Two Deadline Reduction Algorithms for Scheduling Dependent Tasks on Parallel Processors (extended version)

Claire C. Hanen, Alix Munier Kordon, Theo Pedersen

Two Deadline Reduction Algorithms for Scheduling Dependent Tasks on Parallel Processors

Claire Hanen[1,2 [0000−0003−2482−5042]], Alix Munier Kordon[1 [0000−0002−2170−6366]], and Theo Pedersen[1 [0000−0002−4567−1823]]

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

Abstract. This paper proposes two deadline adjustment techniques for scheduling non preemptive tasks subject to precedence relations, release dates and deadlines on a limited number of processors. This decision problem is denoted by $P|prec, r_i, d_i|\star$ in standard notations. The first technique is an extension of the Garey and Johnson algorithm that integrates precedence relations in energetic reasoning. The second one is an extension of the Leung, Palem and Pnueli algorithm that builds iteratively relaxed preemptive schedules to adjust deadlines.
The implementation of the two classes of algorithms is discussed and compared on randomly generated instances. We show that the adjustments obtained are slightly different but equivalent using several metrics. However, the time performance of the extended Leung, Palem and Pnueli algorithm is much better than that of the extended Garey and Johnson ones.

Keywords: Scheduling Problem · Precedence constraints · Energetic reasoning · Preemptive relaxation.

1 Introduction

This paper addresses the decision scheduling problem described in standard notations introduced in [15] as $P|prec, r_i, d_i|\star$. A set of tasks $\mathcal{T}$ and a precedence graph $\mathcal{G}$ are given. Each task $i \in \mathcal{T}$ has a deadline d_i, a release date r_i and a duration p_i. Tasks are performed on m identical processors. We address the existence of a feasible schedule. Notice that the problem is NP-hard in the strong sense, even in the special cases where no precedence constraints exists and one machine is considered $1|r_i, d_i|\star$ [12] or with unit execution times of tasks and common deadline $P|prec, r_i, d_i = D, p_i = 1|\star$ [29].

However, defining efficient polynomial algorithms providing necessary existence conditions is a challenging question since they might be used to improve the efficiency of constraint programming or branch and bound algorithms for the related optimization problems. Indeed, such necessary existence conditions combined to a binary search can provide a lower bound of the makespan ($C_{\max}$) or

the maximum lateness ($L_{\max}$). Exact algorithms make use of these bounds [24], which appear particularly interesting when the branching scheme is based on splitting or reducing the tasks' intervals [2,5].

Such necessary conditions have been investigated by many authors since the early eighties, thoroughly improving the efficiency of exact algorithms. Several of them use interval adjustment techniques (ie. reducing deadlines and increasing release times) by relaxing precedence constraints. The special case of unit execution times has been also investigated with adjustment techniques considering both precedence and resource constraints.

Interval adjustment techniques based on energetic reasoning, ie. on the measure of the mandatory workload of time intervals, have been the subject of much attention when no precedence constraint is considered. These techniques developed for the problem $P|r_i, d_i|\star$ were extended to handle the cumulative scheduling problem (CuSP). In this case, each task i requires c_i resources for its execution, and the total number of resources is bounded. Baptiste et al. [1] and Derrien and Petit [9] have developed low time complexity algorithms by reducing the number of considered intervals. Ouellet and Quimper [27] and Carlier et al. [6] have improved the data structures used in the algorithms. Tesch in [28] analyzed the time needed to reach a fixed point for the technique called energetic edge finding.

Most recent studies taking into account both precedence and resource constraints are devoted to the resource constrained scheduling problem (RCPSP) and extend earlier work on 1-machine and job-shop scheduling problems. According to Laborie and Nuijten [22], energetic constraints are either propagated on precedence relations, or precedence constraints are considered independently from the job's release times and deadlines in "energy precedence constraints". In order to compute a lower bound on the makespan, Haouari et al. proposed to integrate RCPSP resources and precedence constraints through linear programming via relevant relaxations of precedence and valid inequalities, either using energetic [18] or preemptive bounds [19].

In addition, the special case with unit processing times ($p_i = 1$ for each task $i \in \mathcal{T}$) has been investigated by several authors, most of them using reduction of deadline techniques embedding precedence and resource constraints. Garey and Johnson in [12] derived a polynomial algorithm (GJ algorithm in short) based on energetic reasoning that solves the decision problem for $m = 2$ processors. This algorithm was extended by Hanen and Zinder [17] to get an approximation algorithm for the $L_{\max}$ criteria for general parallel machines case when tasks have unit processing times. The Leung, Palem and Pnueli algorithm [23] (abbreviated to LPP algorithm) expresses necessary conditions on deadlines based on the iterative construction of schedules for relaxed sub-problems without precedence constraints. They also proved that this algorithm optimally solves several problems with particular precedence graphs. Hanen and Munier [4] showed that these two algorithms reach the same fixed point deadlines and an experimental study confirmed that the LPP algorithm is faster than the GJ one.

Our contribution in this paper is to extend both the GJ and LPP algorithms to handle tasks with any duration, and to compare the two approaches. Notice that, due to the inherent symmetry of the problem, all the algorithms considered here can be used to modify release dates as well, by simply reversing the orientation of the precedence arcs and swapping release times and deadlines. However, our experiments only considered deadline modifications.

The extension of the GJ algorithm theoretically dominates usual energetic reasoning due to the addition of precedence constraints. Our approach also considers stronger conditions than Laborie [21] who only considers the successors of a task to adjust the deadlines. The extension of LPP algorithm is based on the iterative construction of preemptive schedules. These two approaches were experimentally compared on randomly generated instances: we first proposed several measures of the effective deadline reduction, and the variation of the intrinsic parallelism of the instances. We observed that the reductions of the deadlines are roughly similar for the two extensions, even if the deadlines obtained are not necessarily equal. However, we also observed that according to the theoretical time complexity evaluation, the LPP algorithm has a much lower complexity than GJ.

The paper has seven sections. In Section 2, we present the problem and the main notations. Section 3 is devoted to the extension of the GJ algorithm and the energetic reasoning. Section 4 presents the extension of the LPP algorithm. Section 5 presents our testing procedures, while Section 6 is dedicated to our experiments. Finally, we conclude in Section 7.

2 Notations

An instance $\mathcal{I}$ of our scheduling problem is given by a set of n tasks $\mathcal{T}$, a precedence graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$ and m identical processors. For every task $i \in \mathcal{T}$, we denote by p_i the execution time of i. We suppose that the release time r_i and the deadline d_i of each task i are given, and satisfy

$$r_i + p_i \leq d_i. \tag{1}$$

A feasible schedule assigns a starting time t_i, such that $r_i \leq t_i \leq d_i - p_i$, and a processor among the m available ones, so that two tasks assigned to the same processor do not overlap. We consider the decision problem of the existence of a feasible schedule denoted by $P|prec, r_i, d_i|\star$.

For any pair of tasks $(i, j) \in \mathcal{T}^2$, we note $i \to j$ if there exists a path in $\mathcal{G}$ from i to j. Then, $\Gamma^{+\star}(i)$ (resp. $\Gamma^{-\star}(i)$) is the set of descendants (resp. ancestors) of i, which are tasks j such that $i \to j$ (resp. $j \to i$). For any pair of tasks (i, j) with $i \to j$, we denote by $\ell^{\star}_{ij}$ the maximum value $\sum_{k \in \nu, k \neq j} p_k$ of a path ν of $\mathcal{G}$ from i to j. We assume that these values are pre-processed. This can be done in time complexity $\mathcal{O}(n^3)$ by using the Floyd-Warshall algorithm [8].

