Parametrized analysis of an enumerative algorithm for a parallel machine scheduling problem

Istenc Tarhan, Jacques Carlier, Claire C. Hanen, Antoine Jouglet, Alix Munier-Kordon

Parametrized analysis of an enumerative algorithm for a parallel machine scheduling problem *

Istenc Tarhan[1,2][0000−0002−1632−884X], Jacques Carlier[2], Claire Hanen[1,3][0000−0003−2482−5042], Antoine Jouglet[2][0000−0001−9251−249X], and Alix Munier Kordon[1][0000−0002−2170−6366]

[1] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
{Istenc.Tarhan,Claire.Hanen,Alix.Munier}@lip6.fr
http://www.lip6.fr
[2] UTC, Heudiasyc,France
{Istenc.Tarhan,Jacques.Carlier,Antoine.Jouglet}@hds.utc.fr
[3] UPL, Université Paris Nanterre, F-92000 Nanterre,France

Abstract. We consider in this paper the scheduling problem defined by a set of dependent jobs with release times and deadlines to be processed by identical parallel machines. This problem is denoted by $P|prec, r_i, d_i|\star$ in the literature. We propose an adaptation of the branch-and-bound algorithm of Demeulemeester and Herroelen algorithm, previously designed for $PS|prec|C_{\max}$, as a dynamic programming scheme, considering release dates and deadlines. New dominance rules are proposed. We establish that the algorithm is fixed-parameter tractable. The two parameters are the pathwidth, which corresponds to the maximum number of overlapping jobs time windows and the maximum execution time of a job. The algorithm is experimented on random instances to show how its practical complexity depends on the parameters.

Keywords: Scheduling · fixed-parameter tractable · release times and deadlines · branch and bound· parallel machines

1 Introduction

Scheduling problems with resource limitation and precedence constraints have many applications in various fields, such as production systems, the use of multi-core parallel machines or the design of embedded systems. Also, many authors have developed exact or approximate algorithms to efficiently solve these problems since the beginning of the sixties. Several books and surveys are dedicated to this class of combinatorial optimization problems [3,5,17].

This paper considers the basic scheduling problem defined by a set of n non-preemptive jobs $\mathcal{T}$ to be executed by m identical machines. Each job $i \in \mathcal{T}$ has a positive integer processing time p_i, an integer release time r_i and a deadline d_i.

* Supported by EASI project, Sorbonne universités

Job i has to be scheduled in such a way that its starting time $s(i)$ verifies $r_i \leq s(i) \leq d_i - p_i$. Each job $i \in \mathcal{T}$ has to be scheduled on one machine, each of which can process at most one job at a time. Lastly, a directed acyclic graph $G = (\mathcal{T}, E)$ defines a set of precedence constraints: for each arc $(i, j) \in E$, the associated constraint is $s(i) + p_i \leq s(j)$. The problem is to find, if possible, a feasible schedule. This problem is denoted by $P|prec, r_j, d_j|\star$ using the Graham notation [12].

This problem is clearly difficult to be solved exactly. Indeed the $P|prec, p_j = 1|C_{\max}$ problem was proved to be NP-complete by Ullman [20]. On the same way, Garey and Johnson [11] established that $P||C_{\max}$ is strongly NP-hard.

The development of fixed-parameter tractable algorithms (FPT algorithms in short) makes it possible to push a little further the study of the existence of an efficient algorithm for certain instances of a difficult problem [6,10]. A fixed-parameter tractable algorithm solves any instance of size n of the problem with parameter k in a time $\mathcal{O}(f(k) \times \text{poly}(n))$, where f is allowed to be a computable superpolynomial function and poly(n) a polynome of n.

The (quite) recent article of Mnich and van Bevern [18] surveys the existence of a FPT algorithm for classical scheduling problems and identifies 15 difficult questions in this context. However, most of the results obtained so far conclude the non-existence of FPT algorithms for the considered parameters.

Enumerative techniques [21] such as Branch-and-Bound methods or dynamic programming approaches are commonly considered for solving exactly combinatorial problems.

Dynamic programming approaches rely on the Bellman's principle of optimality [1] and were developed for different optimization sub-problems (see for example [14]). Their characteristic is that a non trivial upper bound of their worst-time complexity can usually be evaluated. For example, Dolev and Warmuth [9] developed such an algorithm solving $P|prec, p_i = 1|C_{\max}$ with time complexity $\mathcal{O}(n^{h(G)(m-1)+1})$, $h(G)$ being the length of the longest path of the precedence graph. For the same problem, Möhring [19] proposed an algorithm of time in $\mathcal{O}(m^{w(G)})$, where $w(G)$ is the width of the precedence graph. None of them are FPT algorithms.

Van Bevern et al. [2] defined an FPT algorithm for the resource constrained scheduling problem (RCPSP) parameterized by the tuple $(w(G), \lambda)$, where λ is the maximum allowed difference between the earliest starting time and factual starting time of a job. More recently, Munier [16] developed a FPT algorithm for the decision problem $P|prec, r_j, d_j, p_j = 1|\star$. Its parameter μ, called the pathwidth, is the maximal number of overlapping jobs time windows at a single time t i.e. $\mu = \max_{t \in A} |\{i \in \mathcal{T} s.t.\ r_i \leq t < d_i\}|$ with $A = [\min_{i \in \mathcal{T}} r_i, \max_{i \in \mathcal{T}} d_i)$. By augmenting this algorithm with a binary search, a FPT algorithm parameterized by μ is obtained for the two classical optimization problems $P|prec, p_i = 1|C_{\max}$ and $P|prec, p_i = 1|L_{\max}$. This approach was extended by Hanen and Munier in [13] to handle different computation time, but with tuple of parameters $(\mu, p_{\max})$ where $p_{\max} = \max_{i \in \mathcal{T}} p_i$. They also proved that $P2|r_i, d_i|\star$ parameterized by the pathwidth is para-NP-complete as well as $P|prec, r_i, d_i|\star$ param-

eterized by $p_{\max}$; it follows that unless $\mathcal{P} = \mathcal{NP}$, there is no FPT algorithm for $P|prec, r_i, d_i|\star$ parameterized by only one of these parameters.

The enumerative Branch-and-Bound methods are usually considered to develop efficient algorithms for NP-complete scheduling problems. In the nineties, several authors developed Branch-and-Bound methods to handle the resource-constrained scheduling project denoted by $PS|prec|C_{\max}$; see Brucker et al. [4] for the notation and a survey on these methods. The Demeulemeester and Herroelen algorithm [7] is one of the most efficient Branch-and-Bound method to solve efficiently this class of problems [8] without release times and deadlines. To our knowledge, there is no study of the worst-case complexity of this algorithm.

Our first aim was to study whether it would be possible to develop a FPT algorithm more efficient in practice than a Branch-and-Bound algorithm. We thus started from the analysis of the Demeulemeester and Herroelen algorithm [7], in order to evaluate the influence of the parameters (μ, p_{max}) on its complexity. As only semi-active schedules are considered, we discovered that it can be transformed to a search in a graph, instead of a tree, linking this algorithm with a dynamic programming approach. We also established several new dominance rules, and modified the generation of successors of a node of the serach graph to handle release times and deadlines. However, to simplify the complexity study, we did not consider bounding techniques to prune nodes.

This lead us to define a new algorithm called Branch-and-Find algorithm (B&F in short) for our decision problem; We analyse its complexity and show it is FPT for parameters (μ, p_{max}). This algorithm is significantly different from that developed by Hanen and Munier [13]. We also ran some experiments on random instances with controlled parameters that confirms their influence on the practical tractability of the problem. Moreover, we also measure the ratio between the number of nodes of the whole graph in practice and in theory, and show it is very low and decreases with parameter μ.

The remainder of this paper is organized as follows. Section 2 is devoted to the presentation of several general properties of feasible solutions of the problem $P|r_j, d_j|\star$ without precedence constraints. These properties allow setting an upper bound on the number of nodes generated at each step of our algorithm. In Section 3, we present the B&F algorithm and its complexity analysis. In Section 4, computational experiments for the B&F algorithm are shared. We conclude with final remarks in Section 5.

2 Feasibility properties of schedules for jobs with release times and deadlines

This section presents several properties on feasible semi-active schedules for instances of $P|r_j, d_j|C_{\max}$, i.e. we do not consider here precedence constraints. These properties will be considered in Section 3 to bound the complexity of our B&F algorithm.

We illustrate some definitions below with an example. Figure 1 shows a feasible schedule on two processors for the example of Table 1.

$jobs$	1	2	3	4	5	6	7	8	9	10	11
r_i	0	0	4	6	3	7	10	10	15	14	16
d_i	6	6	6	10	10	11	15	15	17	18	18
p_i	3	5	2	4	4	2	4	4	2	3	1

Table 1: Release times, deadlines and processing times for a set of $n = 11$ jobs

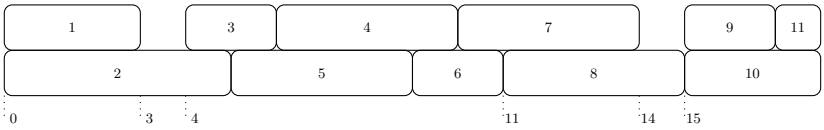

Fig. 1: A feasible schedule on $m = 2$ processors.

A schedule defines for each job i a starting time $s(i)$. It is feasible if no job starts earlier (resp. completes later) than its release time (resp. deadline) and there are not more than m jobs that are in-progress at any time $t \in \mathbb{R}^+$.

