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. Starting from an extension of the Branch-and-Bound algorithm of Demeulemeester and Herroelen to take into account release times and deadlines, we build a state graph of which longest paths represent all active schedule. New dominance rules are also proposed.
We establish that our state graph construction 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. These experiments show that the pathwidth is also a key factor of the practical complexity of the algorithm.

Keywords: Scheduling · Parallel machines · Release times and deadlines · Branch-and-Bound · Fixed-parameter tractable

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 [3,5,16].

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

* Supported by EASI project, Sorbonne universités

positive integer processing time p_i, release time r_i and deadline d_i. 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-hard by Ullman [21]. 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 [17] 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 [22] 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 [18] 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 a FPT algorithm for the resource constrained scheduling problem (RCPSP) parameterized by the pair $(w(G), \lambda)$, where λ is the maximum allowed difference between the earliest starting time and factual starting time of a job. More recently, Munier [19] developed a FPT algorithm for the decision problem $P|prec, r_j, d_j, p_j = 1|\star$. Its parameter μ, called the path-width, 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 the couple 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 methods to solve 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. We discovered that it can be transformed to generation of a state graph, instead of a search tree, linking this algorithm with a dynamic programming approach. We also established several new dominance rules, and modified the generation of successors of a state of the state graph to handle release times and deadlines. However, to simplify the complexity study, we did not consider bounding techniques to prune states.

This leads us to define a new dynamic programming algorithm (DP in short) for our decision problem. We analyse its complexity and show that it is FPT for parameters (μ, p_{max}). This algorithm is significantly different from that developed by Hanen and Munier [13] and has a better time complexity. We also ran some experiments on random instances with controlled parameters that confirms their influence on the practical tractability of the problem. Our experiments show that the practical time complexity of the state graph generation also strongly depends on the pathwidth μ. We also observed that the state graph can be completely generated for small values of the parameters even without bounding techniques.

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 states generated at each step of our algorithm. In Section 3, we present the DP algorithm, the characteristics of the state graph it generates and its complexity analysis. In Section 4, computational experiments for the DP 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 DP 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

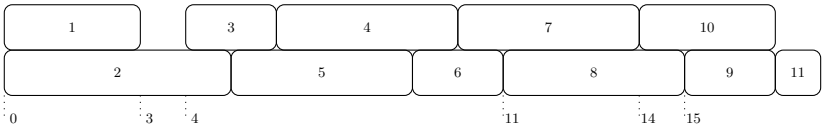

Fig. 1: A feasible schedule on $m = 2$ processors.

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 $(1, 2, 3, 5, 4, 6, 7, 8, 10, 9, 11)$. 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 The DP 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 DP algorithm differentiates from the branch-and-bound algorithm in several ways: a state graph is generated instead of a search tree, release times and deadlines are considered while generating successors of a state, 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 state 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 state graph denoted by $\mathcal{G}(\mathcal{V}, \mathcal{A})$. In Section 3.1, states of the state 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 state. In Section 3.3, the construction of the state graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$ is explained. In Section 3.4, the schedules represented by the paths of the state graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$ are analyzed. Finally, in Section 3.5, the overall complexity of the DP 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.

3.1 State definitions and dominance properties

Definition 1 (Active and Semi-active schedule [20]). *A feasible schedule is called* **semi-active** *(resp.* **active***) if no job can be left shifted by one time unit (resp. scheduled earlier) without changing the starting time of another job.*

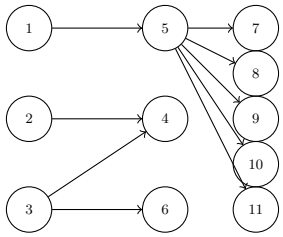

Fig. 2: A precedence graph for the set of jobs of Table 1

Definition 2 (Partial feasible schedule). *Let $V \subseteq \mathcal{T}$ such that no arc (i, j) of G satisfies $i \notin V, j \in V$ and $G_V = (V, E)$ be the precedence sub-graph of G restrained to the set of jobs V. A* **partial feasible schedule** *is a feasible schedule of a subset of jobs $V \subseteq \mathcal{T}$ following the precedence graph $G_V = (V, E)$ and all the constraints on jobs following the initial problem (release times, deadlines and machine limitations).*