We assume that release times and deadlines are consistent with the precedence constraints:

$$\forall (i, j) \in \mathcal{A}, r_i + p_i \leq r_j \text{ and } d_j - p_j \geq d_i. \tag{2}$$

In the sections below, we introduce deadline modification algorithms.

We consider the algorithm PROPAGATE(i, d) that computes a consistent deadline vector assuming that all values of the input deadline vector d, except maybe the modified deadline d_i, are consistent ie. follows conditions (1) and (2). PROPAGATE(i, d) returns false if $d_i - p_i < r_i$, otherwise it adjusts all the ancestors j of i by setting $d_j = \min(d_j, d_i - \ell^\star_{ji} + p_j - p_i)$ and returns true. Notice that then for any ancestor j of i, $r_j + p_j d_j$. The time complexity of PROPAGATE(i, d) is $\mathcal{O}(n)$ provided that the values $\ell^\star_{ij}$ are preprocessed.

3 Extension of the Garey and Johnson Algorithm

In this section we first explain the deadline reduction principle on which the Garey and Johnson algorithm [12] is based. Then we present the extended Garey and Johnson algorithm (eGJ in short) in its weak form, and analyze its time complexity. Finally we present the strong form of eGJ.

3.1 Principles of deadline reductions

The idea of the original Garey and Johnson algorithm [12], which was designed to solve the problem for two processors and tasks with unit processing times, is to reduce the deadline of a job i based on the measure of the number of tasks that must be executed in an interval $[s, t]$ assuming i ends at its deadline. This idea is extended here for tasks with any processing time by considering energetic reasoning [1] on time intervals.

Let i be a task and let us consider two values $s \leq t$ such that i may end between s and t:

$$r_i \leq s \leq d_i \leq t. \tag{3}$$

Figure 1 presents the three subsets of tasks $I(i, s, t)$, $S(i, s, t)$ and $\mathcal{T}(i, s, t)$ that should have a part processed between s and t.

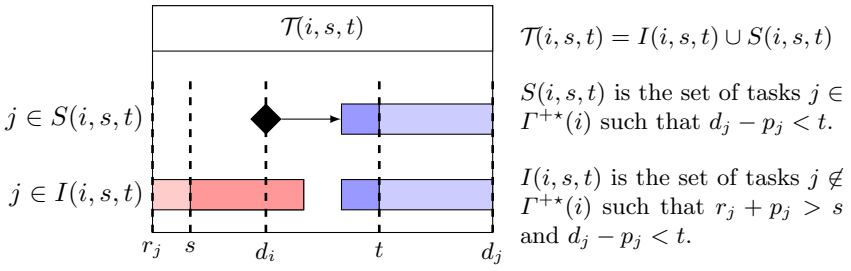


Fig. 1: Sets of tasks $I(i, s, t)$, $S(i, s, t)$ and $\mathcal{T}(i, s, t)$ with some mandatory part in $[s, t]$

Any task $j \in I(i, s, t)$ has no precedence relation with i; we set $w_j(i, s, t)$ as the minimum part of the task j that must be performed between s and t in any

feasible schedule, ie. when j is left shifted and right shifted as illustrated by the blue parts in Figure 1. Clearly, for any task $j \in I(i,s,t)$,

$$w_j(i,s,t) = \min(t-s, p_j, \max(0, r_j + p_j - s), \max(0, t-(d_j - p_j))).$$

Similarly, tasks j from $S(i,s,t)$ are descendants of i with a minimum part $w_j(i,s,t)$ that must be performed between s and t **assuming** i **may end between** s **and** d_i. Notice that the task j starts after s and we just have to consider the contribution of j when it is right shifted.

$$\forall j \in S(i,s,t), \quad w_j(i,s,t) = \min(p_j, \max(0, t-(d_j - p_j))).$$

The total amount of work that is to be performed between s and t considering the minimum contribution of i in the interval $[s,t]$ is then:

$$W(i,s,t) = \max(0, r_i + p_i - s) + \sum_{j \in \mathcal{T}(i,s,t)} w_j(i,s,t).$$

We define the associated slack $\Delta(i,s,t) = W(i,s,t) - m(t-s)$. If $\Delta(i,s,t) > 0$, there is not enough room in the time interval $[s,t]$ to execute the energy $W(i,s,t)$. This situation will fall into one of two cases, as in the following properties:

Property 1. Let us consider a task $i \in \mathcal{T}$ and two values s and t such that $r_i \leq s \leq d_i \leq t$. If $\Delta(i,s,t) > 0$ and $S(i,s,t) = \emptyset$ then no feasible schedule exists.

Proof. If $S(i,s,t) = \emptyset$, then $\mathcal{T}(i,s,t) = I(i,s,t)$; any task $j \in I(i,s,t)$ has no precedence relation with i and thus $w_j(i,s,t)$ part of j must be executed in the time interval $[s,t]$. The total energy $W(i,s,t)$ must then be executed in $[s,t]$ in any feasible schedule. Since $\Delta(i,s,t) > 0$, no feasible schedule exists, the result. □

Property 2. Let us consider a task $i \in \mathcal{T}$ and two values s and t such that $r_i \leq s \leq d_i \leq t$. If $\Delta(i,s,t) > 0$ and $S(i,s,t) \neq \emptyset$, then in any feasible schedule, the completion time C_i of i verifies the inequality

$$C_i \leq t - \left\lceil \frac{W(i,s,t)}{m} \right\rceil. \tag{4}$$

Proof. The part $w_j(i,s,t)$ of any task $j \in S(i,s,t)$ must be executed before time t and after the end of i. Otherwise, if $j \in I(i,s,t)$, $w_j(i,s,t)$ is the part of j that must be executed between s and t. Since $W(i,s,t) > 0$, the only way to execute these tasks is to decrease the completion time C_i of i in order to fit tasks from $\mathcal{T}(i,s,t)$ between C_i and t, thus the completion time C_i of i must verify equation (4). □

3.2 Description of the eGJ algorithm

The eGJ algorithm takes as input an instance $(\mathcal{T}, \mathcal{G}, m, r, d)$ of the problem and outputs either a set of modified deadlines $d^* = (d_i^\star)_{i\in\mathcal{T}}$ that should be fulfilled by any feasible schedule, or indicates that no feasible schedule exists, based on the conditions expressed in Properties 1 and 2.

Triples (i, s, t) are enumerated in a way that will be described below, and at each step if $\Delta(i, s, t) > 0$ then it either results in an infeasibility or defines a modification of the deadline of i based on the inequality (4):

$$d_i \leftarrow t - \left\lceil \frac{W(i, s, t)}{m} \right\rceil$$

Once a modification of d_i occurs, the algorithm propagates the modification to the nodes of $\Gamma^{-\star}(i)$ using PROPAGATE(i, d).

Not all possible triples (i, s, t) following inequality (3) need to be considered; Hanen and Munier [16], inspired by Carlier et al. [7] show that the slack $\Delta(i, s, t)$ has a local maxima only at the dominant triples with the forms (i, r_j, d_k), (i, d_i, d_k), $(i, r_k+d_k-d_j, d_k)$, $(i, d_i, r_k+d_k-d_i)$ with $d_i > r_k$ or $(i, r_j, r_k+d_k-r_j)$ with $r_j > r_k$.