Let us consider a schedule S. We sort the starting times of jobs in increasing order (breaking ties with the job index if necessary), and denote by $J_1, J_2, ..., J_n$ the successive jobs such that $s(J_1) \leq s(J_2) \leq ... \leq s(J_n)$. For any value $\alpha \in \{0, \ldots, n\}$, S_α is a partial schedule of S including only its first α jobs $J_1, J_2, ..., J_\alpha$. The schedule S_0 is empty whereas $S_n = S$. When appropriate, S_α is used to refer to the jobs of the corresponding partial schedule. Let us define the time $t(S_\alpha)$ to be the earliest completion time of a job of S_α after $s(J_\alpha)$: $t(S_\alpha) = \min_{\{j \in S_\alpha, s(j)+p_j > s(J_\alpha)\}} s_j + p_j$. We denote by $P(S_\alpha)$ the set of jobs in schedule S_α that complete or are in-progress at time $t(S_\alpha)$. More formally, $P(S_\alpha) = \{i \in S_\alpha, s(i) < t(S_\alpha) \leq s(i) + p_i\}$ for $\alpha \in \{0, ..., n\}$.

For our previous example, the schedule S presented by Figure 1 is associated by the sequence of jobs $(1, 2, 3, 5, 4, 6, 7, 8, 9, 10, 11)$. The partial schedule S_4 is thus associated with the jobs set $\{1, 2, 3, 5\}$. We also get $t(S_4) = s(3) + p_3 = 6$ and $P(S_4) = \{3, 5\}$.

For each value $\alpha \in \{0, \ldots, n\}$, we define Z_α as the first $max\{0, \alpha - \mu\}$ jobs, when jobs are sorted in increasing order of their deadlines, i.e. $d_1 \leq d_2 \leq ... \leq d_n$. More precisely, $\forall \alpha \in \{0, \ldots, n\}$,

$$Z_\alpha = \begin{cases} \emptyset & \text{if } \alpha \leq \mu \\ \{1, 2, \ldots, \alpha - \mu\} \text{ with } d_1 \leq d_2 \leq \ldots \leq d_n & \text{otherwise.} \end{cases}$$

We note that if the deadlines jobs are not all different, there can have several jobs whose deadline is the $(\alpha - \mu)^{\text{th}}$ smallest deadline among all jobs. In such a case, we break ties considering the indexes of the jobs: the job with the smallest index among the jobs having the $(\alpha - \mu)^{\text{th}}$ smallest deadline is added to set Z_α. Thus, the cardinality of set Z_α is always $max\{0, \alpha - \mu\}$.

Similarly, for each value $\alpha \in \{0, \ldots, n\}$ let Z'_α be the set of the first $min\{n - \max\{0, \alpha - \mu\}, 2\mu\}$ jobs that are not included in Z_α when jobs are sorted in

ascending order of their release times such that $r_1 \leq r_2 \leq ... \leq r_n$. Again, we break ties considering the jobs indexes if necessary.

For convenience, we provide the cardinality of sets Z_α and Z'_α for different n and α values in Table 2. Since $|Z'_\alpha| + |Z_\alpha| \leq n$ for each value $\alpha \in \{0, \ldots, n\}$, Z'_α is properly defined.

Table 2: Values $|Z_\alpha| = max\{0, \alpha - \mu\}$, $|Z'_\alpha| = \min\{n - max\{0, \alpha - \mu\}, 2\mu\}$ and $|Z_\alpha| + |Z'_\alpha|$ following n, α and μ.

Case $n < 2\mu$			
Subcase	$\lvert Z_\alpha\rvert$	$\lvert Z'_\alpha\rvert$	$\lvert Z_\alpha\rvert + \lvert Z'_\alpha\rvert$
$\alpha \leq \mu$	0	n	n
$\alpha > \mu$	$\alpha - \mu$	$n - (\alpha - \mu)$	n

Case $n \geq 2\mu$			
Subcase	$\lvert Z_\alpha\rvert$	$\lvert Z'_\alpha\rvert$	$\lvert Z_\alpha\rvert + \lvert Z'_\alpha\rvert$
$\alpha \leq \mu$	0	2μ	2μ
$\mu < \alpha < n - \mu$	$\alpha - \mu$	2μ	$\alpha + \mu$
$\alpha \geq n - \mu$	$\alpha - \mu$	$n - (\alpha - \mu)$	n

Notice that in the example of Table 1 the jobs are indexed by increasing order of deadlines. The list of jobs ordered by increasing release times is $(1, 2, 5, 3, 4, 6, 7, 8, 10, 9, 11)$. Observe that at most 4 intervals (intervals of jobs $\{1, 2, 3, 5\}$ and $\{7, 8, 9, 10\}$) overlap at a same time, so $\mu = 4$. Table 3 presents sets S_α, Z_α and Z'_α for $\alpha \in \{2, 6, 8\}$.

Table 3: Sets S_α, Z_α and Z'_α associated of the example of Table 1 for $\alpha \in \{2, 6, 8\}$.

α	S_α	Z_α	Z'_α
2	$\{1, 2\}$	$\emptyset$	$\{1, 2, 3, 4, 5, 6, 7, 8\}$
6	$\{1, 2, 3, 4, 5, 6\}$	$\{1, 2\}$	$\{3, 4, 5, 6, 7, 8, 9, 10\}$
8	$\{1, 2, 3, 4, 5, 6, 7, 8\}$	$\{1, 2, 3, 4\}$	$\{5, 6, 7, 8, 9, 10, 11\}$

Now let us consider the partial schedule S_6 of the partial schedule depicted in Figure 1. We have $s(1) = s(2) = 0$, $s(3) = 4$, $s(4) = 6$, $s(5) = 5$ and $s(6) = 9$. Moreover, $t(S_6) = s(4) + p_4 = 10$ and $P(S_6) = \{4, 6\}$.

For any value $\alpha \in \{0, \ldots, n\}$, $|S_\alpha \backslash Z_\alpha| \geq |S_\alpha| - |Z_\alpha| \geq \alpha - \max\{0, \alpha - \mu\}$. Next propositions show inclusion properties between the sets S_α, Z_α, Z'_α and $P(S_\alpha)$.

Proposition 1. *For any feasible schedule S, $\forall \alpha \in \{0, \ldots, n\}$, $Z_\alpha \subseteq S_\alpha$.*

Proof. The set $Z_\alpha = \emptyset$ for $\alpha \leq \mu$. Therefore, we consider the case $\alpha > \mu$ and the corresponding schedule S_α. Let time t be the starting time of the last job in schedule S_α, i.e. $t = \max_{j \in S_\alpha} s(j) = s(J_\alpha)$. Thus, for each job $j \in S_\alpha, r_j \leq s(j) \leq t$. Moreover, by definition of S and S_α, jobs in $S \backslash S_\alpha$ can start at time t at the earliest, i.e. $s(j) \geq t, \forall j \in S \backslash S_\alpha$.

By contradiction, assume that there exists a job $i \in Z_\alpha$ with $i \in S \backslash S_\alpha$. Then, $s(i) \geq t$ and thus $d_i > t$. Now, by definition of set Z_α, all jobs in $S_\alpha \backslash Z_\alpha$ have a deadline greater than or equal to d_i, i.e, $d_j \geq d_i > t, \forall j \in S_\alpha \backslash Z_\alpha$. Two cases must be considered:

- If $r_i > t$, then $\forall j \in S_\alpha \backslash Z_\alpha, r_j \leq t < r_i < d_i \leq d_j$. Since $|S_\alpha \backslash Z_\alpha| \geq \mu$, there will be at least μ jobs overlapping with the time window of job i which contradicts the definition of μ.
- Similarly, if $r_i \leq t$, then $\forall j \in (S_\alpha \backslash Z_\alpha) \cup \{i\}, r_j \leq t < d_i \leq d_j$. All these at least $\mu + 1$ jobs overlap at time t, which contradicts the definition of μ. □

Proposition 2. *For any feasible schedule S, $\forall \alpha \in \{0, \ldots, n\}$, $S_\alpha \subseteq Z_\alpha \cup Z'_\alpha$.*

Proof. If $n < 2\mu$ or if $\alpha \geq n - \mu$, $|Z_\alpha \cup Z'_\alpha| = |Z_\alpha| + |Z'_\alpha| = n$ as shown in Table 2 and the proposition holds for these cases. Therefore, let us consider $n \geq 2\mu$ and $\alpha < n - \mu$; in this case $|Z'_\alpha| = 2\mu$.

By contradiction, let us suppose the existence of a job $i \in S_\alpha \backslash (Z_\alpha \cup Z'_\alpha)$. We first prove that $|Z'_\alpha \backslash S_\alpha| \geq \mu + 1$. Indeed, by Proposition 1, $Z_\alpha \subseteq S_\alpha$. Thus, S_α can be partitioned into the 3 sets Z_α, $\{i\}$ and the remaining jobs set R. Here, $|R| = |S_\alpha \backslash (Z_\alpha \cup \{i\})| = |S_\alpha| - |Z_\alpha| - 1 \leq \alpha - (\alpha - \mu) - 1 = \mu - 1$. Now, since $(Z_\alpha \cup \{i\}) \cap Z'_\alpha = \emptyset$, $Z'_\alpha \cap S_\alpha \subseteq R$ and thus $|Z'_\alpha \backslash S_\alpha| \geq |Z'_\alpha| - |R|$. Now, since $|Z'_\alpha| = 2\mu$ and $|R| \leq \mu - 1$, we get $|Z'_\alpha \backslash S_\alpha| \geq \mu + 1$.