Definition 3 (States [7]). *A* **state** *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 vector indexed following P. Moreover, there exists a partial feasible schedule s of jobs of V such that:*

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

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 of state v. The **level** *of a state v is the number of jobs in $V(v)$.*

Note that σ is a partial schedule of $\mathcal{T}$ since it concerns only jobs from $V(v)$. Moreover, the state 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-active schedules for subsets of jobs $V \subseteq \mathcal{T}$ until reaching a complete schedule of S. The next definition introduces the notion of schedule associated with a state in coherence with the Demeulemeester and Herroelen algorithm [7] where some jobs in P can be delayed.

Definition 4 (Schedule associated with a state and perfect state). *If v is a state, a schedule s is said to be associated with v, if it satisfies the following properties :*

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

A state v is a **perfect state** *is the set of feasible schedules associated with v is nonempty.*

Let us consider a feasible schedule S and a state $v = (S_\alpha, t(S_\alpha), P(S_\alpha), M)$. The state 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 states that are considered in our algorithm:

Definition 5. *A partially feasible state 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 states considered in an enumerative algorithm to be reduced.

Proposition 4. *[7] Consider two partially feasible states 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 $\forall i \in P(v) \cap P(v'), M_i(v) \leq M_i(v')$. Then, if v' is perfect, v is perfect either, so v' can be discarded.*

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

Definition 6. *Let v and v' be two states. The state 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 states $\mathcal{N}$ is said undominated if there is no couple of states $(v, v') \in \mathcal{N}^2$ such that v dominates v'.*

The next lemma bounds the number of admissible undominated states:

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 state (i.e. such that there exists an undominated admissible state v with $V = V(v)$). Moreover, for a given job set V, the number of undominated admissible states v such that $V(v) = V$ is bounded by $(2 \times p_{max})^\mu$. The number of admissible undominated states 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 state v is admissible of level α, i.e. $|V(v)| = \alpha$. The sets Z_α and Z'_α are fixed.

1. By Definition 5, $Z_\alpha \subseteq V(v)\backslash P(v) \subseteq V(v)$, 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 5, $P(v) \subseteq V(v)\backslash Z_\alpha$; 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 6, 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 states is bounded by $(2 \times p_{max})^{\mu}$

Thus, the total number of undominated admissible states 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 states

For any $\alpha \in \{0, \ldots, n\}$, we define $\mathcal{V}_\alpha$ as the set of undominated admissible states 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 state v to set $\mathcal{V}_\alpha$ while preserving the undominance property of set $\mathcal{V}_\alpha$. It considers whether state v is dominated by another state $u \in \mathcal{V}_\alpha$ or not. If so, $AddDiscardOrReplace(v, \mathcal{V}_\alpha)$ returns false and the state v is not added to $\mathcal{V}_\alpha$ (lines 4-5). Otherwise, v will be added to $\mathcal{V}_\alpha$ and states that are dominated by v are removed from $\mathcal{V}_\alpha$ (lines 6-10).

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

Require: v is a state of level α; $\mathcal{V}_\alpha$ is a set of undominated admissible states of level α.
Ensure: Add v to $\mathcal{V}_\alpha$ if possible and maintain that $\mathcal{V}_\alpha$ is an undominated set of states; Returns false if v is not added to $\mathcal{V}_\alpha$, v otherwise.

```
1: Set D = ∅
2: Find Q = {u ∈ V_α, V(u) = V(v)}
3: for each state u ∈ Q do
4:     if u dominates v then
5:         return false
6:     else if v dominates u then
7:         D = D ∪ {u}
8:     end if
9: end for
10: V_α = (V_α − D) ∪ {v}
11: return v
```

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 states sharing the same set V are stored in a separate container in $\mathcal{V}_\alpha$. Let $V(C)$ denote the set V of the states 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 states 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 5. So, we only consider jobs in Z'_α to differentiate the containers. Moreover, $|Z'_\alpha| \leq 2\mu$, thus 2μ bits $b_1 \dots b_{2\mu}$ 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 $V(Q)$.