For a current deadline vector d, we denote by $R(d)$ the set of values t such that there exists a dominant triple (i, s, t). If $t \in R(d)$, we denote by $X_t(d)$ the set of tasks i such that there exists a dominant triple (i, s, t). Finally, for $t \in R(d)$, and $i \in X_t(d)$ we denote by $L_{i,t}(d)$ the set of values s, such that (i, s, t) is a dominant triple.

Algorithm 1 enumerates the dominant triples in three nested loops. The outer loop enumerates t in decreasing order by maintaining a sorted list R corresponding to the set $\{\tau \in R(d), \tau < t\}$. The intermediate loop browses elements i of a list X corresponding to the set $X_t(d)$ in decreasing order of deadlines, and the inner loop browses elements s of a list L corresponding to the set $L_{i,t}(d)$ in increasing order. At each iteration, $\Delta(i, s, t)$ is computed and either a contradiction is found, or d_i is updated and propagated. Ordered lists X and R are then updated.

3.3 Complexity analysis of eGJ

The next lemma will be later used to bound the number of iterations of the outer loop.

Lemma 1. *Consider an iteration t of the outer loop for which at the beginning of the iteration at least one task k satisfies $t = d_k$. At the end of this iteration, either infeasibility is detected or there is at least one task removed from X in line 16.*

Proof. Let us suppose by contradiction that at iteration $t = d_i$ for $i \in X$, no infeasibility is detected and no task is removed from X. If d_i is modified with an interval $[s, t]$, then $\Delta(i, s, t) > 0$ and according to the propagation at each step, any task $j \in \Gamma^{+\star}(i)$ satisfies $d_j - p_j \geq d_i = t$, so $S(i, s, t) = \emptyset$. Following Property 1, the algorithm returns infeasibility (the contradiction). □

Algorithm 1 eGJ algorithm

Require: A precedence graph $\mathcal{G}$, release dates vector r, deadline vector d, processing times vector p and m identical processors
Ensure: Modified deadlines vector $d^\star$ or infeasibility
1: $d_{\max} = \max_{i \in \mathcal{T}} d_i$,
2: $R = R(d_{\max})$ in decreasing order, $X = X_{d_{max}}(d)$ in decreasing order
3: **while** $R \neq \emptyset$ and $X \neq \emptyset$ **do**
4: t = first element of R, remove t from R
5: **for all** $i \in X$ **do**
6: $L = L_{i,t}(d)$ in increasing order
7: **repeat**
8: s = first element of L, remove s from L
9: **if** $\Delta(i, s, t) > 0$ **then**
10: Update d_i or return false (infeasibility) per Properties 1, 2
11: $d = \textsc{Propagate}(i, d)$, return false if inconsistency with release times.
12: Update R (sorted list of $\{\tau < t, \tau \in R(d)\}$)
13: **end if**
14: **until** $s \geq d_i$
15: **end for**
16: Remove from X the tasks j for which $d_j = t$
17: **end while**
18: **return** $d^\star = (d_i)_{i \in \mathcal{T}}$

The next lemma is an outcome of Lemma 1 that bounds the number of iterations of the outer loop of Algorithm 1.

Lemma 2. *The successive values of t are strictly decreasing. Moreover, the total number of these successive values belongs to $\mathcal{O}(n^2)$.*

Proof. At the initialization step, all the values of R are different. At line 12, R is updated with a list of values strictly less than t, thus the successive values of t are strictly decreasing. Let us consider now all the possible successive values for t:

- If $t = d_k$, then following Lemma 1, at least one task i is removed from X. Thus, there are at most n iterations in this case.
- Now, if $t = r_k + d_k - d_i$ with $d_i > r_k$, then $d_k > t$ and thus will not be later modified by the algorithm. Moreover, if d_i is decreased to d'_i, $t' = r_k + d_k - d'_i > t$ and thus will not be considered after t. There are then at most n^2 iterations in this case.
- Lastly, if $t = r_k + d_k - r_j$ with $r_j > r_k$, then $d_k > t$ and will also not be decreased further. There are also n^2 iterations in this case.

We deduce that the number of the successive values of t belongs to $\mathcal{O}(n^2)$, which concludes the lemma.

□

Theorem 1. *Algorithm eGJ is in time $\mathcal{O}(n^5 \log(n))$.*

Proof. For a fixed task $i \in X$, the execution time of the inner loop belongs to $\mathcal{O}(n^2 \log n)$. Indeed, there are $\mathcal{O}(n^2)$ values of s that must be sorted (in time $\mathcal{O}(n^2 \log(n))$. The time of the computation of the slack is in $\mathcal{O}(n)$. The modification of d_i and the propagation happen only once per iteration on i; the next value of d_i is less than s and thus the inner loop on s ends. So the time complexity of this modification and propagation is $\mathcal{O}(n)$. Lastly, updating of the sorted list R is $\mathcal{O}(n^2 \log(n))$.

Now, the size of X is bounded by n, while by Lemma 2, the total number of iterations of the outer loop belongs to $\mathcal{O}(n^2)$, thus the theorem is proved.

□

The tightness of this bound is not proved. We will show in Section 6 that the experimental complexity of this algorithm is much smaller.

3.4 Strong form of eGJ

The deadline reduction can be strengthened by considering for any valid triple (i, s, t) a new slack $\overline{\Delta}(i, s, t)$ assuming i is right shifted ie. i ends exactly at its deadline:

$$\overline{\Delta}(i, s, t) = \min(p_i, d_i - s) + \sum_{j \in \mathcal{T}(i,s,t)} w_j(i, s, t) - m(t - s) \tag{5}$$

Now assume that $\Delta(i, s, t) \leq 0$ and $\overline{\Delta}(i, s, t) > 0$. In any feasible schedule the completion time of i satisfies $C_i - s + \sum_{j \in \mathcal{T}(i,s,t)} w_j(i, s, t) \leq m(t - s)$ so that

$$C_i \leq s + \min(p_i, d_i - s) - \overline{\Delta}(i, s, t). \tag{6}$$

The deadline of i can thus be reduced to the right term of equation (6). This condition can be inserted in Algorithm 1. We add to the inner loop (line 13) the following pseudo-code in the case where $\Delta(i, s, t) \leq 0$ and $\overline{\Delta}((i, s, t) > 0$:

Algorithm 2 s-eGJ pseudo code to be inserted in Algorithm 1

```
1: Modify d_i according to (6)
2: d=PROPAGATE(i, d), return false if inconsistency
3: if not resultP then
4:    return false
5: end if
6: update L
```

The arguments stated in Lemma 1 do not apply in this case; instead, all possible values of $0 \leq t \leq d_{\max}$ might be considered in the outer loop without updating R at each modification of deadlines, which would lead to a pseudo-polynomial complexity.

Theorem 2. *The complexity of strong form of eGJ can be pseudo-polynomially bounded by* $\mathcal{O}(d_{\max} \min(d_{max}, n^2).n^3 \log(n))$.