Let us denote now by $t = \max_{j \in S_\alpha} s(j) = s(J_\alpha)$ the starting time of the last job in the partial schedule S_α. We prove that, for each $j \in Z'_\alpha \backslash S_\alpha$, $r_j \leq t < d_j$. Indeed, each job $j \in Z'_\alpha \backslash S_\alpha$ verifies $s(j) \geq t$, thus $d_j > t$. Now, since job i is scheduled before or at time t, $r_i \leq s(i) \leq t$. As $i \notin Z_\alpha \cup Z'_\alpha$, $r_i \geq r_j$ for every job $j \in Z'_\alpha \backslash S_\alpha$, we get $r_j \leq r_i \leq t$.

Thus all the time windows of jobs in $Z'_\alpha \backslash S_\alpha$ overlap during the interval $(t, t + 1)$. Since $|Z'_\alpha \backslash S_\alpha| \geq \mu + 1$, it contradicts the definition of the interval parameter μ.

□

Proposition 3. *For any feasible schedule S, $\forall \alpha \in \{0, \ldots, n\}$, $P(S_\alpha) \cap Z_\alpha = \emptyset$, and thus $Z_\alpha \subseteq S_\alpha \backslash P(S_\alpha)$ and $P(S_\alpha) \subseteq Z'_\alpha$.*

Proof. Since $Z_\alpha = \emptyset$ for $\alpha \leq \mu$, we only consider $\alpha > \mu$. In this case, by Proposition 1, $|S_\alpha \backslash Z_\alpha| = \mu$.

By contradiction, let us consider a job $i \in P(S_\alpha) \cap Z_\alpha$. Since $i \in P(S_\alpha)$, i is either in-progress or completes at time $t(S_\alpha)$ and therefore $d_i \geq t(S_\alpha) > r_i$. Now, as $i \in Z_\alpha$, every job $j \in S_\alpha \backslash Z_\alpha$ verifies $d_j \geq d_i \geq t(S_\alpha)$. Moreover $r_j \leq s(j) \leq s(J_\alpha) < t(S_\alpha)$.

Thus, every job $j \in \{i\} \cup (S_\alpha \backslash Z_\alpha)$ verifies $r_j < t(S_\alpha) \leq d_j$; we deduce that there are at least $\mu + 1$ jobs which time window intersects any $t \in (t(S_\alpha) - 1, t(S_\alpha))$, a contradiction with the definition of μ.

Lastly, by Proposition 1, $Z_\alpha \subseteq S_\alpha$, and thus $Z_\alpha \subseteq S_\alpha \backslash P(S_\alpha)$. By Proposition 2 this implies that $P(S_\alpha) \subset Z'_\alpha$, which achieves the proof. □

3 Branch-and-Find (B&F) algorithm

It is inspired from a branch-and-bound algorithm proposed by [7] that minimizes the makespan for the RCPSP. Here, we consider the decision problem $P|prec, r_i, d_i|\star$ which includes release times and deadlines for jobs unlike the RCPSP problem. The B&F algorithm differentiates from the branch-and-bound algorithm in several ways: a search graph is generated instead of a search tree , release times and deadlines are considered while generating successors of a nod, dominance based on properties of section 2 are used, and no bounding technique is applied to ease the complexity analysis. On another hand, the dominance properties and node definition on which the branch-and-bound [7] is based are used. Branching principles are also similar but extended to handle time windows.

Let the relevant search graph denoted by $\mathcal{G}(\mathcal{V}, \mathcal{A})$. In Section 3.1, nodes of the search graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$ are defined and some dominance properties presented. Section 3.2 analyses the worst case complexity of finding an undominated node. In Section 3.3, the construction of the search graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$ is explained. Finally, in Section 3.4, the overall complexity of the B&F algorithm is analyzed.

Without loss of generality, we assume from now that $m \leq \mu$, otherwise the earliest schedule would fit on the m processors and the decision problem would be trivial.

In the rest of the section we illustrate some of the notions with the instance described by the set of jobs of Table 1, to which we add a precedence graph shown in Figure 2. The schedule shown in Figure 1 satisfies these precedence constraints.

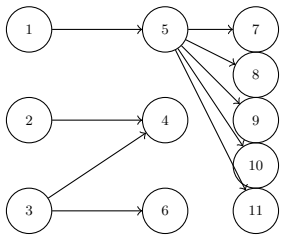

Fig. 2: A precedence graph for the set of jobs of Table 1

3.1 Nodes definitions and dominance properties

A node v is a quadruplet $v = (V, t, P, M)$ where $V \subseteq \mathcal{T}$ is a set of jobs, $t \in \mathbb{N}$ is a date, $P \subseteq V$, and $M \in \mathbb{N}^{|P|}$ is a a vector indexed following P. Hereafter, we use $V(v)$, $t(v)$, $P(v)$, and $M(v)$ to refer to set V, time moment t, set P and function M. The level of a node v is the number of jobs in $V(v)$.

Definition 1. *A node $v = (V, t, P, M)$ is said to be partially feasible if there exists a feasible schedule σ of jobs from V such that*

1. *Every job $i \in V \backslash P$ is completed before t, i.e. $\sigma(i) + p_i < t$;*
2. *Every job $i \in P$ starts before t: $\sigma(i) < t$ and is completed at time $M_i \geq t$: i.e. $M_i = \sigma(i) + p_i \geq t$.*

Note that σ is a partial schedule of $\mathcal{T}$ since it concerns only jobs from $V(v)$. Moreover, the node v alone is clearly not sufficient to define σ. We also observe that jobs of P are either in progress or completed at time t and thus $|P| \leq m$.

Several enumerative algorithms build semi-schedules for subsets of jobs $V \subseteq \mathcal{T}$ until reaching a complete schedule of S. The next definition introduces the notion of fully feasible node in coherence with the Demeulemeester and Herroelen algorithm [7] where some jobs in P can be delayed.

Definition 2. *A partially feasible node $v = (V, t, P, M)$ is said to be fully feasible if there exists a feasible schedule S of jobs from $\mathcal{T}$ such that*

1. *Every job $i \in V \backslash P$ is completed before t, i.e. $s(i) + p_i < t$;*
2. *Every job in $i \in \mathcal{T} \backslash V$ starts after time t, i.e. $s(i) \geq t$;*
3. *Every job $i \in P$ such that $M_i = t$ starts at time $s(i) = M_i - p_i$. Every job $i \in P$ with $M_i > t$ starts either at time $M_i - p_i$ or at time $s(i) \geq t$.*

Again the associated schedule S is not stored with any feasible node v. Testing that a node is feasible is thus uneasy. Next proposition will allow us to limit the exploration of the nodes to a more affordable class.

Proposition 4. *Assume that S is a feasible schedule, and let consider the partial schedules S_α, $\alpha \in \{0, \ldots, n\}$ as defined in Section 2. Consider the set of indexes $\mathcal{A} = \{\alpha \in \{1, \ldots, n\}, t(S_{\alpha+1}) > t(S_\alpha)\}$. Then, $\mathcal{A} \neq \emptyset$. Moreover, $\forall \alpha \in \mathcal{A}$, node $v = (S_\alpha, t(S_\alpha), P(S_\alpha), M)$ with $M_i = s(i) + p_i$ for every job $i \in P(S_\alpha)$ is a fully feasible node.*

Proof. Let us consider that jobs of $\mathcal{T}$ are numbered $J_1, \ldots J_n$ according to S such that $s(J_1) \leq s(J_2) \leq ... \leq s(J_n)$. $\mathcal{A}$ is not empty, otherwise for $\alpha = \mu + 1$, we would have all jobs $J_1, \ldots, J_{\mu+1}$ completed at $t(S_{\mu+1})$ or in progress at that time, a contradiction with the definition of μ. Let us consider now $\alpha \in \mathcal{A}$ and node $v = (S_\alpha, t(S_\alpha), P(S_\alpha), M)$. We first show that v is partially feasible by considering for σ the restriction of S to S_α.

1. Consider a job $i \in S_\alpha \backslash P(S_\alpha)$. By definition of S_α and $P(S_\alpha)$, $s(i) + p_i < t(S_\alpha)$ and the first item of Definition 1 is verified;

2. If now $i \in P(S_\alpha)$, then $s(i) < t(S_\alpha) \leq s(i) + p_i = M_i$, so that the last item is also verified.

Now, we are ready to prove that v is a fully feasible node. Indeed, we observe that S verifies the two items 1 and 3 of Definition 2. Let us show that if $t(S_{\alpha+1}) > t(S_\alpha)$ then $s(J_{\alpha+1}) \geq t(S_\alpha)$. By contradiction assume that $s(J_{\alpha+1}) < t(S_\alpha)$. We know that there is a job i in $P(S_\alpha)$ that completes at time $t(S_\alpha)$. We also know that $s(J_{\alpha+1}) \geq s(J_\alpha)$. This implies that i starts before and completes after $s(J_{\alpha+1})$, and thus it is included in the computation of $t(S_{\alpha+1})$, the least completion time after $s(J_{\alpha+1})$. So $t(S_{\alpha+1}) \leq t(S_\alpha)$ a contradiction. This implies that for any job $j \notin S_\alpha$, $s(i) \geq t(S_{\alpha+1}) \geq t(S_\alpha)$. Hence the second item of Definition 2 holds, and v is a fully feasible node.

