Kernelization Algorithms on Single Machine Scheduling with Time Windows

Maher Mallem[1,*], Claire Hanen, Alix Munier-Kordon

[a] *Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, UMR 5668, Lyon cedex 07, 69342, France*
[b] *Université Paris Nanterre, Nanterre, 92000, France*
[c] *Sorbonne Université, CNRS, LIP6, Paris, 75005, France*
[d] *Sorbonne Université, CNRS, LIP6, Paris, 75005, France*

Abstract

Over the past few years parameterized complexity has successfully been considered on scheduling problems, with many fixed-parameter tractable algorithms established on a variety of parameters. Knowing this, it is standard to look for kernels and associated kernelization algorithms in order to preprocess scheduling instances and improve the runtime of parameterized algorithms. However, multiple parameterized scheduling problems still lack kernel characterizations, especially when the number of job types is not bounded by the parameter in any way. In this paper, we study the kernelization of a single machine scheduling problem with precedence constraints and job time windows with respect to three graph parameters: the proper level, the pathwidth and the vertex cover of the compatibility graph induced by the job time window overlaps. While this problem is known to be fixed-parameter tractable with respect to the pathwidth and the proper level, we show that a polynomial kernel is unlikely to be built with these parameters. Nevertheless we provide a polynomial kernel with respect to the vertex cover. Based on this polynomial kernel, we propose kernels of exponential size with respect to the pathwidth and the proper level. This paves the way to kernel ideas for other scheduling problems with known fixed-parameter tractable algorithms

[*]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.

in the hopes of improving their runtime in practice.

Keywords: scheduling, kernel, single machine, fixed-parameter tractability, vertex cover

---

1. Introduction

Parameterized complexity of scheduling problems has drawn increasing attention in the last ten years. Several fixed-parameter tractable (FPT) algorithms have been proposed, albeit most of the problems have been proven somewhere higher in the W-hierarchy. In this paper we consider a single machine scheduling problem with job time windows and precedence constraints. This decision problem is denoted by $1|prec, r_j, \bar{d}_j|\star$ in Graham's notation. It has been proven NP-hard in the strong sense in the 70's (Lenstra et al. 1977), even without precedence constraints.

Single machine scheduling has been studied from a parameterized point of view by several authors with two main tools. Particular structures of linear programs were considered in (Hermelin et al. 2021, Knop et al. 2019, Mnich and Wiese 2015) to design FPT algorithms for scheduling problems where the parameter is closely related to the number of job types. Within a type of jobs, all jobs share the same properties. Several other parameters have been investigated, this time using dynamic programming to obtain FPT algorithms. For single machine problems, let us mention the pathwidth of the interval graph induced by the time windows (Hanen et al. 2022), the proper level of the interval graph (Mallem et al. 2024), or the width of the precedence graph combined to the maximum allowed lag (van Bevern et al. 2016). Unfortunately these parameters are not sufficient to derive FPT algorithms in the presence of precedence delays (Bodlaender and van der Wegen 2020, Hanen et al. 2022) or parallel processors (Hanen and Munier Kordon 2023). The maximum processing time of a job also leads to a negative parameterized result since the single machine problem is NP-complete with two different fixed values of processing times (Elffers and de Weerdt 2014).

Now it is well known that whenever a problem $\mathcal{Q}$ is FPT with respect to some parameter k, there is a corresponding kernelization algorithm with respect to k (Cai et al. 1997, Flum and Grohe 1998). Given a parameterized instance I of $\mathcal{Q}$ with parameter value $k(I)$, a kernelization is a polynomial-time algorithm which outputs an instance I' of $\mathcal{Q}$ with parameter value $k(I')$ such that I' is equivalent to I and the size of the parameterized instance

$|I'| + k(I')$ is bounded by $f(k(I))$ for some computable function f. Such an instance I' is called a kernel. The kernel is polynomial if f is a polynomial function. There is a general way to infer a kernelization from any FPT algorithm (Flum and Grohe 1998). However, having a kernelization which actively builds the kernel from the input generally gives more information on the nature and shape of the set of kernels in problem $\mathcal{Q}$.

While such kernelizations have already been found for some scheduling problems (Brinkop and Jansen 2022, Jansen et al. 2025, Knop and Koutecký 2022), they were done in the context of high-multiplicity scheduling with the number of job types among the parameters. This avoids dealing with an unbounded number of job types. Considering other structural parameters, we need to reduce the instance size - i.e. both the number of jobs and the data values - to a function of the parameter. To our knowledge, no such kernelization algorithm has been produced in the literature.

In this paper, as a first insight on the reduction techniques, we focus on three parameters related to the interval graph induced by the time windows: its pathwidth denoted by pw, its proper level denoted by q (i.e. the maximum number of time windows which can strictly include a time window on both ends) and its vertex cover number denoted by vc. In Section 2 we give base definitions and relate our three parameters. From this relation and a result from (Hanen et al. 2022) we derive that the single machine scheduling problem with precedence relations and job time windows is FPT with respect to vc. In Section 3 we propose a first polynomial kernel for parameters vc and the maximum processing time of a job $p_{\max}$ combined. Then in Section 4 we provide an additional reduction rule allowing for a second polynomial kernel with parameter vc only. Finally in Section 5 we consider kernelization algorithms with parameters q or pw. In contrast with parameter vc we show that the problem cannot have a polynomial kernel for either parameter under usual parameterized complexity assumptions. In response we propose kernels of exponential size with respect to either parameter, and consider pairing them with a parameter which bounds the number of distinct release date and deadline values in the instance.

2. Problem and Parameter Definitions

In this section we define accurately the decision problem and present the parameters that we use.

Definition 1. *An instance* $I = \langle prec, p_j, r_j, \bar{d}_j \rangle$ *of problem* $1|prec, r_j, \bar{d}_j|\star$ *is defined by:*

1. *a set of* n *jobs* $\mathcal{J}$*. Each job* i *is characterized by a release date* r_i*, a deadline* $\bar{d}_i$ *and a processing time* p_i*,*
2. *a set of precedence constraints in the form of a directed acyclic graph* $H = (\mathcal{J}, A)$*. An arc* (i, j) *in* A *implies that job* j *cannot start before the end of job* i*. This relation is also denoted by* $i \to j$*.*

Definition 2. *A schedule of an instance defines for each job* i *a starting time* $s_i \in [r_i, \bar{d}_i)$ *so that at most one job is processed at each time: i.e. for any two jobs* i, j *either* $s_j \geq s_i + p_i$ *or* $s_i \geq s_j + p_j$*. Moreover, for any arc* $(i, j) \in A$*, we have* $s_i + p_i \leq s_j$*.*

Without loss of generality we assume that the precedence constraints are consistent with the time windows, i.e. if $(i, j) \in A$ then $r_j \geq r_i + p_i$ and $\bar{d}_i \leq \bar{d}_j - p_j$. Such instances will be called *prec*-consistent. We consider the decision problem of checking the existence of a feasible schedule. Notice that by using binary search, a FPT algorithm for this problem provides a FPT algorithm for the optimization problems minimizing makespan (C_{max}) and maximum lateness (L_{max}).

Definition 3. *For an instance of the problem we define the compatibility graph* $G_I = (\mathcal{J}, E)$*. We have undirected edge* $\{i, j\}$ *in* E *if and only if the time windows of jobs* i *and* j *overlap, i.e.* $[r_i, \bar{d}_i) \cap [r_j, \bar{d}_j) \neq \emptyset$*.*

We use the example depicted in Figure 1 throughout our paper to illustrate our algorithms. We consider a set of 23 jobs indexed in $\{1, \ldots, 23\}$:

- For $i \in \{1, \ldots, 20\}$, $r_i = 10 \cdot (i - 1)$ and $\bar{d}_i = 10 \cdot i$. We have $p_i = 2$ if $i = 1$ or $i = 20$ and $p_i = 5$ otherwise.
- Jobs $21, 22, 23$ have processing time 10, release times and deadlines $r_{21} = r_{22} = 0, \bar{d}_{21} = \bar{d}_{22} = 100$, and $r_{23} = 90, \bar{d}_{23} = 200$.
- We have a unique precedence constraint $(22, 23)$.