Proof. Unlike the weak form of eGJ, after a strong modification, the inner loop does not end and L must be updated to include new values s' greater than the current s, but still lower than d_i. Updating and sorting L after a modification of a deadline and its propagation can be done in $\mathcal{O}(n^2 \log(n))$ since only the form $r_k + d_k - d_j$, with $t = d_k$ is concerned. In the intermediate loop on i, the number of updates of d_i is bounded by its margin $d_i - r_i$ and by the maximum number $\mathcal{O}(n^2)$ of elements in L. Let us define $\rho = \min(\max_{i \in \mathcal{T}}(d_i - r_i), n^2)$. In the worst case the inner loop has a complexity $\mathcal{O}(\rho n^2 \log(n))$. So, there are $\mathcal{O}(n)$ iterations in the intermediate loop, and finally, if all possible values of t are considered, the outer loop has $\mathcal{O}(d_{\max})$ iterations. □

3.5 eGJ and energetic reasoning

Energetic reasoning, when used to check feasibility in problems without precedence,tests the energy in intervals which are also considered by the eGJ algorithm. So eGJ dominates energetic reasoning, by considering precedence constraints and making iterative adjustments. Adjustments in usual energetic reasoning use the same ideas as in strong eGJ, but again, considering successors and fixed points lead theoretically to lower deadlines. On another hand, Laborie [21] proposes an adjustment of deadlines method that measures the energy used by the successors of each task i to define a bound on its completion time. The eGJ algorithm uses stronger conditions by also considering tasks independent from i in the energy measure. Moroeover, adjustments are iterated.

4 Extension of the Leung Palem and Pnueli Algorithm

This section is devoted to the description of two extended forms of the Leung, Palem and Pnueli algorithm [23] for tasks with different execution times. Subsection 4.1 presents a general possible extension of this algorithm (eLPP in short), based on an optimization scheduling problem BackwardSchedule. Two implementations are then discussed. In Subsection 4.2, this problem is relaxed to obtain a polynomial time algorithm while an exact pseudo-polynomial time algorithm is presented is Subsection 4.3.

4.1 Description of the eLPP algorithm

For any task $i \in \mathcal{T}$, we note $Indep(i)$ the set of tasks $j \in \mathcal{T}$ such that $i \not\to j$ and $j \not\to i$. We set also $\mathcal{T}_i = \Gamma^{+\star}(i) \cup Indep(i)$. Consider release and deadline vectors r and d and a task $i \in \mathcal{T}$. For any value $t \in \{r_i, \ldots, d_i - p_i\}$ corresponding to a possible starting time of i, we define t-dependent temporary release dates and deadlines for tasks in $\mathcal{T}_i \cup \{i\}$ as:

$$\hat{r}_j(t) = \begin{cases} \max\{r_j, t + \ell^\star_{ij}\} & \text{if } j \in \Gamma^{+\star}(i) \\ t & \text{if } i = j \\ r_j & \text{if } j \in Indep(i) \end{cases} \quad \text{and } \hat{d}_j(t) = \begin{cases} d_j & \text{if } j \in \mathcal{T}_i \\ t + p_i & \text{if } j = i. \end{cases}$$

Consider a function EXISTENCE(i, t, r, d) which checks the feasibility of the preemptive relaxation of the problem for tasks in $\mathcal{T}_i \cup \{i\}$ with release dates $\hat{r}(t)$, due dates $\hat{d}(t)$ and the m machine constraint. In a preemptive schedule each task might be interrupted and resumed on different machines.

EXISTENCE(i, t, r, d)
Input: A task $i \in \mathcal{T}$, release dates and deadlines vectors r and d, and $t \in \{r_i, \ldots, d_i - p_i\}$.
Question: Is there a feasible preemptive schedule of tasks from $\mathcal{T}_i \cup \{i\}$ meeting the release dates $\hat{r}(t)$ and the deadlines $\hat{d}(t)$?

This decision problem belongs to the class $P|r_i, d_i, pmtn|\star$. As shown by Martel [25], it can be transformed polynomially into a network flow problem and thus polynomially solved using a classical maximum-flow algorithm [14].

Let us now define the function BACKWARDSCHEDULE(i, r, d) that returns the maximum value $t^\star \in \{r_i, \ldots, d_i - p_i\}$ such that EXISTENCE$(i, t^\star, r, d)$ is true. If such a value exists, $t^\star + p_i$ is an upper bound of the completion time of i in any feasible schedule of the initial scheduling problem. Otherwise, no feasible schedule exists and the function returns false. We will discuss in the following several implementations of this function.

Algorithm 3 presents the extended version of the LPP algorithm. Tasks are first sorted by decreasing release date. Deadlines of the tasks are then improved iteratively in this order using the previous function BACKWARDSCHEDULE. The calls to PROPAGATE maintain consistent deadlines considering precedence constraints.

Algorithm 3 eLPP algorithm

Require: A precedence graph $\mathcal{G}$, release dates r, deadline d, processing times p and m identical processors
Ensure: Modified deadlines $d^\star$ or infeasibility
1: Adjust all r_i in topological order to reflect precedence
2: Adjust all d_i in reverse topological order to reflect precedence
3: Renumber tasks such that $r_1 \geq r_2 \geq \ldots \geq r_n$
4: **for** $i = 1$ to n **do**
5: resultB=BACKWARDSCHEDULE(i, r, d)
6: **if** not resultB **then**
7: **return** false
8: **end if**
9: d_i =resultB
10: d =PROPAGATE(i, d), return false if inconsistency with release times
11: **end for**
12: **return** $d^* = (d_i)_{i \in \mathcal{T}}$

The main problem addressed below is that no usual binary search on t can be considered to solve BACKWARDSCHEDULE because of the resource con-

straint for the task i. Indeed, a binary search can be considered to compute $t^\star$ if ExistenceR(i,t,r,d) = true for each value $t \in \{r_i, \dots t^\star\}$, and ExistenceR$(i,t,r,d)$ = false for each value $t \in \{t^\star + 1, \dots d_i - p_i\}$. This property on $t^\star$ is not verified.

Then, a simple approach to solve the optimization problem BackwardSchedule would be to start with $t = d_i - p_i$ and check each integer value in decreasing order until ExistenceR(i,t,r,d) is true. The number of steps would then be not polynomially bounded. Two implementations of BackwardSchedule were developed in the following to cope with this problem.

4.2 Weak eLPP algorithm

The simplest way to speed-up the time complexity of the eLPP algorithm is to limit the function ExistenceR to tasks from $\mathcal{T}_i$ instead of $\mathcal{T}_i \cup \{i\}$. Indeed, if the task i is removed, the problem ExistenceR(i,t,r,d) is more constrained when t increases, and thus a binary search on t can be considered to implement BackwardSchedule. The complexity of the deadlines reduction algorithm is in polynomial time in this case as proven in Theorem 3. However, the deadlines obtained might be greater than the ones given by Algorithm 3.

Theorem 3. *The weak eLPP algorithm is in time* $\mathcal{O}(n^4 \times \max_{i\in\mathcal{T}} \log(d_i - p_i - r_i))$.