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 state 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 states 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 state graph

The DP algorithm computes the state 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 DP algorithm starts with the root state $v_0 = (\emptyset, 0, \emptyset, \emptyset)$ and gradually builds the state graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$.

Algorithm 2 The DP algorithm

Require: An instance $\mathcal{I}$ of n jobs of $P|prec, r_j, d_j|\star$
Ensure: The associated state 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** $\alpha \in \{0, \dots, n-1\}$ **do**
3: **for** each state $v \in \mathcal{V}_\alpha$ **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: $v' \leftarrow NewState(v, t_{min}, C)$
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 $\mathcal{V} = \bigcup_{\alpha=0}^{n} \mathcal{V}_\alpha$

At line 1 of Algorithm 2, the subsets of admissible undominated states $\mathcal{V}_0, \ldots, \mathcal{V}_n$ 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 states 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 states 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 $NewState$ at line 6 returns a new admissible state built from C and t_{min} if it is possible, or false otherwise. If a new state v' from state 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 there exists another state in $\mathcal{V}_{\alpha'}$ that is dominating state v. In this case, we do not consider the new state v'. Otherwise, $AddDiscardOrReplace(v', \mathcal{V}_{\alpha'})$ deletes the states in $\mathcal{V}_{\alpha'}$ that are dominated by v' and returns v'. Note that we consider that as soon as a state 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 state 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 state v in the state graph and their time $t(u) = t_{min}$.

Algorithm 3 $SetCandidateNewJobs(v)$

Require: v is an admissible state
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** $(R(v), t_{min})$

The DP algorithm ensures that when a new state u is generated from a state v, the cardinality of $V(u)$ is greater than the cardinality of $V(v)$.

By definition of state v, jobs that are not in $V(v)$ can be started at time $t(v)$ at the earliest in the schedules represented by state 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 in a feasible schedule represented by the admissible state 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 states from state v.

Let us consider, for the example of Table 1 and Figure 2, the state $v = (\{1,2,3,4,5,6\}, 10, \{4,6\}, M)$ 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 state $u = (\{1,2\}, 3, \{1,2\}, M)$ 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 state: Algorithm 4 presents the function $NewState$ that returns, if it possible, a new admissible state u built by considering all the jobs from C. If the two conditions on line 1 hold for subset C, Algorithm 4 returns false since there will be at least one job that can be left-shifted in the schedules

associated with the new state. The first condition on line 1 means that at least one job in set $P(v)$ is not included in subset C (i.e. a delayed job). This will incur an idle time in $[t_{min}-1, t_{min}]$ since the corresponding delayed job is in-progress at time t_{min}. On the other hand, the second condition on line 1 means that at least one job in subset C is eligible to be scheduled at time $t_{min}-1$. Consequently, if the relevant two conditions hold, it means at least one job in subset C will be started at time t_{min} at the new state although it could be started before t_{min}. Hence, in this case, no new state will be generated by using subset C.

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

Require: v is an admissible state, t the time instant of the new state, $C \subset R(v)$ the set of new jobs added to $V(v)$ for the new state.
Ensure: an admissible new state u if it is possible, false otherwise.
1: **if** $P(v)\backslash C \neq \emptyset$ and $(C\backslash P(v)) \cap D(v, t_{min}-1) \neq \emptyset$ **then return** false
2: **end if**
3: Set $V(u) = (V(v)\backslash IP(v, t_{min})) \cup C$ and $\beta = |V(u)|$
4: **if** $Z_\beta \not\subseteq V(u)$ or $V(u) \not\subseteq Z_\beta \cup Z'_\beta$ **then return** false
5: **end if**
6: $P(u) = C$; $\forall j \in C$, $M_j(u) = ect(v, j, t_{min})$; $t(u) = \min_{j \in C} M_j(u)$
7: **return** state u

The set of states $V(u)$ and its cardinality β are defined at line 3. Lines 4-5 discard u if it is not admissible (see Definition 5). Line 6 defines $P(u)$, $M(u)$ and $t(u)$ as well.