Now for this example the graph G_I is composed of a triangle $21-22-23$, plus edges between $21, 22$ and all jobs $\{1, \ldots, 10\}$, and edges between 23 and all jobs $\{10, \ldots, 20\}$. Observe that jobs $\{1, \ldots, 20\}$ define an independent set.
In the literature, several studied parameters related to this compatibility

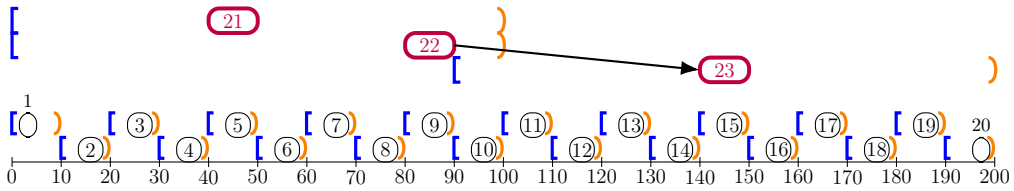


Figure 1: An example with 23 jobs.

graph have been investigated like the pathwidth and the proper level (Mallem et al. 2024). Note that in interval graphs the pathwidth corresponds to the maximum clique size (minus one) (Bodlaender and Möhring 1993). So the pathwidth of compatibility graph G_I can be defined the following way.

Definition 4. *The pathwidth pw of compatibility graph G_I is defined as the maximum number of overlapping time windows at any given time (minus one).*

The proper level q of compatibility graph G_I is defined as the maximum number of time windows $[r_j, \bar{d}_j)$ which can strictly include a time window $[r_i, \bar{d}_i)$ on both ends. In other words: $q = max_{i \in \mathcal{J}}(|\{j \in \mathcal{J}, r_j < r_i \wedge \bar{d}_i < \bar{d}_j\}|)$.

We consider here the minimum vertex cover of the compatibility graph as another parameter.

Definition 5. *A vertex cover of a graph $G = (V, E)$ is a subset of V which covers all edges in E - i.e. given any edge in E at least one of its nodes is in the vertex cover. $vc(G)$ is defined as the minimum size of any vertex cover in G.*

In this paper we define vc as the minimum size of any vertex cover in G_I.

In our example we have $vc = 3$, corresponding to the vertex cover defined by the three nodes $\{21, 22, 23\}$. Also note that the maximum clique size is 4 - and thus $pw = 3$ - e.g. with jobs $21, 22, 23, 10$, the time windows of which overlap at time 90. The maximum number of time windows strictly including a single time window is reached for example by job 6 which is included in the time window of jobs $21, 22$, and thus $q = 2$. Notice that the time window of job 10 is not strictly included in any time window.

As with other common graph parameters it is NP-complete to compute pathwidth and minimum vertex cover for general graphs (Garey and Johnson

1979). However, the underlying interval structure of the compatibility graph eases the computation of these parameters. Pathwidth pw can be computed in time $\mathcal{O}(n \cdot log(n))$ by sorting the release dates and deadlines and then by going over them in order while updating the number of overlapping time windows accordingly (Hanen and Munier Kordon 2023). Proper level can be computed with a straightforward algorithm in $\mathcal{O}(n^2)$. Eventually, a vertex cover of size $vc(G)$ can be computed in linear time when restricted to interval graphs (Marathe et al. 1992), so parameter vc can be computed in linear time.

Bessy and Giroudeau considered the vertex-cover parameter in the context of coupled-task scheduling with a compatibility graph $G_c = (\mathcal{J}, E)$ which, unlike graph G_I, was not necessarily an interval graph (Bessy and Giroudeau 2019). In G_c there was an edge between two coupled tasks in G_c if and only if they could be interleaved. They showed that computing the minimum makespan is FPT parameterized by $vc(G_c)$ plus the maximum length of a coupled task.

Parameter vc is closely related to pathwidth pw of G_I as shown in the following proposition:

Proposition 6. *In any scheduling instance featuring job time windows we have:* $q \leq pw \leq vc$.

Proof. Proof. The relation between q and pw was established by Mallem et al. (2024). Now in an instance with pathwidth pw the job time window interval graph admits a clique of size $pw + 1$. This forces at least pw jobs to be included in any vertex cover of this graph. □

This means that any problem FPT for parameter pw is also FPT for vc. We can thus derive the following result from Hanen et al. (2022) which provides a FPT algorithm parameterized by pw on single machine scheduling with precedence constraints and job time windows minimizing maximum lateness (i.e. the maximum difference between the completion time of a job and its due date).

Corollary 7. $1|prec, r_j, \bar{d_j}|L_{max}$ *is* FPT *parameterized by* vc.

3. A Polynomial Kernel with Parameters Vertex Cover and Maximum Processing Time

In this section we propose a kernelization of $1|prec, r_j, \bar{d_j}|\star$ with respect to parameters vc and $p_{\max} = \max_{i \in \mathcal{J}} p_i$ with size $\mathcal{O}(\log(vc^3 \cdot p_{\max}) \cdot vc^2)$.

Let I be an instance of $1|prec, r_j, \bar{d}_j|\star$. Let $\mathcal{V} = (v_i)_{1 \leq i \leq vc}$ be a vertex cover of G_I with minimum size vc. Let $(u_\alpha)_{\alpha \geq 0}$ be the increasing sequence of distinct release date/deadline values of the jobs from $\mathcal{V}$. Note that there are at most $2vc$ such values. We denote vc_α the number of jobs of vertex cover $\mathcal{V}$ such that $[u_\alpha, u_{\alpha+1})$ is included in their time window: $vc_\alpha = |\{j \in \mathcal{V}, r_j \leq u_\alpha \text{ and } u_{\alpha+1} \leq \bar{d}_j\}|$.

For our example, we get $\mathcal{V} = \{21, 22, 23\}$, $u_0 = 0, u_1 = 90, u_2 = 100, u_3 = 200$, and $vc_0 = 2, vc_1 = 3, vc_2 = 1$.

First note that the jobs outside the vertex cover have limited time window possibilities :

Proposition 8. *Given a vertex cover $\mathcal{V}$ of G_I the jobs in $\mathcal{J} - \mathcal{V}$ form a sequence $(w_j)_{1 \leq j \leq |\mathcal{J}-\mathcal{V}|}$ such that $\bar{d}_{w_j} \leq r_{w_{j+1}}$ for all $1 \leq j < |\mathcal{J} - \mathcal{V}|$.*

Proof. Proof. By the definition of $\mathcal{V}$ all job pairs in $\mathcal{J}-\mathcal{V}$ must have an empty time window intersection. So we get sequence $(w_j)_{1 \leq j \leq |\mathcal{J}-\mathcal{V}|}$ by ordering the jobs in $\mathcal{J} - \mathcal{V}$ by increasing release dates. □

Definition 9. *Let w_j, w_{j+1} be two consecutive jobs of $\mathcal{J}-\mathcal{V}$. In any schedule of these two jobs we call gap the time interval $[s_{w_j} + p_{w_j}, s_{w_{j+1}})$. We call delimiters of the gap the two jobs w_j, w_{j+1}. By extension we call gap g_j the tuple w_j, w_{j+1}. The maximum capacity of the gap is obtained by left shifting w_j and right shifting w_{j+1} and thus equals $\bar{d}_{w_{j+1}} - p_{w_{j+1}} - r_{w_j} - p_{w_j}$. In some interval $[u_\alpha, u_{\alpha+1})$ a gap g_j is an inner gap if $u_\alpha \leq r_{w_j}$ and $\bar{d}_{w_{j+1}} \leq u_{\alpha+1}$ (i.e. if it is included in interval $[u_\alpha, u_{\alpha+1})$ in any schedule). Other gaps are called border gaps.*

By Proposition 8, the existence of a feasible schedule relies on the feasibility of inserting jobs of $\mathcal{V}$ into gaps defined by the sequence of jobs of $\mathcal{J} - \mathcal{V}$. When placing ourselves in some interval $[u_\alpha, u_{\alpha+1})$ - i.e. where no release date/deadline from the vertex cover is interfering - this is reminiscent of the multiple knapsack problem (Gurski et al. 2019). Indeed in an instance of this problem with k items, any solution can be converted to one where only the k knapsacks of highest capacity are used. In our problem the gaps between consecutive jobs of $\mathcal{J} - \mathcal{V}$ could be interpreted as knapsacks, except their capacities can be adjusted from both ends by changing when their delimiters are scheduled, and doing so impacts the capacity of the neighboring gaps. As the number of knapsacks can be bounded by the number of items in multiple knapsack, such analogy suggests that only a bounded number of

gaps is enough to keep the equivalence with the original instance. We first prove that finding a feasible schedule only requires a bounded number of inner gaps in each interval $[u_\alpha, u_{\alpha+1})$.

Lemma 10. *If the instance I is feasible then there is a schedule where any job $i \in \mathcal{V}$ with $r_i \leq u_\alpha, \bar{d}_i \geq u_{\alpha+1}$ is scheduled either in a border gap or in one of the $3vc_\alpha$ highest maximum capacity inner gaps of interval $[u_\alpha, u_{\alpha+1})$.*

Proof. Proof. If there are less than $3vc_\alpha$ inner gaps, the lemma is obvious. Assume there are more than $3vc_\alpha$ inner gaps. The key idea of the proof is that if a gap w_j, w_{j+1} is at maximum capacity, this may reduce its two neighbors gaps w_{j-1}, w_j and w_{j+1}, w_{j+2} but does not impact the size of the farther gaps. So by considering $3vc_\alpha$ gaps we can have vc_α gaps among them at maximum capacity.