Proof. For each task $i \in \mathcal{T}$ and any value $t \in \{r_i, \dots, d_i - p_i\}$, the time complexity for the computations of the vectors $\hat{r}$ and $\hat{d}$ is $\mathcal{O}(n^2)$. The number of nodes (resp. arcs) of the graph associated with the flow problem belongs to $\mathcal{O}(n)$ (resp. $\mathcal{O}(n^2)$) [25]. The time complexity of the flow algorithm is in $\mathcal{O}(n^3)$ using a push-relabel algorithm with a FIFO vertex selection rule [14]. As $t^\star$ is computed using a binary search in the time interval $\{r_i, \dots, d_i - p_i\}$, we conclude that the overall time complexity of weak eLPP algorithm is $\mathcal{O}(n^4 \times \max_{i\in\mathcal{T}} \log(d_i - p_i - r_i))$, proving the theorem. □

4.3 Strong eLPP algorithm

The purpose of the strong version of eLPP is to develop an implementation of Algorithm 3 that is faster but remains exact. Let us consider the relaxed decision problem ExistenceR(i,u,v,r,d) defined as follows which dissociates the parameters for the computation of the release dates and deadlines:

ExistenceR(i,u,v,r,d)
Input: A task $i \in \mathcal{T}$, release dates and deadlines vectors r and d, and a pair of values $(u,v) \in \{r_i, \dots, d_i - p_i\}^2$ with $u \geq v$.
Question: Is there a feasible preemptive schedule of tasks from $\mathcal{T}_i \cup \{i\}$ meeting the release dates $\hat{r}(v)$ and the deadlines $\hat{d}(u)$?

This decision problem also belongs to the class $P|r_i, d_i, pmtn|\star$. Thus, as for ExistenceR it can be solved using Martel's transformation [25] to a network flow

problem coupled with a classical maximum-flow algorithm [14]. We can also note that EXISTENCE is a special case of EXISTENCER for which $t = u = v$.

Now, let us define the function BACKWARDSCHEDULER(i, u, r, d) with $u \in \{r_i, \ldots, d_i - p_i\}$ that returns the maximum value $v^\star \in \{r_i, \ldots, u\}$ such that EXISTENCER$(i, u, v^\star, r, d)$ is true if any, and false otherwise. Observe that, for any $(v, v') \in \{r_i, \ldots, d_i - p_i\}^2$ with $v < v'$, EXISTENCER(i, u, v, r, d) is less constrained than EXISTENCER(i, u, v', r, d). Thus, if EXISTENCER$(i, u, v', r, d) =$ true, then so is EXISTENCER(i, u, v, r, d) and a binary search can be considered to solve BACKWARDSCHEDULER.

The remaining problem is then to find the maximal fixed point of BACKWARDSCHEDULER, that is, $u^\star$ such that $u^\star =$ BACKWARDSCHEDULER$(i, u^\star, r, d)$. The next lemma establishes the relationship between BACKWARDSCHEDULER and BACKWARDSCHEDULE.

Lemma 3. *For any task $i \in \mathcal{T}$, release dates and deadlines vectors r and d, the value $u^\star =$ BACKWARDSCHEDULER$(i, u^\star, r, d)$ exists if and only if the value $t^\star =$BACKWARDSCHEDULE(i, r, d) exists. Moreover, $u^\star = t^\star$.*

Proof. Assume first that $u^\star$ exists; then EXISTENCER$(i, u^\star, u^\star, r, d) =$ true and thus EXISTENCE$(i, u^\star, r, d) =$ true. BACKWARDSCHEDULE(i, r, d) then returns an optimal value $t^\star \geq u^\star$. Conversely, let us suppose that $t^\star$ exists; then EXISTENCE$(i, t^\star, r, d) =$ true. The consequence is that EXISTENCER$(i, t^\star, t^\star, r, d) =$ true, thus $u^\star$ exists and $t^\star \leq u^\star$. □

The following lemma shows an important property of the function BACKWARDSCHEDULER.

Lemma 4. *Let us consider $i \in \mathcal{T}$, release dates and deadlines vectors r and d. For $u \in \{r_i, \ldots, d_i - p_i\}$, the function $u \rightarrow$BACKWARDSCHEDULER(i, u, r, d) is non decreasing (if it returns an integer).*

Proof. Assume that u and u' are two integers in $\{r_i, \ldots, d_i - p_i\}$ with $u' < u$. If BACKWARDSCHEDULER$(i, u', r, d) = v' \in \{r_i, \ldots, u'\}$, then we get EXISTENCER$(i, u', v', r, d) =$ true. Since $u' < u$, EXISTENCER$(i, u, v', r, d) =$ true and $v' \leq u' < u$. Thus, BACKWARDSCHEDULER(i, u, r, d) exists and $v' \leq v$. □

We now show how to compute the value $u^\star$. This can be done by computing a sequence of upper bounds u_β, $\beta \geq 0$ of $u^\star$ that converges to $u^\star$. Indeed, let us consider the sequence of integers u_β defined as:

1. $u_0 = d_i - p_i$;
2. For any $\beta > 0$, $u_\beta =$BACKWARDSCHEDULER$(i, u_{\beta-1}, r, d)$.

The next theorem shows the convergence of this sequence to $t^\star$.

Theorem 4. *If $t^\star$ exists, the sequence u_β tends to $t^\star$ (ie. there exists $\beta^\star \in \mathbb{N}$ such that $u_{\beta^\star} = t^\star$).*

Proof. We first prove that, for any value $\beta \in \mathbb{N}$, $u_\beta = t^\star$ or $u_0 > u_1 > \ldots > u_\beta \geq t^\star$. Indeed, $u_0 = d_i - p_i \geq t^\star$. Now, let us suppose by recurrence that $u_0 > u_1 > \ldots > u_\beta \geq t^\star$ for $\beta \geq 1$. By Lemma 4, the function BackwardScheduleR is non decreasing with respect to u, thus since $u_{\beta-1} > u_\beta$, we get $u_\beta \geq u_{\beta+1}$.

1. If $u_\beta = u_{\beta+1}$, then by definition of $u^\star$, $u_\beta = u^\star = u_{\beta+1}$ and thus by Lemma 3, $u_{\beta+1} = t^\star$;
2. Let us suppose now that $u_\beta > u_{\beta+1}$. Then, since $u_\beta > t^\star$, we get by Lemma 4 that $u_{\beta+1} \geq t^\star$.

Lastly, since the sequence u_β is strictly decreasing until it reaches $t^\star$, there exists a minimum integer $\beta^\star$ such that $u_{\beta^\star} = t^\star$, and the theorem is proved. □

The implementation of BackwardSchedule based on BackwardScheduleR simply consists of computing the sequence u_β until a fixed point is reached. Alas, we do not have any polynomial upper bound of the time complexity of this algorithm.

Theorem 5. *The strong eLPP algorithm is in time* $\mathcal{O}(n^4 \times \max_{i \in \mathcal{T}}(d_i - p_i - r_i))$.

Proof. For each task $i \in \mathcal{T}$, the number of executions of BackwardScheduleR is bounded by $d_i - p_i - r_i$ to compute $t^\star$. Following the same arguments as the proof of Theorem 3, we get the upper bound of the time complexity. □

5 Data generation for tests

This section describes the testing procedure of the four algorithms eGJ and eLPP in their strong and weak forms. Subsection 5.1 presents the algorithm considered to build the instances of the scheduling problem $P|prec, r_i, d_i|\star$, while Subsection 5.2 explains how the parameters were chosen to get significant instances. The minimal consistent date of an algorithm is introduced in Subsection 5.3 to compare the performances of the algorithms, followed by the experiment conditions.

5.1 Description of the instances

The parameters considered for the instances description are the number of tasks n, the number of machines m, the maximum processing time p_{max}, the probability of arc creation p, and the upper limit Δ for release dates and tails.

