An effective branch-and-price algorithm for the Preemptive Resource Constrained Project Scheduling Problem based on minimal Interval Order Enumeration

Aziz Moukrim, Alain Quilliot, Hélène Toussaint

An effective branch-and-price algorithm for the Preemptive Resource Constrained Project Scheduling Problem based on minimal Interval Order Enumeration*

Aziz Moukrim[a], Alain Quilliot[b], Hélène Toussaint[c]

[a] *Université de Technologie de Compiègne, Heudiasyc, CNRS UMR 7253*
Rue Roger Couttolenc, CS 60319, 60203 Compiègne, France
Email: aziz.moukrim@hds.utc.fr
[b] *LIMOS CNRS UMR 6158, LABEX IMOBS3, Université Blaise Pascal*
Bat. ISIMA, BP 10125, Campus des Cézaux, 63173 Aubière, France
Email: quilliot@isima.fr
[c] *LIMOS CNRS UMR 6158, LABEX IMOBS3, CNRS*
Bat. ISIMA, BP 10125, Campus des Cézaux, 63173 Aubière, France
Email: toussain@isima.fr

Abstract

In this paper we address the Preemptive Resource Constrained Project Scheduling Problem (PRCPSP). PRCPSP requires a partially ordered set of activities to be scheduled using limited renewable resources such that any activity can be interrupted and later resumed without penalty. The objective is to minimize the project duration. This paper proposes an effective branch-and-price algorithm for solving PRCPSP based upon minimal interval order enumeration involving column generation as well as constraint propagation. Experiments conducted on various types of instances have given very satisfactory results. Our algorithm is able to solve to optimality the entire set of J30, BL and Pack instances while satisfying the preemptive requirement. Furthermore, this algorithm provides improved best-known lower bounds for some of the J60, J90 and J120 instances in the non-preemptive case (RCPSP).

Keywords: Project scheduling, interval order, column generation, constraint propagation

1. Introduction

This paper deals with the *Resource Constrained Project Scheduling Problem* (RCPSP). RCPSP aims at scheduling a set of activities subject to precedence

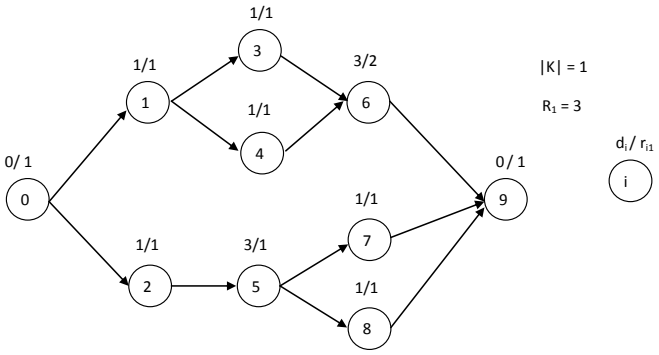
*A preliminary version of this work was communicated in the 6th Workshop on Computational Optimization 2013 [15].

and resource constraints, while minimizing the induced *makespan* (total duration of the project) value. The precedence constraints mean that some activities must be completed before others can start. The resource constraints specify that each activity requires constant amounts of renewable resources (with limited capacities) throughout the scheduling process. RCPSP has been extensively studied in its non-preemptive version where every activity has to be run as a whole without any kind of interruption. The *Preemptive Resource Constrained Project Scheduling Problem* (PRCPSP), on the other hand, corresponds to the case where each activity can be interrupted and later resumed without penalty.

There are not many works on PRCPSP. Demeulemeester and Herroelen [8] have developed a branch-and-bound algorithm; Ballestín et al. [2] have looked at preemption from a heuristic perspective; and Peteghem and Vanhoucke [16] have designed a genetic algorithm for multi-mode Preemptive RCPSP. For the sake of simplicity, authors often assume that all processing times are integral and that preemption only occurs at integer valued dates. It is, however, easily established that such a hypothesis is very restrictive and can produce only an approximation of the optimal value of the problem. We refer to the surveys published by Kolisch and Padman [12], Brucker et al. [5], Herroelen and Leus [10], Hartmann and Briskorn [9]. Recently, Schutt et al. [18] propose an effective algorithm for RCPSP based on Constraint Programming that allows many open instances to be closed and the lower bounds for RCPSP to be improved.