Let $g_1, \ldots, g_{3 \cdot vc_\alpha}$ be the $3 \cdot vc_\alpha$ inner gaps with the highest maximum capacity. Let τ be a feasible schedule on instance I. Then in interval $[u_\alpha, u_{\alpha+1})$ at most vc_α inner gaps of $[u_\alpha, u_{\alpha+1})$ are filled by jobs from vertex cover $\mathcal{V}$. Let $g'_1, \ldots, g'_k$ be these gaps ($k \leq vc_\alpha$). We show that the jobs scheduled in these gaps can be scheduled in gaps $g_1, \ldots, g_{3 \cdot vc_\alpha}$ instead while keeping feasibility. We do so by mapping each gap g'_i to some gap g_j with larger or equal maximum capacity.

If a gap g'_i is already one of the $3 \cdot vc_\alpha$ gaps with the highest maximum capacity then we assign it to itself. Then for all other gaps g'_i we assign a gap g_j which is a neighbor of no other assigned gap $g_{j'}$. This allows us to use gap g_j at its maximum capacity, which is necessarily higher than or equal to equal to the capacity used by schedule τ in gap g'_i. Since each gap has at most two neighbors, at any iteration of this process there can only be at most $3(k-1) < 3 \cdot vc_\alpha$ gaps $g_1, \ldots, g_{3 \cdot vc_\alpha}$ which are either already assigned or a neighbor of an already assigned gap. So for each gap g'_i used in schedule τ a suitable gap g_j can be found in time $\mathcal{O}(vc_\alpha)$. Plus *prec*-consistency ensures that within every interval $[u_\alpha, u_{\alpha+1})$ there is no precedence relation between a job from the vertex cover and an inner job from $\mathcal{J} - \mathcal{V}$. So the jobs in τ can move to their assigned gaps without invalidating any precedence constraint. Thus the schedules which only use the (at most) $3 \cdot vc_\alpha$ inner gaps with the highest maximum capacity in each interval $[u_\alpha, u_{\alpha+1})$ are dominant. □

We now need to provide reduction rules that will reduce the set of jobs and the values of release times and deadlines by keeping only useful gaps.

To this purpose, we first define a basic transformation of the instance, called a Left Shift and Replace, that will be used in the next reduction rule.

Definition 11. *Let I be an instance of our problem,w_i, w_{i+1} and w_j, w_{j+1} two gaps with $i+1 < j$. We call Left Shift and Replace and denote $LSR(I, i, j)$ the following modified instance, illustrated by figure 2, in which the time interval between $\bar{d}_{w_{i+1}}$ and r_{w_j} is reduced to a single time unit occupied by a substitute job, by applying a negative offset to all release times and deadlines greater than or equal to r_{w_j}. Formally:*

1. *remove all jobs w_k with $i + 1 < k < j$,*
2. *add a new substitute job w' with $p_{w'} = 1, r_{w'} = \bar{d}_{w_{i+1}}$ and $\bar{d}_{w'} = r_{w'} + 1$,*
3. *apply an offset of $-(r_{w_j} - \bar{d}_{w'})$ to all release times and deadlines of jobs w_k with $k \geq j$ and jobs $x \in \mathcal{V}$ such that $r_x \geq r_{w_j}$.*

A Left Shift and Replace is said to be valid if: (I is feasible) $\Longleftrightarrow$ ($LSR(I, i, j)$ is feasible).

Observe that the gaps introduced by the substitute jobs w' are gaps w_{i+1}, w' and w', w_j. Notice that $r'_w = \bar{d}_{w_{i+1}}$, so that the maximum capacity of the gap w_{i+1}, w' is less than the maximum capacity of the gap w_{i+1}, w_{i+2} in the original instance. Similarly, the maximum capacity of gap w', w_j is less than the maximum capacity of w_{j-1}, w_j in the original instance, since the time window of w_j is left shifted in instance $LSR(I, i, j)$ so that its release time equals the deadline of w'. Lemma 12 states conditions for which $LSR(I, i, j)$ is valid.

Lemma 12. *Let I be an instance, $[u_\alpha, u_{\alpha+1})$ one of its interval and let $w_{i+1}, \ldots, w_j$, with $r_{i+1} \geq u_\alpha, \bar{d}_j \leq u_{\alpha+1}$ be a sequence of jobs delimiting inner gaps. If:*

$$(I \text{ is feasible}) \implies (\text{there exists a feasible schedule for which no job of } \mathcal{V} \text{ is scheduled in the gaps whose delimiters are in } \{w_{i+1}, \ldots, w_j\}),$$

then $LSR(I, i, j)$ is valid.

Proof. Proof. Let s be a feasible schedule of I with the property stated in the Lemma. We can easily build a schedule s' of $LSR(I, i, j)$ by applying an offset $-(r_{w_j} - \bar{d}_{w'})$ to all starting times after r_{w_j} (i.e. $s'(k) = s(k) - (r_{w_j} - \bar{d}_{w'})$ if $s(k) > r_{w_j}$). We claim that this schedule is feasible. If $s(k) \leq r_{w'}$ then

$s'(k) = s(k)$. And because we assumed that the jobs of the vertex cover were not scheduled in the removed gaps, they are scheduled in the gaps to the left or to the right of w' in instance $LSR(I, i, j)$. Since these gaps have the same length after the offset, no job conflict arises and the proposed schedule is feasible.

Conversely if we have a feasible schedule s' of $LSR(I, i, j)$, then to build a schedule s of the original instance we just apply an offset $(r_{w_j} - \bar{d}_{w'})$ to all starting times $\geq \bar{d}_{w'}$. Then the removed jobs w_{i+2} to w_{j-1} are scheduled at their release times in sequence. Indeed, the gaps w_{i+1}, w_{i+2} has then the same length as w_{i+1}, w' in the modified instance, and the same argument hold for gap w_{j-1}, w_j. As we assumed *prec*-consistent deadlines, if the time window of w_k is included in $[u_\alpha, u_{\alpha+1})$ there cannot be a precedence constraint between w_k and a job x of $\mathcal{V}$ with $r_x \leq u_\alpha$ and $\bar{d}_x \geq u_{\alpha+1}$. So no precedence constraint is violated. □

We can now define our first reduction rule.

Reduction rule 13. *For each α: if there are more than $3vc_\alpha$ inner gaps in $[u_\alpha, u_{\alpha+1})$, select the $3vc_\alpha$ ones with highest maximum capacity. We denote by S the set of selected gaps to which we add the border gaps. While there are two non consecutive gaps w_i, w_{i+1}, w_j, w_{j+1} in S without a substitute job, update the instance: $I \leftarrow LSR(I, i, j)$.*

Figure 2 shows the reduction process.

Applying the reduction to our example would give the following result. In time interval $[u_2, u_3) = [100, 200)$, $vc_2 = 1$, and the three maximum capacity inner gaps are $(17, 18), (18, 19)$ with maximum capacity 10, and $(19, 20)$ with maximum capacity 13. The border gap is $(10, 11)$ So, jobs 12 to 16 can be removed and replaced by substitute job b with release date 110 and deadline 111. The release times and deadlines of jobs $17, 18, 19, 20, 23$ decrease by $160 - 111 = 49$.