Recall that the tail q_i of a task i is the minimum length between the end of the task i and the end of the schedule, ie. $C_i + q_i \leq C$. Once a lower bound C^- of the makespan is fixed, we set $d_i = C^- - q_i$ for each task $i \in \mathcal{T}$.

Algorithm 4 generates a random instance $\mathcal{I}$ of $P|prec, r_i, d_i|\star$. Lines 1−2 build a directed acyclic graph $\mathcal{G}$ from a random non directed graph $H = G_{n,p}$ [11]. Edges of H are then transformed into arcs of $\mathcal{G}$ by setting the arc $a = (i, j)$ if $e = \{i, j\}$ with $i < j$. Lines $3 - 4$ randomly generate p, r and q, while line 5

adjusts r and q such that, for each task i, $r_i = \max(r_i, \max_{j \in \Gamma^-(i)}(r_j + p_j))$ and $q_i = \max(q_i, \max_{j \in \Gamma^+(i)}(q_j + p_j))$. List scheduling via Earliest Due Date, constrained to respect precedence, is considered in line 6 to generate an upper bound C^+ of the makespan for precedence and resource constraints.

For a fixed value of the makespan C, the instance can be relaxed to an instance of $P|r_i, d_i|\star$ by ignoring precedence relations and allowing preemption. As noted above, this last problem can be transformed into a flow problem using the Martel's transformation [25] and thus solved with a classical maximum-flow algorithm [14]. A lower bound C^- of the initial problem is then obtained using a binary search on C at line 7.

We observe that if $C^+ = C^-$, the list schedule is optimal, and thus the instance is discarded for our testing. Otherwise, the deadlines are deduced from C^- and q at line 11.

Algorithm 4 Generation of a random instance $\mathcal{I}$ of $P|prec, r_i, d_i|\star$

Require: Positive integers n, m, p_{max}, p and Δ
Ensure: An instance $\mathcal{I}$ of $P|prec, r_i, d_i|\star$
1: Construct a (non oriented) graph $H = G_{n,p}$ [11]
2: Build a precedence graph $\mathcal{G}$ by transforming each edge $e = \{i, j\}$ of the graph $H = (\{1, \ldots, n\}, E)$ with $i < j$ to an arc (i, j) of $\mathcal{G} = (\{1, \ldots, n\}, \mathcal{A})$
3: Generate random values of p_i in $[1, p_{max}]$ for $i \in \{1, \ldots, n\}$
4: Generate random values of r_i and q_i in $[1, \Delta]$ for $i \in \{1, \ldots, n\}$
5: Adjust r and q to account for precedence
6: Run list scheduling algorithm to get upper bound on optimal makespan, C^+
7: Perform a binary search on the problem without precedence relations and allowing preemption, C^-
8: **if** $C^+ = C^-$ **then**
9: **return** false
10: **end if**
11: For each task $i \in \{1, \ldots, n\}$ set $d_i = C^- - q_i$
12: **return** $\mathcal{I} = (\mathcal{G}, p, r, d)$

5.2 Choice of parameters

Parameters for the experiments were fixed to keep the problem size manageable and to generate non trivial comparable instances with respect to the deadline reduction measures.

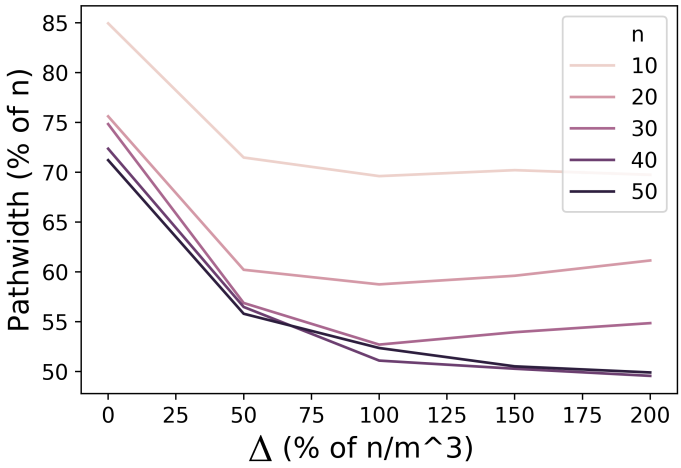
The number of tasks n varied from 10 to 50, while $p_{max} \in [1, 5]$. High values of m make the problem trivial, thus we set $m \in [1, 3]$. The probability of arc creation p was fixed to 0.2 to get precedence graphs with a reasonable amount of connectedness. We generated 10 instances for each combination of input parameters.

The remaining parameter is Δ, the maximum value of r_i and q_i. To fix this parameter we first studied its influence on the parallelism of the instance, measured by the pathwidth.

The pathwidth pw of an instance $\mathcal{I}$ is the maximum number of tasks that can be executed simultaneously considering only release dates and deadlines. This measure was introduced by Munier [26] as the parameter of a fixed-parameter algorithm for $P|prec, r_i, d_i = D, p_i = 1|C_{\max}$.

The relationship between the pathwidth of the generated instances and Δ is not straightforward. For each value of n, the pathwidth is highest when $\Delta = 0$ (in this case, release dates and deadlines are only defined by the precedence relations) and decreases as Δ increases, eventually stabilizing at a roughly constant level. By trial and error, we discovered that $\frac{pw}{n}$ is a piecewise linear function of $\frac{m^3}{n}\Delta$, with only two thresholds as shown in Figure 2. The values of the thresholds for Δ are approximately $\lfloor \frac{n}{2m^3} \rfloor, \lfloor \frac{n}{m^3} \rfloor$. By choosing $\Delta \in \{0, \lfloor \frac{n}{2m^3} \rfloor, \lfloor \frac{n}{m^3} \rfloor\}$, we allow a maximum amount of pathwidth variation in the input instances to be captured.

Fig. 2: Variation of pathwidth by Δ and n



5.3 Minimal consistent dates and testing procedure

The aim of our experiments was to compare the strong and the weak versions of the eGJ and eLPP algorithms. For this purpose, we defined for each instance $\mathcal{I}$ and each algorithm A, the value $\delta_A^\star(\mathcal{I}) \in \mathbb{Z}$ which is the smallest value such that the algorithm A does not detect infeasibility if, for each task $i \in \mathcal{T}$, $d_i + \delta_A^\star(\mathcal{I})$ is used as the deadline.

For any instance $\mathcal{I}$, the values $\delta^\star_A(\mathcal{I})$ and $\delta^\star_{A'}(\mathcal{I})$ can be considered as a performance measure of algorithms A and A'. Indeed, if $\delta^\star_A(\mathcal{I}) > \delta^\star_{A'}(\mathcal{I})$, then A detects an infeasibility for $d + \delta_{A'}(\mathcal{I})$, and thus A beats A'.

For each couple $(A, \mathcal{I})$, this value can be found by binary search. The lower-bound in the beginning of the binary search is given by solving the preemptive relaxation of $\mathcal{I}$ with no precedence constraints. An upper bound can be computed using an Earliest Due Date priority list scheduling algorithm, constrained to respect precedence. For each problem instance $\mathcal{I}$ generated, this binary search to find $\delta^\star_A(\mathcal{I}) \in \mathbb{Z}$ was applied using each algorithm (eLPP and eGJ, weak and strong forms) ie. for $A \in \{\text{s-eGJ}, \text{s-eLPP}, \text{w-eGJ}, \text{w-eLPP}\}$.

