, r_j, d_j|C_{max}$ [19]. One of the most well-known examples is the *earliest deadline rule* [25] - which will be denoted (ED). Given an instance of problem $1|d_j|C_{max}$ the (ED) rule says the following: schedule jobs actively in nondecreasing order of their deadline. This gives the minimum makespan in time $\mathcal{O}(n \cdot log(n))$. The same can be achieved on problem $1|r_j|C_{max}$ with the *earliest release date rule* [2] - which will be denoted (ERD).

However given an instance of $1|prec, r_j, d_j|C_{max}$ such strategies become more difficult to pull off. Indeed both (ED) and (ERD) rules can still work but only when release dates and deadlines follow the same order. If there is at least one job couple (i, j) such that $r_j < r_i$ and $d_i < d_j$ - i.e. job j *surrounds* job i - then both rules become wrong. Fittingly when this happens we have $q \geq 1$.

In response we propose a variant of the (ED) rule. We call it the *weak earliest deadline rule* and denote it (wED). It says the following: schedule jobs in nondecreasing order of their deadline, except when some job surrounds other jobs. Given each job $i \in [\![n]\!]$ the only allowed exceptions are the jobs j which either surround i or surround a job k with a deadline $d_k \leq d_i$ and scheduled between j and i. For each job i we show that under the (wED) rule the number of such jobs j is bounded by our parameter q. This will be enough to get a number of dynamic programming states of the form $f(q) \cdot poly(n)$.

The paper is organized as follows: in Section 2 we give the necessary preliminaries to our approach. In Section 3 we define the (wED) rule on problem $1|prec, r_j, d_j|C_{max}$ and characterize the properties of the schedules which follow this dominance rule. This leads to the summit-based scheduling dominance originally proposed in [11] on problem $1|r_j, d_j|C_{max}$, which we prove to work with problem $1|prec, r_j, d_j|C_{max}$ too. Then in Section 4 we present a dynamic programming algorithm 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. Finally we discuss how this approach can be adapted to criterion L_{max}.

2 Preliminaries

We consider a set of n jobs $\mathcal{J} = \{1, \ldots, n\}$ to be scheduled on a single machine. The 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 $r_j, d_j \in \mathbb{N}$ 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 $\sigma : \mathcal{J} \rightarrow \mathbb{N}$ 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. $C_{\max}(\sigma) = \max_{i \in \mathcal{J}}[\sigma(i) + p_i]$. This problem is usually denoted $1|prec, r_j, d_j|C_{max}$.

We denote $[\![n]\!]$ the integer interval $[1, n]$ and the set of jobs $\mathcal{J}$. Since we have both time windows constraints and precedence relations in our problem, we must ensure that the time constraints are compatible with the partial order $\rightarrow$.

Definition 1. *An instance* $I = \langle \mathcal{J}, prec, p_j, r_j, d_j \rangle$ *of* $1|prec, r_j, d_j|C_{max}$ *is called prec-consistent when:*
$\forall (i,j) \in [\![n]\!]^2,\ (i \rightarrow j) \implies [(r_i + p_i \leq r_j) \wedge (d_i \leq d_j - p_j)]$.

Given an instance of $1|prec, r_j, d_j|C_{max}$ which is not *prec*-consistent, 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. Without it one can build artificial counterexamples like the one in Figure 2.

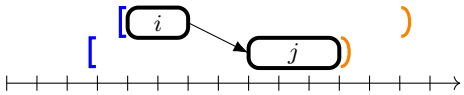


Fig. 2. A non *prec*-consistent counterexample to the (wED) rule.

Consider a *prec*-consistent instance. For the remainder of the paper we suppose that the jobs have been renamed in $[\![n]\!]$ in lexicographic nondecreasing order of (deadline, release date). So when we write "$i < j$" 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:

Definition 2. *(Proper set* Π_i*) Let* $i \in [\![n]\!]$*. We say that a job* $j \in [\![n]\!]$ *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* $\Pi_i = \{j \in [\![n]\!], j$ *surrounds* $i\}$*.*
(q-proper level) We define parameter $q = max_{i \in [\![n]\!]} |\Pi_i|$*. In other words* q *is the maximum number of jobs* j *which can surround a job* i*.*

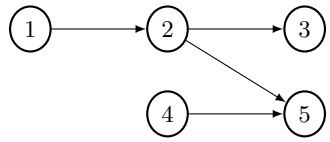
Finally like in [10] we highlight the following subset of jobs:

Definition 3. *We say that a job* $j \in [\![n]\!]$ *is a summit when* j *surrounds no job. In other words: for all jobs* i *in* $[\![n]\!]$ 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 3. 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 1. (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* $j \in \Pi_{s_k}$*.*

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



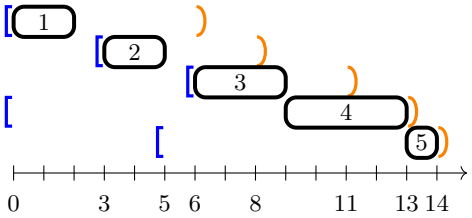


Fig. 3. An example with five jobs and q-proper level two.