Let us consider a feasible schedule S and a node $v = (S_\alpha, t(S_\alpha), P(S_\alpha), M)$ as defined by Propositions 4. The node v is feasible and by Propositions 2 and 3, $Z_\alpha \subseteq S_\alpha \backslash P(S_\alpha)$ and $S_\alpha \subseteq Z_\alpha \cup Z'_\alpha$. These two last conditions will be considered to define the class of nodes that are considered in our algorithm:

Definition 3. *A partially feasible node v is admissible if $Z_\alpha \subseteq V(s) \backslash P(s)$ and $V(s) \subseteq Z_\alpha \cup Z'_\alpha$ for $\alpha = |V(s)|$.*

The following dominance property proved by Demeulemeester and Herroelen [7] allows the number of nodes considered in an enumerative algorithm to be reduced.

Proposition 5. *[7] Consider two partially feasible nodes v and v' such that $V(v) = V(v')$, and such that $t(v') \geq \max_{i \in P(v) \backslash P(v')}(M_i(v))$ and $\forall i \in P(v) \cap P(v'), M_i(v) \leq M_i(v')$. If v' is fully feasible, then v is fully feasible either.*

Following the previous proposition, we now define the notion of dominance between nodes:

Definition 4. *Let v and v' be two nodes. The node v dominates v' if $V(v) = V(v'), t(v') \geq t(v), t(v') \geq \max_{i \in P(v) \backslash P(v')}(M_i(v))$ and $\forall i \in P(v) \cap P(v'), M_i(v) \leq M_i(v')$. A set of nodes $\mathcal{N}$ is said undominated if there is no couple of nodes $(v, v') \in \mathcal{N}^2$ such that v dominates v'.*

The next lemma bounds the number of admissible undominated nodes:

Lemma 1. *For any value $\alpha \in \{0, \ldots, n\}$, there are at most $\binom{2\mu}{\mu}$ different sets V of α jobs associated to an an undominated admissible node (i.e. such that there exists an undominated admissible node v with $V = V(v)$). Moreover, for a given job set V, the number of undominated admissible nodes v such that $V(v) = V$ is bounded by $(2 \times p_{max})^\mu$. The number of admissible undominated nodes of level α is bounded by $f(\mu, p_{max})$ with $f(\mu, p_{max}) = \binom{2\mu}{\mu} \times (2 \times p_{max})^\mu$.*

Proof. Assume that the node v is admissible of level α, i.e. $|V(v)| = \alpha$. The sets Z_α and Z'_α are fixed.

1. By Definition 3, $Z_\alpha \subseteq V(v)\backslash P(v) \subseteq V(v)$, thus $|V(v)\backslash Z_\alpha| = |V(v)| - |Z_\alpha| = \alpha - \max\{0, \alpha - \mu\} \leq \mu$. Moreover, $V(v) \subseteq Z_\alpha \cup Z'_\alpha$ and by definition $|Z'_\alpha| \leq 2\mu$. Since $V(v)$ is built from at most μ elements in Z'_α, the number of possibilities for $V(v)$ is bounded by $\binom{2\mu}{\mu}$, which corresponds to the first part of the lemma.
2. On the same way, by Definition 3, $P(v) \subseteq V(v)\backslash Z_\alpha$; since $|V(v)\backslash Z_\alpha| \leq \mu$, and $P(v)$ contains at most m elements, the total number of possibilities for $P(v)$ when $V(v)$ is fixed is $\sum_{i=0}^{i=m} \binom{\mu}{i} \leq 2^\mu$; since $m \leq \mu$;
3. Let us suppose now that $V(v)$ and $P(v)$ are fixed. Then, if $t(v)$ is fixed, each job $i \in P(v)$ must have its completion time $M_i(v)$ in $\{t(v), \ldots, t(v) + p_i - 1\}$. Thus, $(p_{max})^{|P(v)|}$ is an upper bound of the total number of possible $M(v)$ vectors. Moreover, by Definition 4, for any fixed $M(v)$ vector, only the smaller possible value of $t(v)$ should be considered.
4. So, for a given $V(v)$, the number of undominated admissible nodes is bounded by $(2 \times p_{max})^\mu$

Thus, the total number of undominated admissible nodes of level α is bounded by $A = \binom{2\mu}{\mu} \times \sum_{i=0}^{m} (\binom{\mu}{i}(p_{max})^i) \leq \binom{2\mu}{\mu} \times (p_{max})^m \times \sum_{i=0}^{m} \binom{\mu}{i}$. Now, since $m \leq \mu$ and $\sum_{i=0}^{\mu} \binom{\mu}{i} = 2^\mu$, we get $A \leq \binom{2\mu}{\mu} \times (p_{max})^\mu \times 2^\mu$, which achieves the proof. □

3.2 Management of the undominated sets of admissible nodes

For any $\alpha \in \{0, \ldots, n\}$, we define $\mathcal{V}_\alpha$ as the set of undominated admissible nodes of level α, i.e. $\forall v \in \mathcal{V}_\alpha$, $|V(v)| = \alpha$. Algorithm 1 describes function *AddDiscardOrReplace*$(v, \mathcal{V}_\alpha)$ that considers adding node v to set $\mathcal{V}_\alpha$ while preserving the undominance property of set $\mathcal{V}_\alpha$. It considers whether node v is dominated by another node $u \in \mathcal{V}_\alpha$ or not. If so, *AddDiscardOrReplace*$(v, \mathcal{V}_\alpha)$ returns false and the node v is not added to $\mathcal{V}_\alpha$ (lines 4-5). Otherwise, v will be added to $\mathcal{V}_\alpha$ and nodes that are dominated by v are removed from $\mathcal{V}_\alpha$ (lines 6-10).

Next Lemma analyses the time complexity of Algorithm 1.

Lemma 2. *The time complexity of the function AddDiscardOrReplace*$(v, \mathcal{V}_\alpha)$ *is* $\mathcal{O}(g(\mu, p_{max}))$ *where* $g(\mu, p_{max}) = \mu \times (\mu \ln(\mu) + (2p_{max})^\mu)$.

Proof. Let us assume that the nodes sharing the same set V are stored in a separate container in $\mathcal{V}_\alpha$. Let $V(C)$ denote the set V of the nodes stored in the container C. The set of containers can be stored using AVL trees to speed-up the initialization of Q at line 2. Since all the nodes stored in $\mathcal{V}_\alpha$ are admissible, for every $v \in \mathcal{V}_\alpha$, $Z_\alpha \subseteq V(v)$ and $V(v)\backslash Z_\alpha \subseteq Z'_\alpha$ by Definition 3. So, we only consider jobs in Z'_α to differentiate the containers. Moreover, $|Z'_\alpha| \leq 2\mu$, thus 2μ bits $b_1 \ldots b_{2\mu}$ are required for determining a key associated to $V(Q)$: $b_j = 1$ if and only if the associated job of Z'_α is in $V(Q)$.

Algorithm 1 *AddDiscardOrReplace*$(v, \mathcal{V}_\alpha)$

Require: v is a node of level α; $\mathcal{V}_\alpha$ is a set of undominated admissible nodes of level α.
Ensure: Add v to $\mathcal{V}_\alpha$ if possible and maintain that $\mathcal{V}_\alpha$ is an undominated set of nodes;
Returns false if v is not added to $\mathcal{V}_\alpha$, v otherwise.
1: Set $D = \emptyset$
2: Find $Q = \{u \in \mathcal{V}_\alpha, V(u) = V(v)\}$
3: **for** each node $u \in Q$ **do**
4: **if** u dominates v **then**
5: **return** false
6: **else if** v dominates u **then**
7: $D = D \cup \{u\}$
8: **end if**
9: **end for**
10: $\mathcal{V}_\alpha = (\mathcal{V}_\alpha - D) \cup \{v\}$
11: **return** v

For a fixed value α, there can be at most $\binom{2\mu}{\mu}$ different possible set V as shown in Lemma 1. Therefore, $\mathcal{V}_\alpha$ is partitioned by at most $\binom{2\mu}{\mu}$ different containers. The AVL trees as described before allows to get the container Q in time complexity $O(\mu \times \ln \binom{2\mu}{\mu}) = O(\mu \times \mu \times \ln(\mu))$ since $\binom{2\mu}{\mu} \leq (2\mu)^\mu$.

Testing that a node u dominates v is in time complexity $\mathcal{O}(m)$, and thus $\mathcal{O}(\mu)$. We deduce that the time complexity of a single iteration of the loop at lines 3-9 is $\mathcal{O}(\mu)$.

Lastly, as shown in Lemma 1, the number of undominated admissible nodes in the container Q is $\mathcal{O}((2 \times p_{max})^\mu)$. Therefore, the complexity of Algorithm 1 is $\mathcal{O}(\mu \times (\mu \ln(\mu) + (2 \times p_{max})^\mu))$ which proves the lemma. □

3.3 Construction of the search graph

The Branch-and-Find algorithm computes the search graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$ and is expressed by Algorithm 2. The main ideas of [7] were adapted to handle release times and deadlines. The B&F algorithm starts with the root node $v_0 = (\emptyset, 0, \emptyset, \emptyset)$ and gradually builds the search graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$.

At line 1 of Algorithm 2, the subsets of admissible undominated nodes $\mathcal{V}_0, \ldots, \mathcal{V}_n$ and the set of arcs $\mathcal{A}$ are initialized. The algorithm is composed by 3 nested loops; the first two loops on respectively lines 2 and 3 iterate on the admissible nodes in $\mathcal{V}_0, \ldots, \mathcal{V}_{n-1}$. The set $R(v)$ computed by the function *SetCandidateNewJobs*(v) at line 4 is the set of jobs to be considered for building the nodes u successors of v in $\mathcal{G}$, i.e. $V(u) \subseteq V(v) \cup R(v)$. The value $t_{min} \geq t(v)$ is the minimum time at which a new job can be performed.