The algorithms were implemented using Python 3.7.6 coupled with the packages numpy 1.18.1 and networkX 2.4. All our experiments were performed on an Acer Swift SF314-41 composed of an AMD Ryzen 5 3500U running at 2.1Ghz with 4 cores, 8 Logical Processors and 8MB RAM.

6 Experiments

This section is devoted to the description of our experiments' results. Subsection 6.1 motivates in an experimental sense the choice of the shortest augmenting path flow algorithm [10] with an initial flow built using Jackson's preemptive algorithm [20] to solve the two problems Existence and ExistenceR. Subsection 6.2 compares the running times of our four algorithms, while Subsection 6.3 deals with their output analysis.

6.1 eLPP efficiency choices

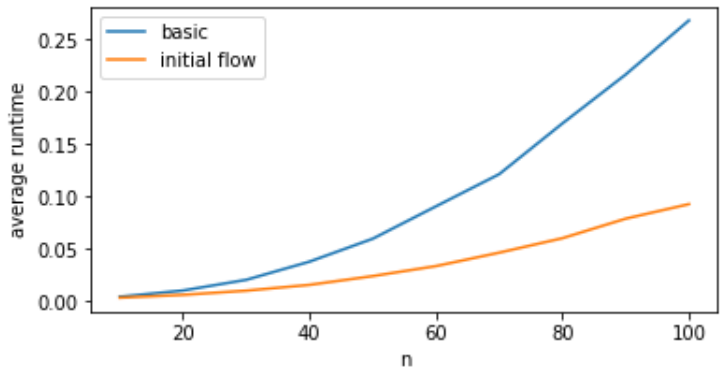
Before beginning comparison of the algorithms, the choice of the maximum flow algorithm is discussed to find the most efficient implementation for the problems Existence and ExistenceR. As expected, this function took up the majority (98%) of eLPP runtime, so these choices had significant impact on time complexity.

Three maximum flow algorithms were considered: preflow push [13], shortest augmenting path [10], and Edmonds Karp [10]. Shortest augmenting path had the best performance on this dataset; preflow push and Edmonds Karp took longer by 11% and 9% respectively. The gap widened as n increased in both cases; however, for preflow push it narrowed as m increased, indicating that this may be more suitable for $m > 3$.

An initial solution attempt was constructed by using Jackson's preemptive schedule [20]. This was tested against the basic approach on randomly generated data (Figure 3) with n extended up to 100 to more accurately assess complexity. On average, the runtime of Existence and ExistenceR was reduced by 47%, and the complexity, determined by a log-log regression of runtime against n, was reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n^{1.6})$.

In practice in the e-LPP tests, runtime was reduced by 36%. In 2% of calls to the function, a solution was found immediately by Jackson's preemptive algorithm [20] so that the maximum flow algorithm was bypassed.

Fig. 3: ExistenceR runtime with and without an initial solution following the number of tasks



In the following, s-eLPP (resp. w-eLPP) was therefore implemented with ExistenceR (resp. Existence) using a shortest augmenting path algorithm supplied with an initial flow.

6.2 Complexity analysis

Here the runtime of each of the four algorithms is compared, as well as how they change with each of the problem parameters.

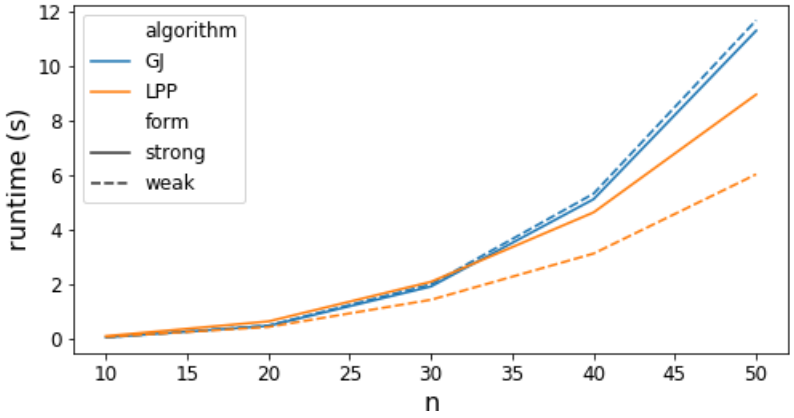
As shown in Figure 4, up to $n = 30$, eGJ performed similarly to eLPP, but eLPP was faster for higher values of n. The weak version of eGJ made no appreciable difference to the speed, but for eLPP the weak version was faster about 33% on average.

The runtime for each problem instance was regressed against n, m, p_{max}, $\frac{pw}{n}$ and the choice of algorithm. The runtime and number of tasks were both log transformed to help meet the model assumptions and identify the order of complexity. 95% of the variation in runtime was able to be explained by these variables. The results were as follows:

- Though neither strong form is proven to be polynomial, in practice both eGJ forms were about $\mathcal{O}(n^{3.4})$, while the eLPP forms were about $\mathcal{O}n^{2.8}$).
- Increasing m by 1 resulted in a 38% decrease in runtime for eGJ, and a 30% decrease for eLPP.
- Increasing p_{max} by 1 increased runtime by 26% for eGJ, and 18% for eLPP.
- Increasing pathwidth by $\frac{n}{10}$ increased runtime by 2% for eGJ, and 9% for eLPP.

As far as speed is concerned, eLPP is clearly the stronger option. It was also less sensitive to increases in most of the parameters, so the trend can be expected to continue beyond the range tested. The usefulness of the weak form of eLPP depends on the results of the following section.

Fig. 4: Comparison of algorithm runtimes following the number of tasks



6.3 Output analysis

If we compare the results of the strong forms s-eGJ vs. s-eLPP, in all but 0.2% of problem instances, the final $\delta_A^\star$ values matched; in 96%, each individual date matched as well. Most of the 4% mismatches were due to one or two slightly higher s-eLPP deadlines. Even when the dates did not match, the differences were few and small, and overall, the s-eGJ dates dominated a majority of the time (but not all the time). When $n = 50$, neither algorithm has a significant advantage over the other in terms of date reduction.

The weak forms w-eGJ and w-eLPP had a $\delta_A^\star$ value that was 1 lower in about 1% of cases for each compared with the respective strong forms s-eGJ and s-eLPP. This decreased with n and increased with Δ. Only 55% of the final deadlines matched their strong counterparts. The difference between the dates produced was often in several tasks and with values larger than 1.

Four metrics, defined below, were calculated for each instance and averaged across all instances.

Percentage of modified instances a 0/1 flag indicates whether any date was modified in an instance;

Percentage of modified tasks the proportion of individual deadline values modified;

Interval shrinkage the reduction as a proportion of the available intervals, so $1 - \frac{\sum_{i=1}^{n}(\tilde{d}_i - \tilde{r}_i)}{\sum_{i=1}^{n}(d_i - r_i)}$ where r, d are the initial dates and $\tilde{d}, \tilde{r}$ the final dates;

Pathwidth reduction the reduction in pathwidth pw as a percentage of the original.

Table 1 summarises the metrics discussed above by algorithm, across all test instances. As the previous results suggested, the strong forms were nearly

identical. The amount of reduction made by the weak forms was slightly less, though the number of modified dates was similar.