In this paper we consider the problem in its most general form and suppose that preemption is allowed for all activities and may occur at arbitrary rational dates, and that no penalties are related to preemption. Ours is a branch-and-price approach which involves constraint propagation, as well as the management of a specific rational *Antichain linear program* whose variables are associated with subsets of activities that may be simultaneously processed during the schedule. This Antichain linear program, first introduced by Mingozzi et al. [14], provides us with a lower bound of both preemptive and non-preemptive RCPSP. Mingozzi et al. [14] have proposed a time-indexed linear formulation for RCPSP involving antichains. This linear program approach requires the implementation of a pricing or column generation scheme. Brucker and Knust [4] combine constraint propagation techniques and this linear programming formulation in order to obtain lower bounds for RCPSP. It was proved in Damay et al. [7] that if the input RCPSP instance satisfies certain *ad hoc* properties, then any optimal solution of the Antichain linear program may be turned into a feasible optimal schedule without any increase in the makespan value. To the best of our knowledge, only the work of Damay et al. [7] deals with optimal solutions of PRCPSP without any restriction. Damay et al. propose a branch-and-bound method for PRCPSP based on the linear programming formulation of Mingozzi et al. [14] and introducing forbidden disjunctions.

The strategy described here is to use the linear programming formulation of Mingozzi et al. [14] in order to perform a tree search which may be viewed as being embedded into the process of enumerating all the minimal extensions of the precedence relation that define *interval orders*. The resulting process, capable of solving exactly all 30 activity instances in the PSPLIB library and improv-



a. An instance of RCPSP

1	3	6	6	6	6	7
2	4	6	6	6	6	8
	5	5	5	7	8	

0 1 2 3 4 4,5 5 5,5

1	3	6	6	6	8	
2	4	6	6	6		
	5	5	5	7		

0 1 2 3 4 5 6

b. A preemptive schedule with makespan 5.5 c. A non-preemptive schedule with makespan 6

1	2	5
7	3	6
8	4	6

0 1 2 5

d. An optimal solution of the Antichain linear Program

Figure 1: A preemptive instance example

ing the best existing lower bounds for several 60/90/120 activity instances in the same library, is seen to be particularly effective, with a powerful separation scheme based on a *forbidden structure of interval orders*.

The paper is organized as follows. We first recall the definition of Preemptive RCPSP in Section 2, and then introduce in Section 3 the theoretical tools related to the Antichain LP and to interval orders, which will be the basis of our new approach. Section 4 describes our IOE algorithm and its implementation, and Section 5 is devoted to the presentation of experimental results.

2. Problem description

An instance $I = (X, K, \prec)$ of the *Resource Constrained Project Scheduling Problem* is defined by:

- A set $X = \{1, ..., n\}$ of n activities: $\forall i \in X, d_i$ denotes the *duration* of activity i.
- A set $K = \{1, ..., m\}$ of m *resources*: $\forall i \in X, \forall k \in K, r_{ik}$ denotes the requirement for resource k by activity i. These resources are given back to the system once the activity is over or interrupted.
- $\forall i, j \in X, i \prec j$ means that i *precedes* j: activity j cannot start before i is over (*Precedence* constraints).

By convention we also introduce two activities 0 and $n + 1$ to respectively represent the start and the end of the schedule. Hence, activity 0 (resp. $n + 1$) is a predecessor (resp. successor) of all the other activities. Furthermore, we set $d_0 = d_{n+1} = 0$ and $r_{0k} = r_{n+1,k} = 0$ for each $k \in K$.

In the case of *Non-Preemptive* RCPSP, scheduling only means computing the starting times $t_i (i \in X)$ of the activities. A schedule $\sigma = (t_i, i \in X)$ is feasible if it satisfies the *Precedence* constraints and the *Resource* constraints. At any time t during the process, and for any resource k, the sum $\sum_{i \in Act(\sigma,t)} r_{ik}$ does not exceed the global resource amount R_k, $Act(\sigma, t) = \{i \in X, t_i \leq t < t_i + d_i\}$ denoting the set of the activities running concurrently at time t according to schedule σ. So, solving *non-preemptive* RCPSP means computing σ with a minimal *makespan* (total duration of the project).

In cases where *preemption* is allowed, scheduling an activity i means first decomposing i into a sequence of sub-activities $i_1, .., i_{h(i)}$, with durations $d_{i,1}, ...,$ $d_{i,h(i)}$, such that: $\sum_{q=1...h(i)} d_{i,q} = d_i$, and next scheduling all these sub-activities in the same way as for standard RCPSP. We also introduce for any feasible schedule σ, $StartTime(\sigma, i)$ and $EndTime(\sigma, i)$ to respectively denote the starting time and finishing time of activity i which is $t_{i,1}$ (respectively $t_{i,h(i)} + d_{i,h(i)}$). In this work we assume there are no restrictions either on the number of sub-activities or on their durations, which may be arbitrarily small.