In time interval $[u_1, u_2)$ no job can be removed, jobs 10, 11 are still in the instance. In time interval $[u_0, u_1) = [0, 90)$, $vc_0 = 2$, we have 10 jobs in $\mathcal{J} - \mathcal{V}$, delimiting 9 gaps fully in interval $[0, 90)$. Gap $(1, 2)$ has maximum capacity 13 (by left shifting job 1 and right shifting job 2) while the other gaps have maximum capacity 10. So we can select only the $6 = 3 \cdot vc_0$ first gaps, add the border gap $(9, 10)$ and remove job 8. Substitute job a replaces 8 with release time 70 and deadline 71. Then release times and deadlines

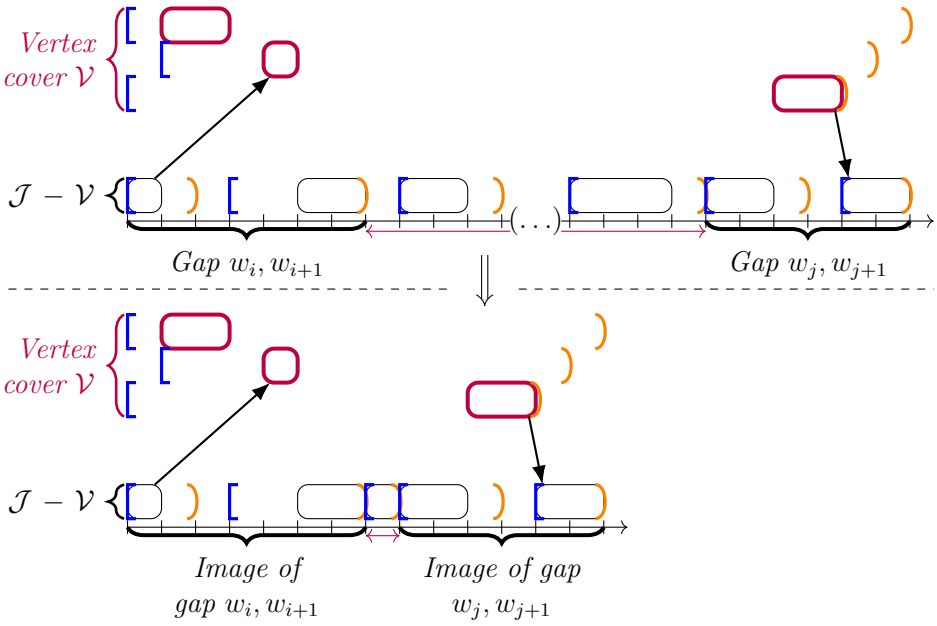


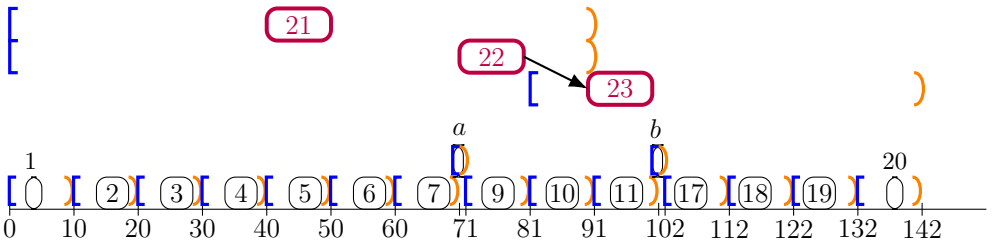
Figure 2: Job removal process $LSR(I, i, j)$ between two consecutive non-neighboring gaps w_i, w_{i+1} and w_j, w_{j+1}.

greater than 80 decrease by $80 - 71 = 9$. This leads to the reduced instance depicted in Figure 3.

Lemma 14. *In each interval $[u_\alpha, u_{\alpha+1})$ of a prec-consistent instance reduction 13 is a valid. After its application to an interval $[u_\alpha, u_{\alpha+1})$, the number of jobs w_j from $\mathcal{J} - \mathcal{V}$ whose time window intersects $[u_\alpha, u_{\alpha+1})$ can be reduced down to $\mathcal{O}(vc_\alpha)$. Thus by applying iteratively reduction 13 the number of jobs w_j from $\mathcal{J} - \mathcal{V}$ can be reduced down to $\mathcal{O}(vc^2)$.*

Proof. Proof. Lemma 10 establishes that if I is a feasible instance, there exists a schedule such that no job of $\mathcal{V}$ is scheduled in a non-selected gap. And Lemma 12 indicates that if there are two non consecutive selected gaps, then the Left Shift and Replace rule is valid. So that iterating on α and on LSR reduction keeps the equivalence between instances, so that reduction rule is valid.

Let us now count the number of jobs. The number of substitute jobs introduced in interval $[u_\alpha, u_{\alpha+1})$ is at most $3vc_\alpha - 1$, plus two delimiters of the left and right non inner gap in interval $[u_\alpha, u_{\alpha+1})$ By adding the two delimiters of the at most $3 \cdot vc_\alpha$ gaps, we successfully bound the number of



i	1	2	3	4	5	6	7	a	9	10	11	b	17	18	19	20	21	22	23
p_i	2	5	5	5	5	5	5	1	5	5	5	1	5	5	5	2	10	10	10
r_i	0	10	20	30	40	50	60	70	71	81	91	101	102	112	122	132	0	0	81
$\bar{d}_i$	10	20	30	40	50	60	70	71	81	91	101	102	112	122	132	142	91	91	142

Figure 3: Example after applying Reduction Rule 13.

jobs from $\mathcal{J} - \mathcal{V}$ in each interval $[u_\alpha, u_{\alpha+1})$ by $9 \cdot vc_\alpha + 1$. As there are at most $2vc$ possible values of α, the lemma holds. □

Now the values of the release times and deadlines are still not related to our parameters. We define reduction rule 15 which allows us to further reduce release times and deadlines to values only dependent on vc and $p_{\max}$. Let I be an instance of the problem, and let $(v_\beta)_{\beta \geq 0}$ be the sorted sequence of release dates and deadlines of jobs in $\mathcal{J}$. Notice that in each interval $[v_\beta, v_{\beta+1})$ at most $vc+1$ jobs could be performed: the jobs of $\mathcal{V}$ and the only job w_j (if any) whose time window crosses this interval. As such we propose the following reduction rule:

Reduction rule 15. *If $v_{\beta+1} - v_\beta > (vc+1) \cdot p_{\max}$, then set $t = v_\beta + (vc+1) \cdot p_{\max}$. Apply an offset of $-(v_{\beta+1} - t)$ to all release times and deadlines $\geq v_{\beta+1}$.*

Figure 4 shows the reduction principle. Notice that it cannot be applied to our example.

Lemma 16. *Reduction rule 15 is valid.*

Proof. Proof. If $v_{\beta+1} - v_\beta > (vc+1) \cdot p_{\max}$ then if there is a feasible schedule, we know that the load of interval $[v_\beta, v_{\beta+1})$ is at most $(vc+1) \cdot p_{\max}$. Since

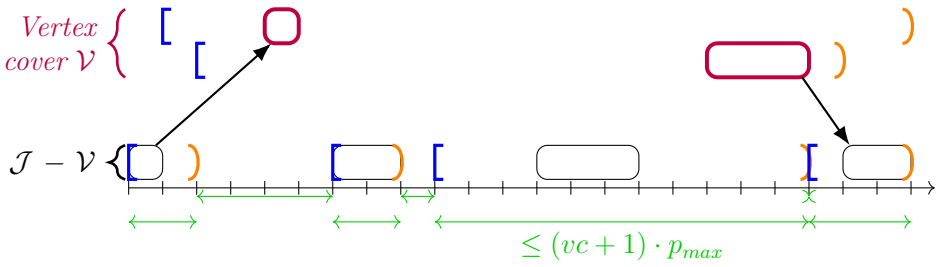


Figure 4: Illustration of the bounds between the release dates and deadlines of the jobs in $\mathcal{J} - \mathcal{V}$.

no release time or deadline is strictly included in this interval we can assume that there is a feasible schedule where the jobs are left shifted in this interval, so that interval $[v_\beta + (vc+1) \cdot p_{\max}, v_{\beta+1})$ is empty. So, by left shifting every start time release date and deadline above $v_{\beta+1}$ we build a feasible schedule of the new instance. And conversely we just have to apply an offset of $(v_{\beta+1} - t)$ to all dates above $v_{\beta+1}$ to get a feasible schedule of the original instance. □

This gives a kernelization with respect to parameter $vc + p_{\max}$.

Proposition 17. *With respect to parameter $vc + p_{\max}$ problem $1|prec, r_j, \bar{d}_j|\star$ has a kernelization in time $\mathcal{O}(n^2 \cdot log(n))$. The resulting kernel has size $\mathcal{O}(log(vc^3 \cdot p_{\max}) \cdot vc^2)$ and $\mathcal{O}(vc^2)$ jobs.*

Proof. Proof. Consider the instance I' build from instance I after applying iteratively all our reduction rules. To build I', we begin by ensuring that the instance is *prec*-consistent in time $\mathcal{O}(n^2)$. Then a vertex cover of minimum size vc is computed in time $\mathcal{O}(|G_I|) = \mathcal{O}(vc^2)$ (Marathe et al. 1992). After that the sequence $(u_\alpha)_\alpha$ of interval bounds is computed in time $\mathcal{O}(vc \cdot log(vc))$. Finally Lemma 14 can be applied to remove all but $\mathcal{O}(vc^2)$ jobs from $\mathcal{J} - \mathcal{V}$. This gives an equivalent instance $\hat{I}$ with $\mathcal{O}(vc^2)$ release times and deadlines. Now according to reduction rule 15, after iterative application of the rule the distance between two consecutive release dates or deadlines v_β is $\mathcal{O}(vc \cdot p_{\max})$. So the maximum deadline of any job is $\mathcal{O}(vc^3 \cdot p_{\max})$. Thus the encoding of each release dates/deadline in I' is $\mathcal{O}(\log(vc^3 \cdot p_{\max}))$. □

4. A Polynomial Kernel for the Vertex Cover Parameter

In this section we give an additional reduction rule which reduces the size of the kernel of jobs down to $\mathcal{O}(vc^8)$. Note that in our previous kernel, $p_{\max}$

was only needed to bound the size of the time values. So our goal is to reduce these values down to a function of vc.