Table 1: Date modifications across all tests

	s-eGJ	s-eLPP	w-eGJ	w-eLPP
% instances modified	81%	81%	79%	79%
% tasks modified	34.2%	34.1%	32.4%	32.4%
Interval shrinkage	12.0%	12.0%	9.4%	9.5%
Pathwidth reduction	8.9%	8.9%	7.2%	7.2%

Reductions improved with n for all algorithms; Table 2 shows the same statistics when $n = 50$. Dates were also reduced more with smaller m, and with larger pathwidth pw (as a proportion of n). In particular, the interval shrinkage was approximately halved with each addition of a machine. The gap between the weak and strong forms narrowed slightly as n increased.

Table 2: Date modifications for $n = 50$

	s-eGJ	s-eLPP	w-eGJ	w-eLPP
% instances modified	98%	98%	97%	97%
% tasks modified	52.8%	52.9%	52.1%	52.2%
Interval shrinkage	18.7%	18.8%	16.9%	16.9%
Pathwidth reduction	13.6%	13.6%	12.9%	12.9%

With such little difference between the outputs of the strong forms, eLPP maintains its advantage from the runtime results. The weak form offers a trade-off; while it is considerably faster, less reduction is performed. It may be useful in time-sensitive contexts where incremental reductions are relatively less valuable, particularly if n is large.

7 Conclusions

We developed in this paper several extensions of the GJ and the LPP algorithms to handle tasks with different processing times and precedence relations. The aim here was to evaluate whether considering at the same time precedence and resource constraints in deadline reduction algorithm was an interesting approach.

Two versions of each algorithm was developed: the weak ones (of polynomial time complexity), and the strong ones (non polynomially time bounded complexity). The strong version of the two extensions improves slightly the results, with experimentally the same complexity as their weak counterpart. As the LPP extensions outperforms the GJ ones in terms of theoretical as well as

experimental complexity, it should be preferred. However, their time complexity remains still large, and further improvement should be investigated, inspired by the recent improvements of the computational complexity of energetic reasoning for problems without precedence constraints [3,27].

Our approach that embeds precedence and resources should be experimentally compared to a process of usual precedence relaxation to compute reduced deadlines followed by precedence propagation, repeated iteratively. An interesting further study would compare the results of several interval reduction techniques, in particular the one proposed by Haouari et al. [18,19].

Most of the algorithms that have been proposed for problems with parallel processors extend quite naturally to cumulative resources. The extension of eGJ and eLPP to such problems might be easier for eGJ.

Finally, the aim of the algorithms presented in this paper is to improve the efficiency of either branch and bound or constraint programming algorithms. This should be experimentally investigated in subsequent research.

References

1. Baptiste, P., Le Pape, C., Nuijten, W.: Satisfiability tests and time-bound adjustments for cumulative scheduling problems. Annals of Operations Research **92**(0), 305–333 (Jan 1999)
2. Bellenguez-Morineau, O.: Methods to solve multi-skill project scheduling problem. 4OR **6**(1), 85–88 (2008)
3. Bonifas, N.: A $\mathcal{O}(n^2 \log(n))$ propagation for the energy reasoning. In: Congrès ROADEF (02 2016)
4. Carlier, A., Hanen, C., Munier Kordon, A.: The equivalence of two classical list scheduling algorithms for dependent typed tasks with release dates, due dates and precedence delays. J. Scheduling **20**(3), 303–311 (2017)
5. Carlier, J., Latapie, B.: Une méthode arborescente pour résoudre les problèmes cumulatifs. RAIRO - Operations Research - Recherche Opérationnelle **25**(3), 311–340 (1991)
6. Carlier, J., Pinson, E., Sahli, A., Jouglet, A.: An $\mathcal{O}(n^2)$ algorithm for time-bound adjustments for the cumulative scheduling problem. European journal of Operations Research **286**(2), 468–476 (2020)
7. Carlier, J., Pinson, E., Sahli, A., Jouglet, A.: Comparison of three classical lower bounds for the cumulative scheduling problem. (submitted) (2021)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition. The MIT Press, 3rd edn. (2009)
9. Derrien, A., Petit, T.: A new characterization of relevant intervals for energetic reasoning. In: O'Sullivan, B. (ed.) Principles and Practice of Constraint Programming CP2014, Lyon, France. Lecture Notes in Computer Science, vol. 8656, pp. 289–297. Springer (2014)
10. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM (JACM) **19**(2), 248–264 (1972)
11. Erdös, P., Réwi, A.: On random graphs i. Publicationes Mathematicae **6**(290-297), 18 (1959)
12. Garey, M.R., Johnson, D.S.: Two-processor scheduling with start-time and deadlines. SIAM Journal on Computing **6**, 416–426 (1977)

13. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum-flow problem. Journal of the ACM (JACM) **35**(4), 921–940 (1988)
14. Goldberg, A.V., Tarjan, R.E.: Efficient maximum flow algorithms. Communications of the ACM **57**(8), 82–89 (2014)
15. 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)
16. Hanen, C., Munier Kordon, A.: Two deadline reduction algorithms for scheduling dependent typed-tasks systems. In: ROADEF conference (2020)
17. Hanen, C., Zinder, Y.: The worst-case analysis of the Garey-Johnson algorithm. Journal of Scheduling **12**(4), 389–400 (2009)
18. Haouari, M., Kooli, A., Néron, E.: Enhanced energetic reasoning-based lower bounds for the resource constrained project scheduling problem. Comput. Oper. Res. **39**(5), 1187–1194 (2012)
19. Haouari, M., Kooli, A., Néron, E., Carlier, J.: A preemptive bound for the resource constrained project scheduling problem. J. Sched. **17**(3), 237–248 (2014)
20. Jackson, J.R.: Scheduling a production line to minimize maximum tardiness. management science research project (1955)
21. Laborie, P.: Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. Artif. Intell. **143**(2), 151–188 (2003)
22. Laborie, P., Nuijten, W.: Constraint Programming Formulations and Propagation Algorithms, chap. 4, pp. 63–72. John Wiley & Sons, Ltd (2008)
23. Leung, A., Palem, K.V., Pnueli, A.: Scheduling time-constrained instructions on pipelined processors. ACM Trans. Program. Lang. Syst. **23**, 73–103 (January 2001)
24. Lombardi, M., Milano, M.: Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. Constraints An Int. J. **17**(1), 51–85 (2012)
25. Martel, C.: Preemptive scheduling with release times, deadlines, and due times. Journal of the Association of Computing Machinery **29**(3), 812–829 (Jul 1982)
26. Munier Kordon, A.: A fixed-parameter algorithm for scheduling unit dependent tasks on parallel machines with time windows. Discrete Applied Mathematics **290**, 1 – 6 (2021)
27. Ouellet, Y., Quimper, C.: A $\mathcal{O}(n \log^2 n)$ checker and $\mathcal{O}(n^2 \log n)$ filtering algorithm for the energetic reasoning. In: van Hoeve, W.J. (ed.) CPAIOR 2018, Delft, The Netherlands, June 26-29. Lecture Notes in Computer Science, vol. 10848, pp. 477–494. Springer (2018)
28. Tesch, A.: Improving energetic propagations for cumulative scheduling. In: Hooker, J.N. (ed.) CP 2018, Lille, France, August 27-31. Lecture Notes in Computer Science, vol. 11008, pp. 629–645. Springer (2018)
29. Ullman, J.: NP-complete scheduling problems. J. Comput. System Sci. **10**, 384–393 (1975)