The inner loop at lines 5-13 iterates on every non empty maximal subset C of $R(v)$. The function *NewNode* at line 6 returns a new admissible node built from C and t_{min} if it is possible, or false otherwise. If a new node v' from node v, set C and time t_{min} is generated, we call *AddDiscardOrReplace*$(v', \mathcal{V}_{\alpha'})$ where $\alpha' = |V(v')|$ at line 8. As seen in Subsection 3.2, this function returns false if

Algorithm 2 The Branch-and-Find algorithm

Require: An instance $\mathcal{I}$ of n jobs of $P|prec, r_j, d_j|\star$
Ensure: The associated search graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$
1: $\mathcal{V}_0 = \{(\emptyset, 0, \emptyset, \emptyset)\}$, $\mathcal{V}_\alpha = \emptyset$ for $\alpha \in \{1, \cdots, n\}$, $\mathcal{A} = \emptyset$
2: **for** $\alpha \in \{0, \ldots, n-1\}$ **do**
3: **for** each node $v \in \mathcal{V}_\alpha$ **do**
4: $R(v), t_{min} \leftarrow SetCandidateNewJobs(v)$
5: **for** each subset $C \neq \emptyset$ of $R(v)$ s.t. $|C| = \min(m, |R(v)|)$ **do**
6: $v' \leftarrow NewNode(v, t_{min}, C)$
7: **if** $v' \neq false$ **then**
8: $u \leftarrow AddDiscardOrReplace(v', \mathcal{V}_{\alpha'})$ where $\alpha' = |V(v')|$
9: **if** $u \neq false$ **then**
10: $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, u)\}$
11: **end if**
12: **end if**
13: **end for**
14: **end for**
15: **end for**
16: **return** $\mathcal{G}(\mathcal{V}, \mathcal{A})$ where $\mathcal{V} = \bigcup_{\alpha=0}^{n} \mathcal{V}_\alpha$

there exists another node in $\mathcal{V}_{\alpha'}$ that is dominating node v. In this case, we do not consider the new node v'. Otherwise, $AddDiscardOrReplace(v', \mathcal{V}_{\alpha'})$ deletes the nodes in $\mathcal{V}_{\alpha'}$ that are dominated by v' and returns v'. Note that we consider that as soon as a node is deleted from $\mathcal{V}$, all its adjacent arcs are automatically removed from $\mathcal{A}$.

Computation of the set of candidate jobs $R(v)$ and the time instant t_{min}: let us consider an admissible node v and a time instant $t \geq t(v)$. For any job i we denote by $\Gamma^{-\star}(i)$ the set of all ancestors of i in the precedence graph. We then define the following subsets of jobs:

- $IP(v,t) = \{i \in P(v), M_i(v) > t\}$ is the set of jobs of $P(v)$ that are in progress at time t;
- $E(v,t) = \{i \in \mathcal{T} \backslash V(v), \Gamma^{-\star}(i) \subseteq V(v) \backslash IP(v,t)\}$ is the set of eligible jobs at time t following v, i.e. these jobs are not in $V(v)$ and all of their ancestors are in $V(v)$ and completed by time t;
- $D(v,t) = E(v,t) \cap \{i \in \mathcal{T}, r_i \leq t\}$, the set of eligible jobs that are released before or at time t.

Algorithm 3 describes the determination of the set of jobs to be considered to build successors u of node v in the search graph and their time $t(u) = t_{min}$.

The B&F algorithm ensures that when a new node u is generated from a node v, the cardinality of $V(u)$ is greater than the cardinality of $V(v)$.

By definition of node v, jobs that are not in $V(v)$ can be started at time $t(v)$ at the earliest in the schedules represented by node v. Let us define $t_{min} \geq t(v)$ as the minimum value such that a new job j from $\mathcal{T} \backslash V(v)$ may be executed

Algorithm 3 *SetCandidateNewJobs*(v)

Require: v is an admissible node
Ensure: The set of jobs $R(v) \subseteq (\mathcal{T} \backslash V(v) \cup P(v)$, the time instant t_{min}
1: $t_{min} = \min\{t : t \geq t(v), D(v,t) \neq \emptyset\}$
2: $ect^\star(v, t_{min}) = \min\{ect(v, i, t_{min}), i \in E(v, t_{min}) \cup IP(v, t_{min})\}$
3: **if** there exists a job $i \in \mathcal{T} \backslash V(s)$ s.t. $ect(v, i, t_{min}) > d_i$ **then**
4: **return** $(\emptyset, 0)$
5: **end if**
6: $R(v) = IP(v, t_{min}) \cup (E(v, t_{min}) \cap \{i \in \mathcal{T}, r_i < ect^\star(v, t_{min})\})$
7: **return** $(R(v), t_{min})$

in a feasible schedule represented by the admissible node v. We observe that $j \in D(v, t_{min})$ and thus t_{min} can be defined following line 1 of Algorithm 3.

Now, for any job $i \in IP(v,t) \cup (\mathcal{T} \backslash V(v))$, let us define the lower bound of the completion time of i following v as

$$ect(v,i,t) = \begin{cases} M_i(v) & \text{if } i \in IP(v,t) \\ \max\{t, r_i\} + p_i & \text{if } i \in \mathcal{T} \backslash V(v). \end{cases}$$

By definition of v, and t_{min} any job $i \in \mathcal{T} \backslash V(s)$ starts not earlier than t_{min}, thus $ect(v, i, t_{min})$ is a lower bound of the completion time of i. If $ect(v, i, t_{min}) > d_i$, v cannot lead to a feasible schedule and the function returns $\emptyset$ (line 4).

To avoid overlooking jobs that are not released immediately at t_{min} but very close in time, $ect^\star(v, t_{min})$ is defined as the earliest possible completion time among the jobs in $E(v, t_{min}) \cup IP(v, t_{min})$; then, the set $R(v)$ includes any eligible job at time t_{min} that is released strictly before $ect^\star(v, t_{min})$ (lines 6 of Algorithm 3).

One can observe that jobs in $\mathcal{T} \backslash (V(v) \cup R(v))$ cannot start their execution following v in the interval $[t_{min}, ect^\star(v, t_{min}))$. Indeed, considering job $i \in \mathcal{T} \backslash (V(v) \cup R(v))$, if $i \in E(v, t_{min})$, then $r_i \geq ect^\star(v, t_{min})$. Otherwise, $i \notin E(v, t_{min})$ and thus i has at least one predecessor j which is not completed at time t_{min}. By definition of $ect^\star(v, t_{min})$, no job is completed in the interval $[t_{min}, ect^\star(v, t_{min}))$ and $ect(v, j, t_{min}) \geq ect^\star(v, t_{min})$, thus i cannot start its execution before $t = ect^\star(v, t_{min})$.

Lastly, no job of $R(v)$ will complete earlier than $ect^\star(v, t_{min})$. Thus, it is sufficient to consider only the jobs in set $R(v)$ while generating new nodes from node v.

Let us consider, for the example of Table 1 and Figure 2, the node $v = (\{1,2,3,4,5,6\}, 10, \{4,6\}, M)$ with $M_4 = 10, M_6 = 11$. According to the precedence graph, all the predecessors of the remaining jobs are completed at 10, and at least one of them has a release time not greater than 10. So that $t_{min} = 10$. The earliest completion time is then $ect^\star(v, 10) = 11$ the completion time of job 6. Hence $R(v)$ is the set of eligible jobs with release time less than 11 plus job 6, so $R(v) = \{6, 7, 8\}$. Similarly if we consider the node $u = (\{1,2\}, 3, \{1,2\}, M)$ with $M_1 = 3, M_2 = 5$, then job 5 is available at time 3 so that $t_{min} = 3$,

and the earliest completion time of a job in progress at t_{min} is 5. So the set $R(u) = \{2, 3, 5\}$ since job 3 has release time less than 5.

Computation of a new node: Algorithm 4 presents the function $NewNode$ that returns, if it possible, a new admissible node u built by considering all the jobs from C.

For any $t \geq t(v)$, we define the set $X(v,t) \subseteq E(v,t)$ of eligible jobs j released before t, and such that no predecessor of j ends at $t(v)$. i.e. $X(v,t) = \{j \in E(v,t), r_j < t \text{ and } \forall i \in \Gamma^{-\star}(j), i \notin P(v)\}$. We also note $\overline{X}(v,t) = E(v,t) \backslash X(v,t)$.

Now, by definition of $R(v)$, any set $C \subseteq R(v)$ can be partitioned into three subsets $C_P \subseteq IP(v, t_{min})$, $C_X \subseteq X(v, t_{min})$ and $C_{\overline{X}} \subseteq \overline{X}(v, t_{min})$. In any semi-active schedule, jobs from C_X cannot be executed at time t_{min} or later with an idle slot just before, since all their predecessors are completed at a times $t' < t(v) \leq t_{min}$. Let Q define the set of jobs in $P(v)$ that complete at time t_{min} (see line 2 of Algorithm 4). If $|Q| < |C_X|$, then there is at least one job in C_X that will be executed at time t_{min} with an idle time in the previous time slot $[t_{min} - 1, t_{min})$, and the schedule associated with the new node will not be semi-active. Thus, the set C must be discarded and Algorithm 4 returns false (lines 3-4).