First we consider the following integer program with positional variables that was proved equivalent to the associated scheduling instance by Lasserre and Queyranne (1992):

$$x_{i,j} = \begin{cases} 1 & \text{if job } j \text{ is the } i^{th} \text{ job in the schedule,} \\ 0 & \text{otherwise.} \end{cases}$$

Binary integer program $BIP(I)$ is defined as follows:

$$\begin{cases} \text{One job per position} & \sum_{j=1}^{n} x_{i,j} = 1 \text{ for all } 1 \leq i \leq n. \\ \text{One position per job} & \sum_{i=1}^{n} x_{i,j} = 1 \text{ for all } 1 \leq j \leq n. \\ \text{Precedence constraints} & \sum_{i=1}^{i_0} (x_{i,j_1} - x_{i,j_2}) \geq 0 \text{ for all } 1 \leq i_0 \leq n, (j_1, j_2) \in A. \\ \text{Time window checks} & \sum_{j=1}^{n} r_j \cdot x_{i_1,j} + \sum_{i=i_1}^{i_2} \sum_{j=1}^{n} p_j \cdot x_{i,j} \leq \sum_{j=1}^{n} \bar{d}_j \cdot x_{i_2,j} \\ & \text{for all } 1 \leq i_1 \leq i_2 \leq n. \end{cases}$$

The linear constraint associated with a precedence constraint $(j_1, j_2) \in A$, checks whether, for each position i_0 if j_2 is scheduled in an earlier position than i_0 then j_1 also is. The constraint representing time windows checks for each tuple of positions $i_1 \leq i_2$ that the release time of the job in position i_1 plus the sum of the processing times of all jobs in consecutive positions i_1 to i_2 does not exceed the deadline of the job in position i_2. This constraints relies on the dominance of active schedules for makespan minimization. So such a schedule can be decomposed into "blocks" of jobs which start at the release date of the first job in each block. The constraints consider all the $\frac{n(n-1)}{2}$ block possibilities.

Lemma 18. *(Lasserre and Queyranne 1992) An instance I of $1|prec, r_j, \bar{d}_j|\star$ is feasible if and only if binary integer program $BIP(I)$ has a solution.*

Although there are $\Theta(n^2)$ variables in some constraints of $BIP(I)$, given a solution of the program, we show that at most $n+2$ variables can be equal to one in each constraint.

Lemma 19. *In any solution of $BIP(I)$ at most $n+2$ variables are equal to one in each constraint.*

Proof. Proof. At most one variable can be equal to one in the first two set of constraints (one job per position and one position per job). Now, considering

the time window constraint, it sums up with release time coefficient the variables of jobs in position i_1 (so only 1 variable equals one), similarly the third term of the sum considers the deadlines of jobs in position i_2, so only one of them equals one, whereas the intermediate sum considers all positions between i_1 and i_2, so at most n variables equal one. This gives our $(n+2)$ bound. Finally for the precedence constraints, only two variables might be equal to 1 in the sum. □

Knowing this, we intend to use Theorem 20 in order to bound the processing time, release date and deadline values by a function of vc.

Theorem 20. *(Frank and Tardos 1987) Let $\mathbf{w} = (w_1, \ldots, w_d)$ be a vector of rational numbers and let N be a positive integer. Then in time $\mathcal{O}(d^8 \cdot (d + log(N)))$ one can build a vector $\mathbf{w}' = (w'_1, \ldots, w'_d)$ of positive integers such that:*

- *$w'_i \leq 2^{4d^3} \cdot N^{d(d+2)}$ for all $1 \leq i \leq d$,*
- *for every $\mathbf{x} = (x_1, \ldots, x_d)$ vector of integers such that $\sum_{i=1}^{d} |x_i| \leq N-1$:*

$$\sum_{i=1}^{d} w_i x_i \geq 0 \text{ if and only if } \sum_{i=1}^{d} w'_i x_i \geq 0.$$

Let I be an instance of $1|prec, r_j, \bar{d}_j|\star$ and let $\kappa(I)$ be the $(vc + p_{\max})$-kernel obtained with Proposition 17. Let $\tilde{n}$ be the number of its jobs. By Lemma 18 we can consider program $BIP(\kappa(I))$ equivalent to it. We wish to modify the values of the release times, deadlines and processing times using Theorem 20 to make the instance size polynomial in the number of jobs and thus in the parameter vc.

To this end, we set $d = 3\tilde{n}$. Consider vector $\mathbf{w} = (p_1, r_1, \bar{d}_1, \ldots, p_{\tilde{n}}, r_{\tilde{n}})$. Let us set $N = \tilde{n} + 3$. Notice that $d, \tilde{n}, N$ are all bounded by $\mathcal{O}(vc^2)$ by Proposition 17. Now Lemma 19 states that any feasible solution of $BIP(\kappa(I))$ satisfies that the sum of variables involved in a time window constraint is at most $N - 1$. We can thus apply Theorem 20 to get a vector $\mathbf{w}' = (p'_1, r'_1, \bar{d}'_1, \ldots, p'_{\tilde{n}}, r'_{\tilde{n}}, \bar{d}'_{\tilde{n}})$, which represents an instance I' equivalent to $\kappa(I)$ according to Lemma 18. The time values in I' are all bounded by a function of vc.

This gives the following reduction rule:

Reduction rule 21. *Let I be an instance I and $\kappa(I)$ be the instance obtained after applying iteratively reductions 13 and 15. Let $\tilde{n}$ be the number of jobs in $\kappa(I)$. Then we set vector $\mathbf{w} = (p_1, r_1, \bar{d}_1, \ldots, p_{\tilde{n}}, r_{\tilde{n}}, \bar{d}_{\tilde{n}})$ from the time values of $\kappa(I)$ and apply Theorem 20 with $d = 3\tilde{n}$ and $N = \tilde{n} + 3$.*

Applying our three reduction rules leads to a polynomial kernel with respect to vc.

Theorem 22. *With respect to parameter vc problem $1|prec, r_j, \bar{d}_j|\star$ has a polynomial kernelization in time $\mathcal{O}(n^2 \cdot log(n) + (vc^2)^8 \cdot (vc^2 + log(vc^2))) = \mathcal{O}(n^{18})$. The resulting kernel has size $\mathcal{O}(vc^8)$ and $\mathcal{O}(vc^2)$ jobs.*

Proof. Proof. Let I be an instance of $1|prec, r_j, \bar{d}_j|\star$. First we build the $(vc + p_{\max})$-kernel $\kappa(I) = \langle prec, p_j, r_j, \bar{d}_j \rangle$ obtained in time $\mathcal{O}(vc \cdot n \cdot log(n))$ by Proposition 17. Let I' be the instance build from $\kappa(I)$ by the reduction rule 21.

We know that the new coefficients are not greater than $2^{4d^3} \cdot N^{d(d+2)}$. Substituting d by $3\tilde{n}$ and N by $\tilde{n}+3$ yields vector $\mathbf{w}' = (p'_1, r'_1, \bar{d}'_1, \ldots, p'_{\tilde{n}}, r'_{\tilde{n}}, \bar{d}'_{\tilde{n}})$ with components not greater than $2^{\mathcal{O}(\tilde{n}^3)}$. Since $\tilde{n} = \mathcal{O}(vc^2)$, the size of each encoded component is $\mathcal{O}(vc^6)$. And since we have $\mathcal{O}(vc^2)$ jobs in I' we get the announced space bound for it.

Similarly the complexity $\mathcal{O}(d^8 \cdot (d + \log(N)))$ yields complexity $\mathcal{O}((vc^2)^8 \cdot (vc^2 + log(vc^2)))$. Use these values to define $I' = \langle prec, p'_j, r'_j, \bar{d}'_j \rangle$ instance of $1|prec, r_j, \bar{d}_j|\star$. We show that $\kappa(I)$ is feasible if and only if I' is feasible. Lemma 19 ensures that bound $N = \tilde{n}+3$ of all the positive values of variables in each inequality was enough to make integer programs $BIP(\kappa(I))$ and $BIP(I')$ equivalent according to Theorem 20. Plus by Lemma 18 we know that $\kappa(I)$ (resp. I') is feasible if and only if $BIP(\kappa(I))$ (resp. $BIP(I')$) has a solution. So I' is indeed equivalent to $\kappa(I)$, which is itself equivalent to I according to Proposition 17.