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 The (wED) Rule

In this section we define the weak earliest deadline (wED) rule and establish the dominance of schedules following this rule. Then we show that such optimal schedules can be further modified to get optimal schedules with a stronger structure based on the summits of the instance.

3.1 Definition and Dominance

First we define the (wED) rule formally:

Definition 4. *A feasible schedule σ on an instance I follows the (wED) rule when: $\forall (i,j) \in [\![n]\!]^2$, if $i < j$ then either i is scheduled before j (i.e. $\sigma(i) < \sigma(j)$) or j surrounds i (i.e. $j \in \Pi_i$) or there exists $h < i$ such that $\sigma(j) < \sigma(h) < \sigma(i)$.*

On problem $1|prec, r_j, d_j|C_{max}$ we show that the schedules following the (wED) rule are dominant.

Lemma 2. *Given a feasible schedule σ on a prec-consistent instance I 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 $i < j$, $j \notin \Pi_i$ 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. □

3.2 A Summit-based Decomposition

Upon closer inspection the (wED)-following schedules have these properties, which will serve as the basis of our dynamic programming approach in the next section:

Lemma 3. *Any schedule on a prec-consistent instance I of $1|prec, r_j, d_j|C_{max}$ which is feasible and 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 1 (i) either h is a summit or it surrounds a summit. In the first case Lemma 1 (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 1 (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 1 (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 1 (ii) j also surrounds s_N.
b. If $k \in [2, N]$: again job j is not a summit so by Lemma 1 (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 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. We do so by fixing the job order in each T_k. As a result we get the dominance proposed in [11] for problem $1|r_j, d_j|C_{max}$, except we must deal with precedence constraints and restrict ourselves to *prec*-consistent instances. Such schedules will be called *summit-ordered*.

Definition 5. *A schedule on a prec-consistent instance I of $1|prec, r_j, d_j|C_{max}$ 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 3 are satisfied and moreover 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 3. 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. We conclude this section with one final dominance result:

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

Proof. Let σ be a feasible schedule. From Lemmas 2 and 3 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 3 (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 non-decreasing 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. □

4 The Algorithm

In this section we propose 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 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. By the definition of *summit-ordered* schedules we only need to recall the jobs greater than each summit s_k and scheduled before them. To this purpose we define the following set:

Definition 6. *(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 4. *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 5 (i) $s_1, \dots, s_k$ 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 5 (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:

Lemma 5. *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 $(i_1, L_1) \to (\dots) \to (i_N, L_N)$ 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. $C_N = \{j \in \Pi_{s_N}, j \notin L_N\}$.

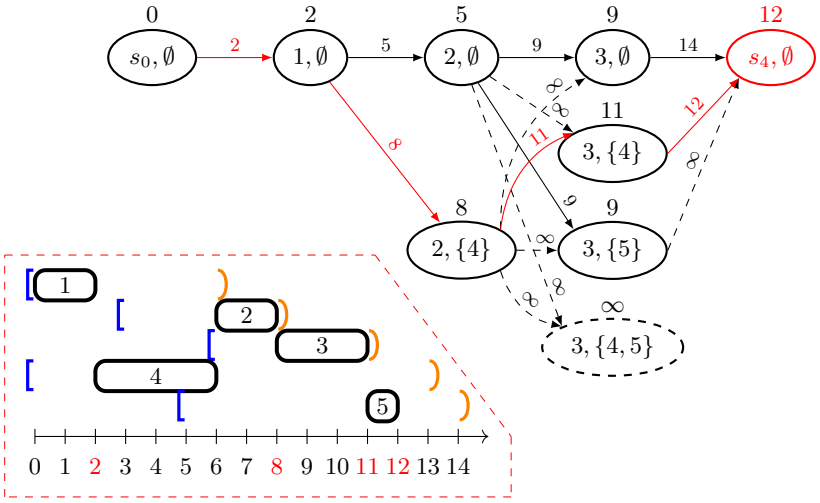


Fig. 4. State graph associated to the example given in Figure 3. The value of each arc is the makespan of the scheduling candidate computed by extend(k, L_k, L_{k+1}). The red path corresponds to active schedule "14235", which gives the optimal makespan.