In our example, starting from state $u = (\{1,2\}, 3, \{1,2\}, M)$ 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 Paths of the state graph

In this section, we define the relationships between the schedules and the paths of the state graph $\mathcal{G}$. We first show that any path from the root state v_0 to a state $v \in \mathcal{V}$ represents at least one schedule.

Lemma 3. *If u is a successor of a feasible state v in the state graph as defined in the DP algorithm, and if s is a partial schedule associated with v, then s can be extended to s' (a feasible schedule of jobs of $v(u)$) such that jobs of $P(u)$ complete at time $M(u)$. We deduce that any path of the graph from $\mathcal{V}_0$ to a state of $\mathcal{V}_n$ is associated with at least one schedule.*

Proof. Let s be a partial schedule associated with state v satisfying definition 3. Let C the subset of jobs used to generate u from v. Recall that $V(u) = (V(v)\backslash IP(v, t_{min})) \cup C$. We define the partial schedule s' as follows:

- If $i \in C\backslash IP(v, t_{min})$ then $s'(i) = M_i(u) - p_i = \max(t_{min}, r_i)$.
- otherwise $s'(i) = s(i)$

We claim that s' is a schedule, in which jobs of $P(u)$ end at times described in $M(u)$.

We now consider the following property:

Theorem 1. *Let $\mathcal{G}$ be the state graph built without applying the dominance criteria in Proposition 4. If s is a feasible active schedule, then there is a path in the state graph $\mathcal{G}$ from the initial state to a state of $\mathcal{V}_n$, so that s is associated with each state of the path.*

Proof. Let s be a feasible active schedule. We define for each value the partial schedules s_α, $\alpha \in \{0, \ldots, n\}$. s_α is a partial schedule of s including only its first α jobs $J_1, J_2, ..., J_\alpha$ sorted by increasing starting times (breaking ties with job index). 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 $s(J_\alpha)$: $t(s_\alpha) = \min_{\{j \in s_\alpha, s(j)+p_j > s(J_\alpha)\}} s(j) + p_j$.

For each $\alpha \in \{0, \ldots, n\}$, we define $P(s_\alpha)$ to be the set of jobs of s_α that complete not before $t(s_\alpha)$. Moreover, $M(s_\alpha)$ is the completion time in s_α of jobs of $P(s_\alpha)$.

Now, we consider the set of indexes $\mathcal{A} = \{\alpha \in \{0, \ldots, n\}, t(s_{\alpha+1}) > t(s_\alpha)\}$

Let us show that if $\alpha \in \mathcal{A}$ then

$$s(J_{\alpha+1}) \geq t(s_\alpha) \tag{1}$$

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 not later than 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 $t(s_{\alpha+1}) \leq t(s_\alpha)$ is a contradiction. This implies that for any job $j \notin s_\alpha$, $s(j) \geq s(J_{\alpha+1}) \geq t(s_\alpha)$.

Let us denote by $\{\alpha_0 = 0, \alpha_1, \ldots, \alpha_\kappa = n\}$ the increasing sequence of indexes of $\mathcal{A}$.

For $k \in \{0, \ldots, \kappa\}$, we define $v_k = (s_{\alpha_k}, t(s_{\alpha_k}), P(s_{\alpha_k}), M(s_{\alpha_k}))$. We claim that states $v_0, v_1, \ldots, v_\kappa$ are on a path of the state graph $\mathcal{G}$. We first prove that for any k, v_k fulfils Definition 3 of a state.

1. Consider a job $i \in s_{\alpha_k}\backslash P(s_{\alpha_k})$. By definition of s_{α_k} and $P(s_{\alpha_k})$, $s(i) + p_i < t(s_{\alpha_k})$ and the first item of Definition 3 is verified;
2. If now $i \in P(s_{\alpha_k})$, then $s(i) < t(s_{\alpha_k}) \leq s(i) + p_i = M_i$, so that the last item is also verified.