□

5. Kernels with Parameters Pathwidth or Proper Level

In this section, we consider kernelization algorithms on $1|prec, r_j, \bar{d}_j|\star$ with respect to pathwidth pw or proper level q. As stated in (Flum and Grohe 1998), a problem has a kernel of size bounded by some parameter if and only if it is fixed-parameter tractable with respect to this parameter. The following FPT results in the literature suggest the consideration of kernels for parameters pathwidth pw or proper level q.

Lemma 23. *(Hanen et al. 2022, Mallem et al. 2024) Problem* $1|prec, r_j, \bar{d}_j|\star$ *has a FPT algorithm with respect to pathwidth pw (resp. proper level q) in time* $\mathcal{O}((pw+1)^2 \cdot 4^{pw} \cdot n + n \cdot log(n))$ *(resp.* $\mathcal{O}((q+1)^2 \cdot 4^q \cdot n + n \cdot log(n))$*).*

5.1. A Lower Bound on the Kernel Size with Respect to Pathwidth

First, we use the cross-composition technique developed in (Bodlaender et al. 2014) to show that our single machine problem, although FPT parameterized by pathwidth pw, does not admit a polynomial kernel under usual parameterized complexity assumptions.

We consider here a special case of cross-composition where the equivalence relation mentioned in the original definition of (Bodlaender et al. 2014) decides whether a string is a valid instance of a decision problem or not.

Definition 24. *Let* Σ *be an alphabet and let* $L \subset \Sigma^*$ *and* $C \subset \Sigma^*$ *with* $L \subset C$ *be two languages over* Σ*. Let* $Q \subset \Sigma^* \times \mathbb{N}$ *be a parameterized problem.* L *cross-composes into* Q *if there is an algorithm which, given* t *strings* $x_1, x_2, \ldots, x_t$ *in* C *computes an instance* $(x^*, k^*) \in \Sigma^* \times \mathbb{N}$ *in time polynomial in* $(\Sigma_{i=1}^{t}|x_i|)$ *such that:*

1. *deciding whether a given* $x \in \Sigma^*$ *is in* C *can be done in* $\mathcal{O}(|x|^{\mathcal{O}(1)})$*,*
2. $(x^*, k^*) \in Q \iff x_i \in L$ *for some* $1 \leq i \leq t$*,*
3. k^* *is bounded by a polynomial in* $\max_{1 \leq i \leq t} |x_i| + \log(t)$*.*

We consider here that L will be the set of feasible instances of the PARTITION problem, whereas C will be the set of instances of the PARTITION problem, which is defined as follows:

Definition 25. *An instance of PARTITION is defined by a set* A *of* n *integers* $a(1), \ldots, a(n)$ *and an integer* B *such that* $\sum_{i=1}^{n} a(i) = 2B$*. The instance is feasible if there exists a subset* $X \subset A$ *such that* $\sum_{a(i) \in X} a(i) = B$*.*

Consider t instances of the PARTITION problem. We denote n_k the number of integers of the instance I_k, by $a_k(i)$ the i^{th} integer of instance I_k, and by B_k its threshold. From these t instances we build an instance $\mathcal{J}(I_1, \ldots, I_t)$ of $1|r_i, \bar{d}_i|\star$ parameterized by pw - i.e. our single machine problem with an empty precedence relation. We first set $T = 2 \cdot (\max_{1 \leq i \leq t} B_i)$, and for all $1 \leq k \leq t$: $D_k = (T+1) \cdot k + 2 \cdot (\sum_{i=1}^{k}(B_i + 1))$. We notably have: $D_k - D_{k-1} = 2B_k + T + 3$.

All jobs associated to instance I_k have their time window between D_{k-1} and D_k. First to each PARTITION instance I_k are associated three unit-time

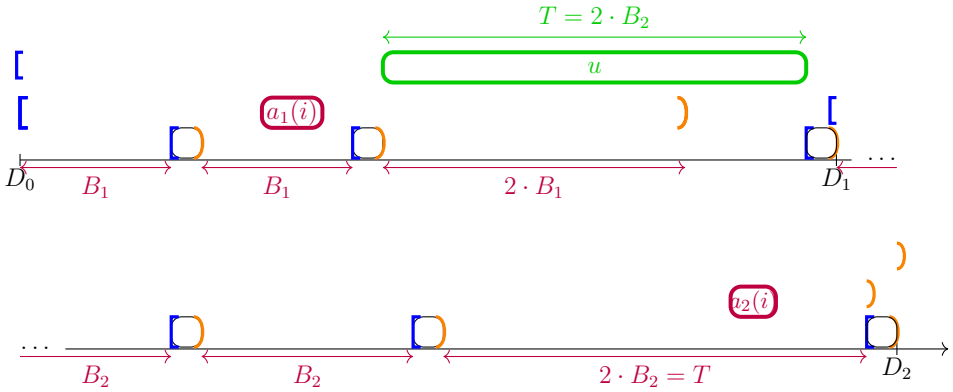


Figure 5: Cross-composition from two instances I_1, I_2 of PARTITION.

fill jobs $f_{k,i}$ $(1 \leq i \leq 3)$. These jobs have a time window of length one and delimit three zones in interval $[D_{k-1}, D_k)$: two zones of length B_k and one of length T. See Figure 5 for an example with $k = 2$. Below is a more detailed description of the fill job time windows:

	$f_{k,1}$	$f_{k,2}$	$f_{k,3}$
r_i	$D_{k-1} + B_k$	$D_{k-1} + 2B_k + 1$	$D_k - 1$
$\bar{d}_i$	$D_{k-1} + B_k + 1$	$D_{k-1} + 2B_k + 2$	D_k

Then for each integer $a_k(i)$ in PARTITION instance I_k we set a job of processing time $a_k(i)$, release time D_{k-1} and deadline $D_{k-1} + 4B_k + 2$ $(= D_k - T - 1 + 2B_k)$.

Finally we define a job u, which we call the *selector job*, with processing time T and time window $r_u = 0$, $\bar{d}_u = D_t$. Due to its processing time it can only be scheduled in the third zone of an interval $[D_{k-1}, D_k)$ for some k. This forces all jobs associated to PARTITION instance I_k to be scheduled in the two remaining zones of length B_k in this interval.

Lemma 26. *The instance $\mathcal{J}(I_1, \ldots, I_t)$ is a cross-composition.*

Proof. Proof. The polynomiality of the construction is straightforward. Let us now compute the value of parameter pw. As each instance I_k has all its associated jobs within the interval $[D_{k-1}, D_k)$, which interferes with at most one fill job and the job u at a given time, so that the pathwidth of the instance is $pw = 1 + \max_{1 \leq k \leq t} n_k$. As the size of I_k is a function of n_k our parameter satisfies the cross-composition condition.

We finally have to verify that the instance $\mathcal{J}(I_1, \ldots, I_t)$ is feasible if and only if one of the PARTITION instance is feasible.

$\Longleftarrow$ Assume that the PARTITION instance I_k is feasible. Let X_k be the subset of values such that $\sum_{a_k(i) \in X_k} a_k(i) = B_k$. We build a feasible schedule by scheduling job u at time $D_k - T - 1$, the tasks of X_k in sequence in the interval $[D_{k-1}, D_{k-1} + B_k)$, and the other tasks $a_k(j)$ are performed in sequence in interval $[D_{k-1} + B_k + 1, D_{k-1} + 2B_k + 1)$. The jobs $a_l(i)$ of the other instance are all performed in any order in the interval $[D_{l-1} + 2B_l + 2, D_{l-1} + 4B_l + 2)$. Job u does not interfere with any other job, so the schedule is feasible.

$\Longrightarrow$ Let us observe that in any feasible schedule, there is a k such that job u is performed at time $D_k - 1 - T$ between $f_{k,2}$ and $f_{k,3}$. Indeed, the size of the interval between two other fill jobs is always strictly less than T. Now the other jobs $a_k(i)$ cannot be scheduled in interval $[D_k - 1 - T, D_k)$ that perform u and a fill job. So they can only be executed before $f_{k,1}$ or between $f_{k,1}$ and $f_{k,2}$. If all jobs are executed this defines a partition of this instance: X_k is the set of jobs performed before $f_{k,1}$. □

We immediately derive the following result:

Corollary 27. *Unless $NP \subseteq coNP/poly$ there is no polynomial kernel for $1|r_i, \bar{d}_i|\star$ parameterized by pathwidth pw (resp. proper level q).*

Proof. Proof. It is well known that the PARTITION problem is NP-hard (Garey and Johnson 1979). From Theorem 9 of (Bodlaender et al. 2014) we deduce that Corollary 27 holds for parameter pw. Notice that as $q \leq pw$ (see Proposition 6). Hence, a polynomial kernel for parameter q would be a polynomial kernel for pw. So, the same conclusion can be driven about the existence of polynomial kernel for parameter q. □

5.2. Exponential Size Kernels with Respect to Pathwidth or Proper Level

In response one can look for kernels of exponential size instead. In fact the latter can be directly inferred from the existing FPT algorithms in the literature. Algorithm 1 gives an example of such a kernelization algorithm.