Proof. This directly comes from Definition 5 (iv) and Lemma 4. □

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 $T_1 s_1 \dots T_k s_k$:

Lemma 6. *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 $T'_1 s_1 \dots T'_k s_k$ has optimal makespan among the 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 1 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 5. For 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 3 example is given in Figure 4. 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 $(s_0, \emptyset) \to (1, \emptyset) \to (2, \{4\}) \to (3, \{4\}) \to (s_4, \emptyset)$.

Algorithm 1 main()

preprocessing() ▷ Check *prec*-consistency. Compute proper sets and lingering sets.
$r_{s_0}, p_{s_0}, d_{s_0}, p_{s_{N+1}} \leftarrow 0$; $r_{s_{N+1}} \leftarrow max_{i\in[\![n]\!]}(r_i)$; $d_{s_{N+1}} \leftarrow \infty$
$\Pi_{s_0}, \Lambda_0, \Pi_{s_{N+1}}, \Lambda_{N+1} \leftarrow \emptyset$
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 $T[N+1, \emptyset]$

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 5.
$C \leftarrow list(\{j \in \Pi_{s_k}, j \notin L_k \cup \Pi_{s_{k+1}}\})$ ▷ C_k
sort(C, nondecreasing d_j)
$A \leftarrow list(\{j \in \Pi_{s_k} \cap \Pi_{s_{k+1}} \cap L_{k+1}, j \notin L_k\})$ ▷ A_{k+1}
$B \leftarrow list(\{j \in \Pi_{s_{k+1}} \cap L_{k+1}, j \notin \Pi_{s_k} \cup L_k\})$ ▷ B_{k+1}
sort(B, nondecreasing r_j)
$add \leftarrow C \cdot A \cdot B \cdot [s_{k+1}]$
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 $Cmax$

4.2 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 7. *$\forall k \in [1, N]$, $\Lambda_k = \Pi_{s_k}$.*

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

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 1 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 5 they can be retrieved from sets L_k and L_{k+1} and appended accordingly to Definition 5 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 5 (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 1 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 1 this requires space $\mathcal{O}(q + log(D))$ where $D = max_{i \in [\![n]\!]}(d_i)$. 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 1. *$1|prec, r_j, d_j|C_{max}$ 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 checks between each couple $((s_k, L_k), (s_{k+1}, L_{k+1}))$. So if our instance has no precedence relations then each of these couples take time $\mathcal{O}(q \cdot log(q))$ to process instead of time $\mathcal{O}(q^2)$.

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

Finally note that if only the optimal value is wanted and not a schedule, it is possible to reduce the space complexity to $\mathcal{O}((q + log(D)) \cdot 2^q)$. 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}. Finally instead of a preprocessing phase proper sets and lingering sets are only computed whenever needed and forgotten once all the states associated to the corresponding job have interacted with all their successors.

4.4 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 $D' = [max_{i\in[\![n]\!]}(r_i) + \sum_{i\in[\![n]\!]} p_i]$ 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 3. *Any instance I of problem $1|prec, r_j|L_{max}$ can be solved in time $\mathcal{O}(max(1, q^2 \cdot 4^q) \cdot N \cdot log(D') + n^2)$ and space $\mathcal{O}((q + log(D')) \cdot 2^q \cdot N)$.*

5 Conclusion

In this paper we showed that problems $1|prec, r_j, d_j|C_{max}$ and $1|prec, r_j|L_{max}$ are FPT parameterized by the q-proper level of the interval graph given by the job time windows. We used 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 [11], which we proved to still work in the presence of precedence constraints. Pairing this with a dynamic programming algorithm on a restricted selection of scheduling prefixes allowed us to solve both problems in FPT time.

Further research would use a similar approach beyond the single machine case, like a generalization to parallel machines. Allowing arbitrary processing times on two machines already leads to para-NP-hardness with pathwidth μ [16], which is a stronger parameter than q. So the parallel machine problem with unit-time processing times, precedence constraints and time windows could be the next step with parameter q.