By recurrence on k, assume that $v_0, \ldots, v_k$ is a path of $\mathcal{G}$. We know that $t(s_{\alpha_k+1}) > t(s_{\alpha_k})$ and that $\forall \alpha'$ such that $\alpha_k < \alpha' \leq \alpha_{k+1}$, we have $t(s_{\alpha_{k+1}}) \leq t(s_{\alpha'}) \leq t(s_{\alpha_k+1})$. Moreover by definition we necessarily have $t(s_{\alpha_{k+1}}) > s(J_{\alpha_{k+1}})$ and we know by Equation 1 that $s(J_{\alpha_{k+1}}) \geq t(s_{\alpha_k})$. This implies that all such jobs $J_{\alpha'}$ start not earlier than $t(s_{\alpha_k})$ (by Equation 1) and complete at or after $t(s_{\alpha_{k+1}})$. So, no job is completed in the interval $(t(s_{\alpha_k}), t(s_{\alpha_{k+1}}))$.

Now, when considering state v_k, the computation of t_{min} would give a value $t_{min} \geq t(s_{\alpha_k})$.

Consider the set

$$C = \{J_{\alpha_k+1}, \ldots, J_{\alpha_{k+1}}\} \cup \{j \in P(v_k), s_j + p_j \geq t(s_{\alpha_{k+1}})\}$$

Jobs of C are completed or in-progress at $t(s_{\alpha_{k+1}})$ in schedule s. So C cannot comprise more than m jobs. Thus, two cases may occur:

Case 1: $ect^*(v_k) > s(J_{\alpha_{k+1}})$. In this case, set C is included in $R(v_k)$ so that the state v_{k+1} is a valid successor of state v_k. Observe that $P(v_k, t_{min}) \backslash C = \emptyset$ (no job has been removed from $P(v_k)$ to build $V(v_{k+1})$ so that v_k cannot be pruned using the rule related to semi-active schedules in Algorithm 4).

Case 2: $ect^*(v_k) < s(J_{\alpha_{k+1}})$. In this case, some jobs of C do not belong to $R(v)$.

Case 2.1: $ect^*(v_k)$ is the completion time of a job in $P(v_k)$. Then necessarily $s(J_\beta) < ect^*(v_k)$ for all $J_\beta \in C - P(v_k)$, otherwise we would have $t(s_{\beta-1}) < t(s_\beta)$, a contradiction.

Case 2.2: $ect^*(v_k)$ is the completion time of a job in $C - P(v_k)$. Similarly, if for some $J_\beta \in C$, $s(J_\beta) \geq ect^*(v_k)$ then $t(s_{\beta-1}) < t(s_\beta)$, a contradiction.

Case 2.3: $ect^*(v_k)$ is the completion time of a job not in C. Assume that there exists a first job $J_\beta \in C$ with $s(J_\beta) \geq ect^*(v_k)$. If J_β was eligible before $ect^*(v_k)$ it could be left shifted and s would not be active, since we know that all processors processing jobs in $C - P(v_k)$ are free at time $t(v_k)$ in state v_k (otherwise the machine is processing a job of $P(v_k)$).

So the release time of J_β is at least $s(J_\beta)$. And in schedule s, the processor processing J_β is free between $t(v_k)$ and $s(J_\beta)$, whereas there is a job $j \notin C$ for which $ect(v_k, j) = ect^*(v_k)$. Job j is scheduled later in S, whereas it could be scheduled earlier, without changing the other starting times. So s is not an active schedule.