While this first algorithm successfully yields kernels of size exponential in pw, in the "if" branch the instance is left untouched. This means that the kernel size is lower bounded by the time values, notably by $max_{j \in \mathcal{J}}(log(\bar{d}_j))$. To get around this, one can use the polynomial kernel with respect to vertex cover vc proposed in Section 4. Algorithm 2 gives an example of such an

Algorithm 1 NaiveKernel($1|prec, r_j, \bar{d}_j|\star$) with pathwidth pw

Input: an instance I of $1|prec, r_j, \bar{d}_j|\star$.
Compute pathwidth pw in time $\mathcal{O}(n \cdot log(n))$.
if $|I| \leq 2^{pw}$ **then**
    return I
else
    Use the FPT algorithm from (Hanen et al. 2022) in time $\mathcal{O}((pw+1)^2 \cdot 4^{pw} \cdot n + n \cdot log(n)) = \mathcal{O}(|I|^2 \cdot log(|I|) \cdot n)$.
    if the output is YES **then**
        return a dummy YES instance of size $\mathcal{O}(1)$.
    else
        return a dummy NO instance of size $\mathcal{O}(1)$.
    end if
end if

updated kernelization algorithm. Note that adding our polynomial kernel with respect to vc to the "if" branch allowed us to update the "if" condition, which led to an upper bound on the time complexity of the "else" branch that depended on vc and n only, and no longer on input size $|I|$.

Now recall that after applying Reduction Rule 13 in the polynomial kernel with respect to vc, the number of inner gaps in each interval $[u_\alpha, u_{\alpha+1})$ is linearly bounded by the number vc_α of jobs from the vertex cover, the time window of which intersects with interval $[u_\alpha, u_{\alpha+1})$. Note that the number of such inner gaps can be bounded by pathwidth pw or proper level q instead.

Lemma 28. *For every interval $[u_\alpha, u_{\alpha+1})$: $vc_\alpha \leq pw + 1$.*
For every interval $[u_\alpha, u_{\alpha+1})$ with at least two inner gaps: $vc_\alpha \leq q$.

Proof. Proof. The first bound comes from the definition of pathwidth pw being applied to interval $[u_\alpha, u_{\alpha+1})$. Now if there are at least two inner gaps in an interval $[u_\alpha, u_{\alpha+1})$, then there is a job j such that $u_\alpha < r_j < \bar{d}_j < u_{\alpha+1}$. So all the vc_α jobs from the vertex cover, the time window of which intersects with interval $[u_\alpha, u_{\alpha+1})$, have a lower release date than r_j and a higher deadline than $\bar{d}_j$. Thus $vc_\alpha \leq q$.

□

Then in the "if" branch of Algorithm 2, using Lemmas 14, and 28, the number of jobs in the kernel is $\mathcal{O}((pw+1) \cdot vc)$. Then when applying Theorem 22 the bounds on the number of jobs and the kernel size can be rewritten

Algorithm 2 Kernel($1|prec, r_j, \bar{d}_j|\star$) with pathwidth pw

1: Input: an instance I of $1|prec, r_j, \bar{d}_j|\star$.
2: Compute pathwidth pw in time $\mathcal{O}(n \cdot log(n))$.
3: **if** $vc \leq 2^{pw}$ **then**
4: **return** the polynomial kernel of I with respect to parameter vc from Section 4.
5: **else**
6: Use the FPT algorithm from (Hanen et al. 2022) in time $\mathcal{O}((pw+1)^2 \cdot 4^{pw} \cdot n + n \cdot log(n)) = \mathcal{O}(vc^2 \cdot log(vc) \cdot n)$.
7: **if** the output is YES **then**
8: **return** a dummy YES instance of size $\mathcal{O}(1)$.
9: **else**
10: **return** a dummy NO instance of size $\mathcal{O}(1)$.
11: **end if**
12: **end if**

as $\mathcal{O}((pw+1) \cdot 2^p w)$ and $\mathcal{O}((pw+1)^4 \cdot 2^{4pw})$ respectively, which sightly improves the upper bound on the time complexity of the polynomial kernel with respect to vc. Finally recall that both pw and vc are at most equal to the number of jobs n. So Algorithm 2 runs in polynomial time and yields kernels, the size of which is no longer lower bounded by the time values in the original instance.

Corollary 29. *With respect to parameter pw problem $1|prec, r_j, \bar{d}_j|\star$ has a kernelization in time $\mathcal{O}(vc^2 \cdot log(vc) \cdot n + (pw+1)^9 \cdot vc^9) = \mathcal{O}(n^{18})$. The resulting kernel has size $\mathcal{O}((pw+1)^4 \cdot 2^{4pw})$ and $\mathcal{O}((pw+1) \cdot 2^p w)$ jobs.*

Note that by changing line 2 in Algorithm 2 from "$vc \leq 2^{pw}$" to "$vc \leq 2^q$" and using the FPT algorithm from (Mallem et al. 2024), one can get a similar kernelization result with respect to proper level q. Indeed when there are less than two inner gaps in interval $[u_\alpha, u_{\alpha+1})$ the number of jobs w_j in this interval is bounded by a constant, so in either case the reduction rules lead to an instance with $\mathcal{O}((q+1) \cdot vc)$ jobs.

Corollary 30. *With respect to parameter q problem $1|prec, r_j, \bar{d}_j|\star$ has a kernelization in time $\mathcal{O}(vc^2 \cdot log(vc) \cdot n + (q+1)^9 \cdot vc^9) = \mathcal{O}(n^{18})$. The resulting kernel has size $\mathcal{O}((q+1)^4 \cdot 2^{4q})$ and $\mathcal{O}((q+1) \cdot 2^q)$ jobs.*

5.3. Bounding the Number of Distinct Release Date and Deadline Values

Now, while parameters pw and q can be used to bound the number of jobs in each interval $[u_\alpha, u_{\alpha+1})$ in the kernel, they are unrelated to the number of such intervals. One way to bound the latter is to consider the number of values that delimit these intervals, which corresponds to the number of distinct release date and deadlines. As such, we propose the following two parameters.

Definition 31. *We define parameter $\#\{r_j, \bar{d}_j\}$ as the number of distinct release date and deadline values in the instance. We define parameter $\#_{vc}\{r_j, \bar{d}_j\}$ as the minimum number of distinct release date and deadline values in a vertex cover of G_I.*

Note that with parameter $\#_{vc}\{r_j, \bar{d}_j\}$ we consider all vertex covers and not just the ones of minimum size. In the example given in Figure 6, the purple jobs define the only vertex cover of minimum size vc, which has $2 \cdot vc$ distinct release date and deadline values. However, $\#_{vc}\{r_j, \bar{d}_j\}$ is obtained by taking the $vc+1$ black jobs as the vertex cover, which only compiles $vc+2$ distinct release date and deadlines values.

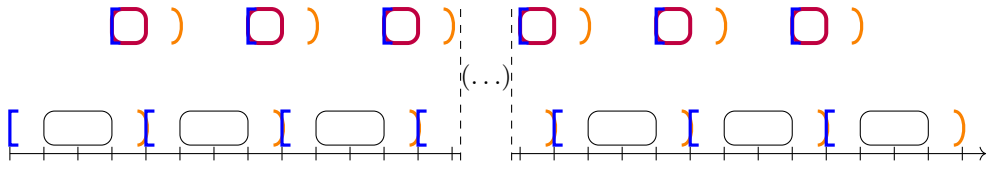


Figure 6: An example where $\#_{vc}\{r_j, \bar{d}_j\}$ is only attained by vertex covers of size greater than vc.

We show that parameter vc can be bounded by a function of parameters pw and $\#_{vc}\{r_j, \bar{d}_j\}$.

Lemma 32. $vc \leq (pw + 1) \cdot (\#_{vc}\{r_j, \bar{d}_j\} - 1)$.