Algorithm 4 $NewNode(v, t_{min}, C)$

Require: v is an admissible node, t the time instant of the new node, $C \subset R(v)$ the set of new jobs added to $V(v)$ for the new node.
Ensure: an admissible new node u if it is possible, false otherwise.
1: Let $C = C_P \cup C_X \cup C_{\overline{X}}$ with $C_P \subseteq IP(v, t_{min})$, $C_X \subseteq X(v, t_{min})$ and $C_{\overline{X}} \subseteq \overline{X}(v, t_{min})$
2: Let $Q = \{i \in P(v), M_i(v) = t_{min}\}$
3: **if** $|Q| < |C_X|$ **then return** false
4: **end if**
5: Set $V(u) = (V(v) \backslash IP(v, t_{min})) \cup C$ and $\beta = |V(u)|$
6: **if** $Z_\beta \not\subseteq V(u)$ or $V(u) \not\subseteq Z_\beta \cup Z'_\beta$ **then return** false
7: **end if**
8: $P(u) = C$; $\forall j \in C$, $M_j(u) = ect(v, j, t_{min})$; $t(u) = \min_{j \in C} M_j(u)$
9: **return** node u

The set of nodes $V(u)$ and its cardinality β are defined at line 5. Lines 6-7 discard u if it is not admissible (see Definition 3). Line 8 defines $P(u)$, $M(u)$ and $t(u)$ as well.

In our example, starting from node $u = (\{1,2\}, 3, \{1,2\}, M)$ with $M_1 = 3, M_2 = 5$ we would try three successors with sets $C_1 = \{2,3\}, C_2 = \{2,5\}, C_3 = \{3,5\}$, leading to the quadruplets $v_1 = (\{1,2,3\}, 5, \{2,3\}, M_2 = 5, M_3 = 6), v_2 = (\{1,2,5\}, 5, \{2,5\}, M_2 = 5, M_5 = 7), v_3 = (\{1,3,5\}, 6, \{3,5\}, M_3 = 6, M_5 = 7)$. Then, v_2 will be discarded because if job 3 is scheduled at 5 it misses its deadline. Similarly, if 2 is scheduled after 6, it will miss its deadline, so v_3 is discarded too.

3.4 Complexity analysis of the Branch-and-Find algorithm

In this section we prove that the algorithm is fixed parameter tractable for the parameters μ, p_{max}.

Proposition 6. *For any admissible node $v \in \mathcal{V}$, $|R(v)| \leq \mu$.*

Proof. By definition of $R(v)$, all jobs in $R(v)$ are schedulable in the interval $[ect^\star(v, t_{min}) - 1, ect^\star(v, t_{min}))$, thus their time windows overlap, and the proposition holds. □

Proposition 7. *The time complexity of the function $SetCandidateNewJobs(v)$ (see Algorithm 3) is $\mathcal{O}(n^2 \times \mu)$ if no specific data structure is used.*

Proof. $SetCandidateNewJobs(v)$ requires first to find t_{min}(see line 1). To this purpose we can observe that t_{min} is either $t(v)$, a release time $r_i > t(v)$ or a value $M_i(v)$ for $i \in IP(v, t(v))$. Let us denote by $\Delta_1, \ldots, \Delta_k$ these values following increasing order i.e. $\Delta_1 < \Delta_2 < \ldots \Delta_k$, $\Delta_0 = t(v)$ and $k \leq m$ (with $m \leq \mu$).

- For each value Δ_b, $b \in \{0, \ldots, k\}$ sets $E(v, \Delta_i)$ can be computed in time complexity $\mathcal{O}(n^2)$: we consider at most n jobs to check if their ancestors are completed by time Δ_i and there can be at most $n-1$ ancestors for a given job. Then, sets $D(v, \Delta_i)$ can be deduced in time complexity $\mathcal{O}(n)$. The overall computation of these sets is then in time $\mathcal{O}(n^2 \times \mu)$;
- Let $b^\star$ be the minimum value in $b \in \{0, \ldots, k\}$ such that $D(v, \Delta_b) \neq \emptyset$. If $b^\star = 0$, $t_{min} = t(v)$. Else, we get $\Delta_{b^\star - 1} < t_{min} \leq \Delta_{b^\star})$. We define the set $A = E(v, \Delta_{b^\star - 1}) \cap D(v, \Delta_{b^\star}$. If $A = \emptyset$, then jobs in $D(v, \Delta_{b^\star})$ are not eligible at time $\Delta_{b^\star - 1}$, and thus $t_{min} = \Delta_{b^\star}$. Otherwise, $t_{min} = \min_{i \in A} r_i$. Without any specific data structure, the time complexity is in $\mathcal{O}(m + \mu)$.
- We conclude that the computation of t_{min} is in time complexity $\mathcal{O}(n^2 \times \mu)$.

Once t_{min} and set $D(v, t_{min})$ are fixed, the computation of $ect^\star(v, t_{min})$ at line 3 and of $R(v)$ take both $\mathcal{O}(n)$. The total complexity of $SetCandidateNewJobs(v)$ is thus $O(n^2 \times \mu)$. □

Proposition 8. *The time complexity of the function $NewNode(v, t_{min}, C)$ (see Algorithm 4) is $\mathcal{O}(n)$.*

Proof. The time complexity of the instructions at lines 1 and 5 of Algorithm 4 are $\mathcal{O}(n)$, while those at lines 2 and 8 are $\mathcal{O}(m)$. Since $m \leq n$, the whole time complexity of this algorithm is $\mathcal{O}(n)$, which proves the lemma. □

Our main theorem follows:

Theorem 1. *The Branch-and-Find algorithm (see Algorithm 2) is an FPT algorithm of time complexity*

$$\mathcal{O}\left(n^3 \times \mu f(\mu, p_{max}) + n^2 \times h(\mu, p_{max}) + n \times g(\mu, p_{max}) h(\mu, p_{max})\right)$$

with $f(\mu, p_{max}) = \binom{2\mu}{\mu} \times p_{max}^{\mu} \times 2^{\mu}$, $g(\mu, p_{max}) = \mu(\mu ln(\mu) + (2 \times p_{max})^{\mu})$ *and* $h(\mu, p_{max}) = \binom{\mu}{\lceil \mu/2 \rceil} \times f(\mu, p_{max})$.

Proof. Algorithm 2 consists of three nested loops starting respectively at lines 2, 3 and 5.

1. By Proposition 8 and Lemma 2, the time complexity of the loop body of the inner loop (lines 6-12) is $\mathcal{O}(n+g(\mu, p_{max}))$. Now, by Proposition 6, $|R(v)| \leq \mu$. Thus the total number of sets $R(v)$ is $\binom{\mu}{m} \leq \binom{\mu}{\lceil \mu/2 \rceil}$. The complexity of one execution of the inner loop (line 5-13) is in time complexity $\mathcal{O}(A)$ with $A = (n + g(\mu, p_{max})) \times \binom{\mu}{\lceil \mu/2 \rceil}$;
2. By Proposition 7, the time complexity of the loop body of the intermediate loop (lines 4-13) is $\mathcal{O}(A + n^2\mu)$. Following Lemma 1, the number of iterations of the intermediate loop (line 3-14) is bounded by $f(\mu, p_{max})$, thus its complexity belongs to $\mathcal{O}((A + n^2\mu) \times f(\mu, p_{max}))$;
3. Lastly, the outer loop (lines 2-14) is executed n times, thus the overall time complexity is $\mathcal{O}((A + n^2\mu) \times f(\mu, p_{max}) \times n)$.

Replacing A by its value, we get the theorem. □

4 Computational experiments

4.1 Data generation

We develop a problem instance generator that for given values of n, μ, m, γ^- and γ^+, guarantees to produce an instance for which i) the number of jobs and machines are n and m, respectively, ii) the maximum number of jobs with overlapping time windows is μ, iii) the maximum processing time among all jobs is p_{max} and iv) the maximum number of predecessors and successors a job has is γ^- and γ^+, respectively.

Algorithm 5 *InstanceGenerator*$(n, m, \mu, p_{max}, \gamma^-, \gamma^+, \rho)$

1: $\mathcal{T} = \emptyset$
2: *GenerateFirstμJobs*$(\mathcal{T}, m, \mu, p_{max}, \gamma^-, \gamma^+)$
3: *GenerateRemainingJobs*$(\mathcal{T}, n, m, \mu, p_{max})$
4: *SetAdditionalPrecedenceRelations*$(\mathcal{T}, \gamma^-, \gamma^+, \rho)$
5: **return** $\mathcal{T}$

As shown in Algorithm 5, the proposed instance generator first calls procedure *GenerateFirstμJobs* described by Algorithm 6 which i) generates μ jobs with at least one job having processing time p_{max} (lines 1-3), ii) ensures that time windows of all of these jobs overlap (lines 4-6) and iii) ensures that there will be two jobs with γ^- predecessors and γ^+ successors, respectively (see lines 7-8). In the generation of new jobs, the following functions are used to set their deadlines:

$$C_1(S) = \max_{i \in S}(r_i + p_i + q_i)$$

Algorithm 6 *GenerateFirstμJobs*$(\mathcal{T}, m, \mu, p_{max}, \gamma^-, \gamma^+)$

1: Add μ jobs to $\mathcal{T}$ such that p_i, r_i and q_i are random integer numbers in $[1, p_{max}]$ $\forall i \in \mathcal{T}$ and there exits a job $i \in \mathcal{T}$ where $p_i = p_{max}$
2: Set C to an integer number randomly generated in $[C_1(\mathcal{T}), C_2(\mathcal{T})]$
3: Set $d_i = C - q_i$ $\forall i \in \mathcal{T}$
4: **while** maximum number of jobs with overlapping time windows $< \mu$ **do**
5: Choose job i s.t. $r_i = \max_{j \in \mathcal{T}} r_j$ and set $r_i \leftarrow \min_{j \in \mathcal{T}} r_j$
6: **end while**
7: Choose two jobs i and j for which $r_i = \min_{k \in \mathcal{T}} r_k$ and $r_j = \max_{k \in \mathcal{T}} r_k$
8: Set randomly chosen γ^+ (γ^-) jobs as the successors (predecessors) of job i (job j). Adjust (if necessary) time windows of the corresponding jobs accordingly

$$C_2(S) = \max_{i \in S} r_i + \lceil \tfrac{\mu}{m} \rceil \max_{i \in S} p_i + \max_{i \in S} q_i$$

After the generation of first the μ jobs, we have already satisfied the requirements according to the p_{max}, γ^- and γ^+ values. Moreover, we have generated μ jobs for which time windows overlap. In the subsequent, procedure *GenerateRemainingJobs* described in Algorithm 7 generates new jobs as preserving the maximum number of jobs for which the time windows overlap. To this end, it iteratively considers the earliest time t' for which there is a job with deadline t' but there is no job released at time t' and generate new jobs that are released at time t' (lines 3-4).