References

1. Baart, R., de Weerdt, M., He, L.: Single-machine scheduling with release times, deadlines, setup times, and rejection. European Journal of Operational Research **291**(2), 629–639 (2021)
2. Baker, K.R.: The effects of input control in a simple scheduling model. Journal of Operations Management **4**(2), 99–112 (1984)
3. Bodlaender, H.L., Fellows, M.R.: W[2]-hardness of precedence constrained k-processor scheduling. Operations Research Letters **18**(2), 93–97 (1995)
4. Bodlaender, H.L., van der Wegen, M.: Parameterized complexity of scheduling chains of jobs with delays. In: Cao, Y., Pilipczuk, M. (eds.) 15th International Symposium on Parameterized and Exact Computation (IPEC), December 14-18, 2020, Hong Kong, China (Virtual Conference). LIPIcs, vol. 180, pp. 4:1–4:15 (2020)
5. Bredereck, R., Bulteau, L., Komusiewicz, C., Talmon, N., van Bevern, R., Woeginger, G.J.: Precedence-constrained scheduling problems parameterized by partial order width. In: International conference on discrete optimization and operations research. pp. 105–120. Springer (2016)
6. Chu, C., Proth, J.M.: Sequencing with chain structured precedence constraints and minimal and maximal separation times. In: Proceedings of the fourth international conference on computer integrated manufacturing and automation technology. pp. 333–338. IEEE (1994)
7. Downey, R., Fellows, M.: Parameterized complexity. Springer (1999)
8. Downey, R.G., Fellows, M.R., Regan, K.W.: Descriptive complexity and the W hierarchy. In: Proof Complexity and Feasible Arithmetics. pp. 119–134 (1996)
9. Du, J., Leung, J.Y., Young, G.H.: Scheduling chain-structured tasks to minimize makespan and mean flow time. Information and Computation **92**, 219–236 (1991)
10. Erschler, J., Fontan, G., Merce, C.: Un nouveau concept de dominance pour l'ordonnancement de travaux sur une machine. RAIRO-Operations Research **19**(1), 15–26 (1985)
11. Erschler, J., Fontan, G., Mercé, C., Roubellat, F.: 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). https://doi.org/10.1287/OPRE.31.1.114
12. Flum, J., Grohe, M.: Parameterized complexity theory. Springer (1998)
13. Gordon, V.S., Werner, F., Yanushkevich, O.: Single machine preemptive scheduling to minimize the weighted number of late jobs with deadlines and nested release/due date intervals. RAIRO-Operations Research **35**(1), 71–83 (2001)
14. Grigoreva, N.: Single machine scheduling with precedence constrains, release and delivery times. In: Information Systems Architecture and Technology: Proceedings of 40th Anniversary International Conference on Information Systems Architecture and Technology–ISAT 2019: Part III. pp. 188–198. Springer (2020)
15. Hall, L.A., Shmoys, D.B.: Jackson's rule for single-machine scheduling: Making a good heuristic better. Mathematics of Operations Research **17**(1), 22–35 (1992)
16. Hanen, C., Munier Kordon, A.: Fixed-parameter tractability of scheduling dependent typed tasks subject to release times and deadlines. Journal of Scheduling pp. 1–15 (2023)
17. Hermelin, D., Karhi, S., Pinedo, M., Shabtay, D.: New algorithms for minimizing the weighted number of tardy jobs on a single machine. Annals of Operations Research **298**(1), 271–287 (2021)
18. Kordon, A.M., Tang, N.: A fixed-parameter algorithm for scheduling unit dependent tasks with unit communication delays. In: European Conference on Parallel Processing. pp. 105–119. Springer (2021)

19. Lawler, E.L.: Optimal sequencing of a single machine subject to precedence constraints. Management science **19**(5), 544–546 (1973)
20. Lenstra, J., Rinnooy Kan, A., Brucker, P.: Complexity of machine scheduling problems. Ann. of Discrete Math. **1**, 343–362 (1977)
21. Mallem, M., Hanen, C., Munier-Kordon, A.: Parameterized complexity of a parallel machine scheduling problem. In: 17th International Symposium on Parameterized and Exact Computation (IPEC) (2022)
22. Mnich, M., Wiese, A.: Scheduling and fixed-parameter tractability. Mathematical Programming **154**(1), 533–562 (2015)
23. Proskurowski, A., Telle, J.A.: Classes of graphs with restricted interval models. Discrete Mathematics & Theoretical Computer Science **3** (1999)
24. Sourd, F., Nuijten, W.: Scheduling with tails and deadlines. Journal of Scheduling **4**(2), 105–121 (2001)
25. Ware, E.B.: Job shop simulation on the ibm 704. In: Preprints of papers presented at the 14th national meeting of the Association for Computing Machinery (1959)