3.5 Complexity analysis of the DP algorithm

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

Proposition 5. *For any admissible state $v \in \mathcal{V}$, $|R(v)| \leq \mu$.*

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

Proof. The time complexity of the instructions at lines 1, 3 and 4 of Algorithm 4 are $\mathcal{O}(n)$, while it is $\mathcal{O}(m)$ at line 6. Since $m \leq n$, the whole time complexity of this algorithm is $\mathcal{O}(n)$, which proves the lemma. □

Our main theorem follows:

Theorem 2. *The DP algorithm (see Algorithm 2) is a 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 7 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 5, $|R(v)| \leq \mu$. 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 6, 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: $GenerateFirst\mu Jobs(\mathcal{T}, m, \mu, p_{max}, \gamma^-, \gamma^+)$
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:

$$C_1(S) = \max_{i \in S}(r_i + p_i + q_i)$$
$$C_2(S) = \max_{i \in S} r_i + \lceil \tfrac{\mu}{m} \rceil \max_{i \in S} p_i + \max_{i \in S} q_i$$

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

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

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**

deadline t' but there is no job released at time t' and generate new jobs that are released at time t' (lines 3-4).

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 $\lfloor \frac{\mu}{4} \rfloor$. Time windows of a given job and its successors (and its predecessors as well) must overlap, otherwise the precedence relations between them would be redundant. Therefore,

if γ^- and γ^+ get high values, available jobs at a given time in terms of the time windows would consist only a few jobs and their successors and thereby precedence relations would excessively dictate the possible schedules. To avoid such cases, we keep the magnitude of γ^- and γ^+ limited. 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

In our computational experiments, we apply the depth-first search in accordance with the objective of finding a feasible solution. The enumeration of subsets C on line 5 of Algorithm 2 uses sorted earliest starting times of the jobs in $R(v)$ where ties are broken considering the ascending order of deadlines. Thus, the first chosen new state schedules the jobs with the earliest starting times (and earliest deadlines in case of ties) in set $R(v)$ and thereby follows the Jackson's rule [15]. We use one hour time limit for each instance such that the DP algorithm is terminated after one hour if the state graph cannot be completely generated yet.

In Table 4, we provide the percentage of the instances for which the state graph can be generated completely. In our results, the impact of the instance size on the complete state graph generation percentages is less significant according to the impacts of other parameters, especially μ. This is consistent with the complexity of the proposed FPT so that its complexity is polynomial in the number of jobs n.

- We first note that μ seems to be a key parameter in practice, since the percentages are clearly decreasing with μ in every cases. For $\mu = 10$ the whole state graph can be generated, for $\mu = 25$ it is hopeless;
- The value of m has an impact. We can observe that for $m = 2$ or $m = 10$ the percentages are often similar, whereas when $m = 5$ the percentage dramatically decreases. This could be partially explained since the number of enumerated sets C is bounded by $\binom{\mu}{m}$ which is lower for low or high values of m;
- The impact of n is quite limited with respect to μ when m is either 2 or 10, even for $p_{\max} = n$.

Besides, we observed that higher number of states can be pruned by the dominance criterion as the number of machines gets smaller. Specifically, the overall percentages of the dominated states over the total number of states generated are 63.5%, 45.5% and 28.7% when the number of machines is 2, 5 and 10, respectively.

Table 4: Complete state graph generation percentages for different tuples $(n, p_{\max}, m, \mu)$.

			μ				
n	$p_{\max}$	m	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

In all our experiments, when the state graph was completely generated, instance required less than 1812.7s on average. Most of the instances requires much less time.

We also analyzed when the first feasible solutions are found in the feasible instances. For most of them, we can find a feasible solution in less than 0.01s. 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.

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 states 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)
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)
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)
13. Hanen, C., Munier Kordon, A.: 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. Leung, J.Y.T.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapman & Hall/CRC, 1st edn. (2004)
17. Mnich, M., van Bevern, R.: Parameterized complexity of machine scheduling: 15 open problems. Computers and Operations Research **100**, 254 – 261 (2018)
18. Möhring, R.H.: Computationally Tractable Classes of Ordered Sets, pp. 105–193. Springer Netherlands, Dordrecht (1989)
19. Munier Kordon, A.: A fixed-parameter algorithm for scheduling unit dependent tasks on parallel machines with time windows. Discret. Appl. Math. **290**, 1–6 (2021)
20. Sprecher, A., Kolisch, R., Drexl, A.: Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. European Journal of Operational Research **80**(1), 94–102 (1995)
21. Ullman, J.D.: Np-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (jun 1975)
22. 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)