Algorithm 7 *GenerateRemainingJobs*$(\mathcal{T}, n, m, \mu, p_{max})$

1: $t = 0$
2: **while** $|\mathcal{T}| < n$ **do**
3: $t' = \min_{i \in \mathcal{T}: d_i > t} d_i$ and n' is the number of jobs in $\mathcal{T}$ having deadline t'
4: Add $\min\{n', n - |\mathcal{T}|\}$ new jobs to $\mathcal{T}$ such that $r_i = t'$, p_i and q_i are random integer numbers in $[1, p_{max}]$ for each new job i
5: Set C to an integer number randomly generated in $[C_1(S), C_2(S)]$ where $S = \{i \in \mathcal{T}, d_i > t'\}$
6: Set $d_i = C - q_i$ $\forall i \in S$, $t \leftarrow t'$
7: **end while**

Lastly, the instance generator calls *SetAdditionalPrecedenceRelations* that is shown in Algorithm 8 which generates further precedence constraints (in addition to the ones generated in Algorithm 6). Algorithm 8 requires a given probability ρ which determines the density of the precedence graph. Specifically, as ρ increases, we are expecting denser interval graphs (line 2). We do not set a precedence relationship between two jobs if this would require the adjustments of their time windows (line 1).

Algorithm 8 *SetAdditionalPrecedenceRelations*$(\mathcal{T}, \gamma^-, \gamma^+, \rho)$

1: **for** each pair $(i,j) \in \mathcal{T}^2$ s.t. $|\Gamma^+(i)| < \gamma^+, |\Gamma^-(j)| < \gamma^-, r_i + p_i \leq r_j, d_i \leq d_j - p_j$ and $d_i > r_j$ **do**
2: Set precedence relation $i \rightarrow j$ with probability ρ
3: **end for**

In our experiments, we set both γ^- and γ^+ equal to $\lfloor \frac{\mu}{4} \rfloor$ and consider only a single class of machines. Other parameters are given values as follows: $n \in \{50, 100, 250, 500\}$, $\mu \in \{5, 10, 15, 20, 25\}$, $m \in \{2, 5, 10\}$ and $\rho \in \{0.25, 0.50, 0.75\}$. Considering p_{max} value, we consider two possibilities: i) $p_{max} = \mu$ or ii) $p_{max} = n$. We generate instances for all cross-combinations of the possible parameter values except the cases where $\mu < m$. For each distinct tuple $(n, \mu, m, p_{max}, \rho)$, we generate 5 instances.

Computational experiments are conducted on a workstation with Processor 2x Intel Xeon X5677, 144Go RAM and 3.47 GHz through Visual Studio 2019.

4.2 Computational results

We defined the B&F algorithm with respect to the breadth-first search in Algorithm 2. However, in our computational experiments, we apply the depth-first search in accordance with the objective of finding a feasible solution. When a new node needs to be chosen among multiple candidates to generate, we choose the new node by following the Jackson's rule [15] (node with earliest completion time of a job of C). We use one hour time limit for each instance such that the B&F algorithm is terminated after one hour if the search graph cannot be completely generated yet. We call the instances with 50 and 100 jobs moderate-size as the instances with 250 and 500 jobs are referred to as large-size instances. Similarly, when p_{max} is equal to μ and n, it is referred to as small and high, respectively. In the following, we present computational results for each possible instance tuple (instance size, p_{max} size,m,μ). We present different metrics to analyze the results in terms of the feasible solution existence, complete search graph generation, the number of nodes generated and the solution time. The value of a particular metric for an instance tuple is equal to the mean of that metric's value for the instances in the corresponding instance tuple.

In Table 4, for each instance tuple, we provide the percentage of the instances i) for which a feasible solution is found, ii) which are proven to be infeasible and iii) which are neither proven to be feasible nor infeasible (i.e. the algorithm terminated due to the time limit without finding a feasible solution). When μ is in $\{5, 10\}$, we can solve all instances (an instance is said to be solved if it is proven to be feasible or infeasible). On the other hand, as μ increases, the number of solvable instances gradually decreases. All of the unsolved instances with $\mu = 15$ have 5 machines. This signifies that the problem may get more difficult when the number of machines is neither too small nor too big. When μ becomes larger than 15, there exists at least one instance that cannot be solved

Table 4: Percentages (in %) of instances with respect to feasibility status: i) Feasible (F), ii) Infeasible (IF) and iii) Unknown (UNK)

size	p_{max}	m	μ = 5			10			15			20			25		
			F	IF	UNK	F	IF	UNK	F	IF	UNK	F	IF	UNK	F	IF	UNK
moderate	small	2	53.3	46.7	0.0	60.0	40.0	0.0	90.0	10.0	0.0	93.3	6.7	0.0	90.0	0.0	10.0
		5	-	-	-	93.3	6.7	0.0	96.7	3.3	0.0	100.0	0.0	0.0	100.0	0.0	0.0
		10	-	-	-	-	-	-	96.7	3.3	0.0	100.0	0.0	0.0	100.0	0.0	0.0
	high	2	40.0	60.0	0.0	80.0	20.0	0.0	80.0	20.0	0.0	90.0	6.7	3.3	86.7	0.0	13.3
		5	-	-	-	66.7	33.3	0.0	96.7	3.3	0.0	96.7	0.0	3.3	90.0	0.0	10.0
		10	-	-	-	-	-	-	86.7	13.3	0.0	96.7	3.3	0.0	100.0	0.0	0.0
large	small	2	6.7	93.3	0.0	23.3	76.7	0.0	60.0	40.0	0.0	66.7	20.0	13.3	80.0	0.0	20.0
		5	-	-	-	80.0	20.0	0.0	90.0	6.7	3.3	90.0	3.3	6.7	93.3	0.0	6.7
		10	-	-	-	-	-	-	63.3	36.7	0.0	90.0	6.7	3.3	86.7	0.0	13.3
	high	2	0.0	100.0	0.0	13.3	86.7	0.0	73.3	26.7	0.0	66.7	6.7	26.7	90.0	0.0	10.0
		5	-	-	-	26.7	73.3	0.0	73.3	0.0	26.7	80.0	0.0	20.0	86.7	0.0	13.3
		10	-	-	-	-	-	-	26.7	73.3	0.0	53.3	40.0	6.7	73.3	6.7	20.0

in the instance tuples with large instance size. On the other hand, all moderate instances can be solved when the number of machines is high (i.e. $m = 10$).

The complexity of the Branch&Find algorithm is a function of n, μ and p_{max} (see Theorem 1). In accordance with its complexity, the number of solutions of which statuses are unknown is increasing in the corresponding parameters. Specifically, when instance size is large, $\mu \in \{20, 25\}$ and p_{max} is high, the number of solutions with unknown feasibility status increases and ranges in [10%, 27%] except the instance tuple (large, high, 10, 20). In this exception case, many of the instances are infeasible. Being able to fathom nodes in the early iterations due to the infeasibilities enables to solve the corresponding infeasible instances.

In Table 5, we show for what percent of the instances in each instance tuple, the search graph can be generated completely. Table 5 shows that when μ is small, we not only solve all the instances as shown in Table 4, we can also generate their search graph completely. When μ is in $\{15, 20\}$, we can still generate the complete search graphs for most of the instances except the ones with $m = 5$. Our results indicate again that the problem gets more difficult for medium number of machines. On the other hand, when μ becomes 25, we cannot generate the complete search graph for most of the instances. All of the instances having $\mu = 25$ for which complete graph is generated has the highest number of machines (i.e, $m = 10$). In our results, the impact of the instance size on the complete search graph generation percentages is less significant in relative to the impacts of other parameters. This is in line with the complexity of the proposed FPT so that its complexity is polynomial in the number of jobs, i.e. n.

In Table 6, for each instance tuple, we present two values: i) the average ratio of the actual number of nodes generated over the theoretical maximum number of nodes to be generated and ii) average solution (running) times, by considering

Table 5: Complete search graph generation percentages (in %)

size	p_{max}	m	μ 5	10	15	20	25
moderate	small	2	-	100.0	100.0	66.7	0.0
		5	-	100.0	80.0	0	0.0
		10	-	-	100.0	86.7	10.0
	high	2	100.0	100.0	100.0	66.7	0.0
		5	-	100.0	66.7	0.0	0.0
		10	-	-	100.0	73.3	13.3
large	small	2	100.0	100.0	100.0	50.0	0.0
		5	-	100.0	33.3	3.3	0.0
		10	-	-	100.0	70.0	16.7
	high	2	100.0	100.0	83.3	10.0	0.0
		5	-	100.0	13.3	0.0	0.0
		10	-	-	100.0	70.0	13.3

only the instances of which search graphs are generated completely[4]. The number of nodes generated in our experiments is strictly smaller than the number of nodes that can be generated in the worst case (see Lemma 1). The highest ratio of the actual number of nodes over its theoretical maximum is less than 10^{-5}. This shows that for none of the instances, our empirical results approach the worst-case complexity. As μ increases, the number of nodes generated increases yet not in the order of μ unlike the theoretical maximum number of nodes for most of the instance tuples. Therefore, the ratio of the actual number of nodes and the theoretical maximum number of nodes gets significantly smaller as μ increases. In terms of solution times required to generate the complete search graphs, instances requires less than 1812.7s to be solved on average even if we use one hour time limit. Most of the instances are solved in much less time. This suggests that we can solve instances before their search graph reaches a certain boundary yet they cannot be solved beyond this boundary.