Proof. Proof. Let be V be a vertex cover of minimum size vc and let V' be a vertex cover with a number of distinct release date and deadline values equal to $\#_{vc}\{r_j, \bar{d}_j\}$. Then: $vc = |V| \leq |V'|$. Now sort the distinct release date and deadline values associated to V' into an increasing sequence $(u'_\alpha)_{1 \leq \alpha \leq \#_{vc}\{r_j, \bar{d}_j\}}$. Then given each segment $[u'_\alpha, u'_{\alpha+1})$, the number of jobs from V', the time window of which intersects with this segment, is bounded

by $pw + 1$. Since the time window of every job in V' intersects with at least one of the segments $[u_\alpha, u'_{\alpha+1})$, we have:

$|V'| \leq \sum_{\alpha=1}^{\#_{vc}\{r_j,\bar{d}_j\}-1}(pw+1) \leq (pw+1) \cdot (\#_{vc}\{r_j, \bar{d}_j\} - 1)$. □

This bound is tight and attained when, for every $0 \leq \alpha \leq \#_{vc}\{r_j, \bar{d}_j\} - 1$, exactly $pw+1$ jobs from the vertex cover have time window $[u_\alpha, u_{\alpha+1})$. Then Theorem 22 implies the following result.

Corollary 33. *With respect to parameter* $(pw + \#_{vc}\{r_j, \bar{d}_j\})$, $1|prec, r_j, \bar{d}_j|\star$ *has a polynomial kernelization.*

More generally when comparing multiple parameters on the same problem, we can consider the following notions.

Definition 34. *Given a problem* $\mathcal{P}$ *and two parameters* k, k'*, we say that* k *is weaker than* k' *on problem* P *if and only if there is a function* $f : \mathcal{P} \to \mathbb{N}_0$ *such that for all instances* I *of* $\mathcal{P}$ *we have* $k(I) \leq f(k'(I))$*. If this is the case, we also say that* k' *is stronger than* k *on* $\mathcal{P}$*. We say that* k *and* k' *are equivalent if and only if* k *is weaker than* k' *on* $\mathcal{P}$ *and* k' *is weaker than* k *on* $\mathcal{P}$*.*

This defines a partial order relation between the parameters that we have considered on problem $1|prec, r_j, \bar{d}_j|\star$. Consider a parameter k weaker than some other parameter k'. If we have a kernelization with respect to k, then the same algorithm is also a kernelization with respect to k'. And if both the original kernel and the relation between both parameters are polynomial, then it is also a polynomial kernel with respect to k'. Conversely, if there is no kernelization with respect to k', then there is no kernelization with respect to k either. Figure 7 summarizes the kernelization results which can be derived from this partial order relation.

For instance by Lemma 32 we know that parameter vc is weaker than $(pw+\#_{vc}\{r_j, \bar{d}_j\})$ on $1|prec, r_j, \bar{d}_j|\star$. And since parameters pw and $\#_{vc}\{r_j, \bar{d}_j\}$ are bounded by vc and $2vc$ respectively, this means that parameters vc and $(pw + \#_{vc}\{r_j, \bar{d}_j\})$ are actually equivalent. In contrast note that parameter $(q+\#_{vc}\{r_j, \bar{d}_j\})$ is strictly weaker than vc. For example in instances where all jobs have the same time window, vc is equal to $n-1$ whereas $(q+\#\{r_j, \bar{d}_j\})$ is equal to 2.

Now if we consider all distinct release date and deadline values instead of just the ones from a vertex cover, parameter $(pw + \#\{r_j, \bar{d}_j\})$ is strictly

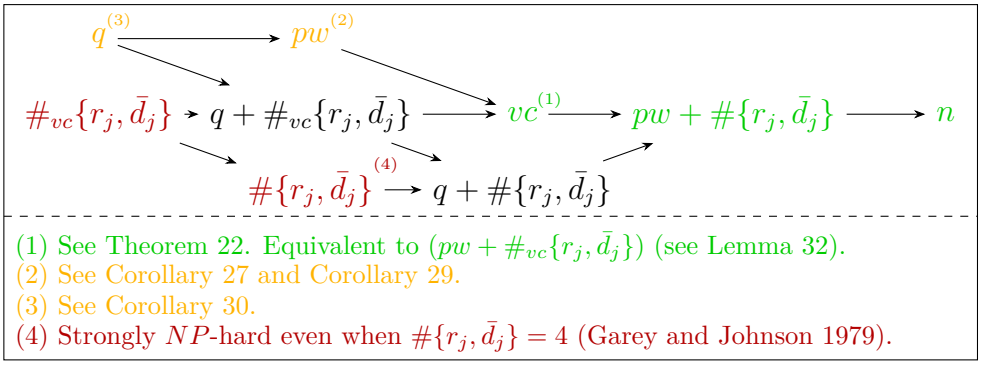


Figure 7: Polynomial kernel status and relations between parameters of $1|prec, r_j, \bar{d}_j|\star$. A parameter is green when there is a polynomial kernel, orange when there is an exponential size kernel and no polynomial kernel (unless $NP \subseteq coNP/poly$), red when there is no kernelization algorithm (unless $P = NP$) and black when there is an exponential size kernel but the existence of a polynomial kernel is open.

stronger than vc while parameter $(q + \#\{r_j, \bar{d}_j\})$ is incomparable to it. Indeed in instances with an empty vertex cover there is no job time window intersection at all, which means that the number of distinct release date and deadline values is at least $n + 1$. So the number of distinct release date and deadline values considered is no longer bounded by a function of vc.

6. Conclusion

In this paper we proved that the single machine scheduling problem with precedence constraints and time windows has no polynomial kernel with respect to the pathwidth of the interval graph induced by time windows (unless $NP \subseteq coNP/poly$). In contrast we provided a polynomial kernel with vertex cover as the parameter. Based on this polynomial kernel, we proposed kernels of exponential size with respect to the pathwidth and the proper level of the interval graph induced by time windows. Such kernelizations give a first insight on reduction rules which might be used to reduce the size of an instance in practice, especially when the number of jobs and the time values are unbounded with respect to the given parameter. Future work would investigate similar single/parallel machine scheduling problems and see in which cases a polynomial kernel can be built.

References

S. Bessy and R. Giroudeau. Parameterized complexity of a coupled-task scheduling problem. *Journal of Scheduling*, 22(3):305–313, June 2019.

H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM Journal on Discrete Mathematics*, 6(2):181–188, 1993.

H. L. Bodlaender and M. van der Wegen. Parameterized Complexity of Scheduling Chains of Jobs with Delays. In *15th Int. Symposium on Parameterized and Exact Computation (IPEC 2020)*, LIPIcs, pages 4:1–4:15, 2020.

H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.

H. Brinkop and K. Jansen. High multiplicity scheduling on uniform machines in fpt-time. *CoRR*, abs/2203.01741, 2022. doi: 10.48550/ARXIV.2203.01741. URL https://doi.org/10.48550/arXiv.2203.01741.

L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, 1997. Asian Logic Conference.

J. Elffers and M. de Weerdt. Scheduling with two non-unit task lengths is np-complete. Technical report, KTH Royal Institute of Technology, 2014. URL http://arxiv.org/abs/1412.3095.

J. Flum and M. Grohe. *Parameterized complexity theory*. Springer, 1998.

A. Frank and É. Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987.

M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

F. Gurski, C. Rehs, and J. Rethmann. Knapsack problems: A parameterized point of view. *Theoretical Computer Science*, 775:93–108, 2019.

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.

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)*, pages 131–134, 2022.

D. Hermelin, S. Karhi, M. Pinedo, and D. Shabtay. New algorithms for minimizing the weighted number of tardy jobs on a single machine. *Annals of Operations Research*, 298(1):271–287, 2021.

K. Jansen, K. Kahler, and E. Zwanger. Exact and approximate high-multiplicity scheduling on identical machines. In I. Finocchi and L. Georgiadis, editors, *Algorithms and Complexity - 14th International Conference, CIAC 2025*, pages 1–17. Springer Nature Switzerland, 2025.

D. Knop and M. Koutecký. Scheduling kernels via configuration LP. In *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, LIPIcs, pages 73:1–73:15, 2022.

D. Knop, M. Koutecký, A. Levin, M. Mnich, and S. Onn. Multitype integer monoid optimization and applications. *CoRR*, abs/1909.07326, 2019. URL http://arxiv.org/abs/1909.07326.

J. B. Lasserre and M. Queyranne. Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling. In *Proc. of the 2nd Integer Programming and Combinatorial Optimization Conf.*, pages 136–149, 1992.

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

M. Mallem, C. Hanen, and A. Munier-Kordon. A new structural parameter on single machine scheduling with release dates and deadlines. In *Combinatorial Optimization*, pages 205–219. Springer Nature Switzerland, 2024.

M. V. Marathe, R. Ravi, and C. P. Rangan. Generalized vertex covering in interval graphs. *Discrete Applied Mathematics*, 39(1):87–93, 1992.

M. Mnich and A. Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1-2):533–562, Dec. 2015.

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 *Discrete Optimization and Operations Research*, pages 105–120. Springer Int. Publishing, 2016.