We used the generation of the complete search graph and the time limit as the termination condition of our algorithm for the purpose of more detailed analysis. On the other hand, in accordance with the objective of the proposed algorithm, it can be stopped when a feasible solution is found. To this end, we also analyzed when the first feasible solutions are found in the feasible instances. Since we can find a feasible solution in less than 0.01s for most of the instances, we do not share the details of the first feasible solution finding times and just note the following. For only 18 of all feasible instances, the first feasible solution finding time is greater then 0.10s and only for 5 of them, it is greater than 5s. The

[4] i). Average number of nodes x for a tuple is rounded up to $\lceil x \rceil$ if $x < 100000$. Otherwise, it is rounded up to $1000 * y$ where $y = \lceil x/1000 \rceil$ and presented as yK nodes. ii) If a ratio is presented as "$< 10^{-x}$", it means the corresponding ratio is in $[10^{-x-5}, 10^{-x})$. iii) If an average solution time is less than 0.01, it is presented as "< 0.01".

Table 6: Comparison of the actual and theoretical maximum number of nodes (Nb and Nb*) and solution times for instances of which the search graph is completely generated

size	p_{max}	m	μ = 5			10			15			20			25		
			Nb	Nb/Nb*	Time(s)	Nb	Nb/Nb*	Time(s)	Nb	Nb/Nb*	Time(s)	Nb	Nb/Nb*	Time(s)	Nb	Nb/Nb*	Time(s)
moderate	small	2	322	$< 10^{-5}$	<0.01	25278	$< 10^{-15}$	0.19	1303K	$< 10^{-25}$	21.27	20745K	$< 10^{-35}$	926.20	-	-	-
		5	-	-	-	3359	$< 10^{-15}$	0.02	2865K	$< 10^{-25}$	508.51	-	-	-	-	-	-
		10	-	-	-	-	-	-	172	$< 10^{-30}$	<0.01	172K	$< 10^{-40}$	80.59	1492K	$< 10^{-50}$	573.63
	high	2	285	$< 10^{-10}$	<0.01	99037	$< 10^{-25}$	0.83	2529K	$< 10^{-35}$	33.52	51270K	$< 10^{-50}$	1812.70	-	-	-
		5	-	-	-	3069	$< 10^{-25}$	0.02	1691K	$< 10^{-35}$	271.85	-	-	-	-	-	-
		10	-	-	-	-	-	-	2216	$< 10^{-40}$	0.01	383K	$< 10^{-50}$	285.18	1154K	$< 10^{-65}$	223.60
large	small	2	861	$< 10^{-5}$	0.02	72825	$< 10^{-15}$	1.03	2130K	$< 10^{-25}$	42.48	27238K	$< 10^{-35}$	1071.64	-	-	-
		5	-	-	-	25284	$< 10^{-15}$	0.27	3386K	$< 10^{-25}$	366.91	3387K	$< 10^{-40}$	177.15	-	-	-
		10	-	-	-	-	-	-	836	$< 10^{-30}$	0.01	584K	$< 10^{-40}$	105.00	1427K	$< 10^{-50}$	773.65
	high	2	353	$< 10^{-15}$	0.01	733K	$< 10^{-30}$	17.11	17619K	$< 10^{-45}$	573.52	41820K	$< 10^{-65}$	1696.00	-	-	-
		5	-	-	-	21195	$< 10^{-30}$	0.33	4439K	$< 10^{-45}$	1118.05	-	-	-	-	-	-
		10	-	-	-	-	-	-	1988	$< 10^{-50}$	0.03	338K	$< 10^{-65}$	290.48	1499K	$< 10^{-85}$	461.62

highest first solution finding time is 59.15 which is, surprisingly, for an instance with 50 jobs, 2 machines, small p_{max}, $\mu = 20$ and $\rho = 0.75$.

5 Conclusion

In this paper we developed a new dynamic programming approach to solve the decision problem $P|pre, r_i, d_i|\star$ starting from the Demeulemeester and Herroelen Branch-and-Bound algorithm [7]. New dominance rules were provided, and an efficient way to manage the set of undominated nodes lead to prove that our algorithm is FPT with respect to parameters (μ, p_{max}). Experiments show that the practical efficiency of our algorithm depends on the parameters but that the theoretical complexity is overestimated. Our study could be extended to measure the impact time windows adjustment, and bounds. Generalization of the resource constraints and introduction of optimization criteria is also as secondary goal.

Branch-and-bound methods are widely used and often efficient to solve scheduling problems; however, it is rare that a theoretical fine analysis of their efficiency is performed. From this point of view, parameterized complexity offers a new angle of approach to measure the parameters that explain an efficiency or inefficiency for some instances. In "The Middle Class Gentleman", Molière's character Mr. Jourdain says: "By my faith! For more than forty years I have been speaking prose without knowing anything about it..." Probably several researchers designed FPT algorithms without knowing anything about them. This could be a partial hidden reason why some branch and bound techniques are very efficient in practice. Thus, as a perspective of this work, the study of other Branch-and-Bound-based methods and their adaptation in FPT, depending on the parameters, seems promising.

Acknowledgements This work was supported by the EASI project funded by Sorbonne Universités

References

1. Bellman, R.: The theory of dynamic programming. Bulletin of the American Mathematical Society **60**, 503–515 (1954)
2. van Bevern, R., Bredereck, R., Bulteau, L., Komusiewicz, C., Talmon, N., Woeginger, G.J.: Precedence-constrained scheduling problems parameterized by partial order width. In: Kochetov, Y., Khachay, M., Beresnev, V., Nurminski, E., Pardalos, P. (eds.) Discrete Optimization and Operations Research. pp. 105–120. Springer International Publishing, Cham (2016)
3. Brucker, P.: Scheduling algorithms (4. ed.). Springer (2004)
4. Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research **112**(1), 3–41 (1999). https://doi.org/https://doi.org/10.1016/S0377-2217(98)00204-5, https://www.sciencedirect.com/science/article/pii/S0377221798002045
5. Chen, B., Potts, C.N., Woeginger, G.J.: A Review of Machine Scheduling: Complexity, Algorithms and Approximability, pp. 1493–1641. Springer US, Boston, MA (1998)
6. Cygan, M., Fomin, F.V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer Publishing Company, Incorporated, 1st edn. (2015)
7. Demeulemeester, E., Herroelen, W.: A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science **38**(12), 1803–1818 (1992)
8. Demeulemeester, E.L., Herroelen, W.S.: New benchmark results for the resource-constrained project scheduling problem. Management Science **43**(11), 1485–1492 (1997), http://www.jstor.org/stable/2634582
9. Dolev, D., Warmuth, M.K.: Scheduling precedence graphs of bounded height. J. Algorithms **5**(1), 48–59 (1984)
10. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Springer-Verlag London, 1st edn. (2013)
11. Garey, M., Johnson, D.: Strong NP-completeness results: motivation, examples, and implications. J. Assoc. Comput. Mach. **25**(3), 499–508 (1978)
12. Graham, R., Lawler, E., Lenstra, J., Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer, P., Johnson, E., Korte, B. (eds.) Discrete Optimization II, Annals of Discrete Mathematics, vol. 5, pp. 287–326. Elsevier (1979). https://doi.org/https://doi.org/10.1016/S0167-5060(08)70356-X, https://www.sciencedirect.com/science/article/pii/S016750600870356X
13. Hanen, C., Munier, A.K.: Fixed-parameter tractability of scheduling dependent typed tasks subject to release times and deadlines. Accepted to Journal of Scheduling (2022)
14. Held, M., Karp, R.: A dynamic programming approach to sequencing problems. SIAM Journal on Applied Mathematics **10**(1), 196–210 (1962)
15. Jackson, J.R.: Scheduling a production line to minimize maximum tardiness. Tech. rep., University of California (1955)
16. Kordon, A.M.: A fixed-parameter algorithm for scheduling unit dependent tasks on parallel machines with time windows. Discret. Appl. Math. **290**, 1–6 (2021). https://doi.org/10.1016/j.dam.2020.11.024, https://doi.org/10.1016/j.dam.2020.11.024

17. Leung, J.Y.T.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapman & Hall/CRC, 1st edn. (2004)
18. Mnich, M., van Bevern, R.: Parameterized complexity of machine scheduling: 15 open problems. Computers and Operations Research **100**, 254 – 261 (2018)
19. Möhring, R.H.: Computationally Tractable Classes of Ordered Sets, pp. 105–193. Springer Netherlands, Dordrecht (1989)
20. Ullman, J.D.: Np-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (jun 1975). https://doi.org/10.1016/S0022-0000(75)80008-0, https://doi.org/10.1016/S0022-0000(75)80008-0
21. Walker, R.: An enumerative technique for a class of combinatorial problems. In: American Mathematical Society Symposia in Applied Mathematics. pp. 91–94. No. 10 (1960)