Let us consider one resource with capacity $R_1 = 3$ and 8 activities with duration and resource requirements as described in Figure 1.a. The activities have durations $d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 3, d_6 = 3, d_7 = 1$ and $d_8 = 1$. The activity requirements are such that $r_{1,1} = r_{2,1} = r_{3,1} = r_{4,1} = r_{5,1} = r_{7,1} = r_{8,1} = 1$ and $r_{6,1} = 2$. We assume further that there are precedence constraints: $1 \prec 3, 1 \prec 4, 2 \prec 5, 3 \prec 6, 4 \prec 6, 5 \prec 7$ and $5 \prec 8$.

A feasible preemptive (resp. non-preemptive) schedule with makespan 5.5 (resp. 6) is given in Figure 1.b. (resp. Figure 1.c.).

3. Basic tools

Let $I = (X, K, \prec)$ be some Preemptive RCPSP instance, defined according to the notation given in Section 2. We suppose (as we are clearly entitled to do) that the precedence relation $\prec$ is transitive. Then we define an *antichain* as being any subset a of X such that there does not exist $i, j \in a$ such that $i \prec j$. We say that such an antichain is *valid* if: $\forall k \in K, \sum_{i \in a} r_{ik} \leq R_k$. It follows that a subset $a \subseteq X$ of activities is a *valid antichain* if and only if the activities belonging to a may be simultaneously run inside some feasible schedule. We denote the set of all valid antichains as $\mathcal{A}$.

3.1. Antichain Linear Program

We are now able to express the following linear program, known as an *Antichain Linear Program*, associated with a Preemptive RCPSP instance $I = (X, K, \prec)$, introduced in Mingozzi et al. [14], also used in Damay et al. [7], and which we shall denote as $\mathcal{P}_{\mathcal{A}}$:

$$\text{Minimize} \sum_{a \in \mathcal{A}} z_a \tag{1}$$

s. t.

$$\forall i \in X, \sum_{a \in \mathcal{A} | i \in a} z_a = d_i \tag{2}$$

$$\forall a \in \mathcal{A}, z_a \geq 0 \tag{3}$$

Let σ be any feasible schedule related to instance I, and for any valid antichain a let $z(\sigma)_a$ denote the total amount of time during which the activities simultaneously running in σ correspond precisely to the activities in a. We can see that $z(\sigma) = (z(\sigma)_a, a \in \mathcal{A})$ is a feasible solution of $\mathcal{P}_\mathcal{A}$, since constraints (2) express the fact that any activity i needs to have completed, or, equivalently, that the duration of all antichains containing i must be equal to the duration of i. It follows that the optimal value of $\mathcal{P}_\mathcal{A}$ provides us with a lower bound of the optimal value of I, which we denote as $LB(I)$.

Let us consider the instance described in Figure 1.a. Figure 1.d. shows an optimal solution z of the Antichain linear program with $z_{a_1} = 1, z_{a_2} = 1, z_{a_3} = 3$ and $z_a = 0 \ \forall a \in \mathcal{A} - \{a_1, a_2, a_3\}$ where $a_1 = \{1, 7, 8\}, a_2 = \{2, 3, 4\}$ and $a_3 = \{5, 6\}$. Also, we note that Figure 1.b. shows a preemptive schedule with valid antichains $\{1, 2\}, \{3, 4, 5\}, \{5, 6\}, \{6, 7\}, \{6, 8\}$ and $\{7, 8\}$.

3.2. Column generation

Since set $\mathcal{A}$ may be very large, even when the activity set X is small, $\mathcal{P}_\mathcal{A}$ needs to be handled using *column generation* (see Mingozzi et al. [14] and Brucker and Knust [4]). Column generation is a technique commonly used for solving a linear program (LP) containing an exponential number of variables. It involves first initializing this LP with a small number of *active* variables (which may be obtained by applying some heuristic), iteratively solving the induced *restricted problem* to optimality, and then using the dual variables to generate new improving primal variables. The search for these improving primal variables is called the related *Pricing Problem*. The new variables are added to the restricted problem and the process goes on until no more improving variables can be found. The solution of the restricted problem is then the optimal solution. When this technique is associated with a branch-and-bound process (usually for integer formulation) it gives rise to a *branch-and-price* solution method.

For our needs, let us consider some active antichain subset $\mathcal{B} \subseteq \mathcal{A}$, together with some dual solution λ of the restricted Linear Programming formulation $\mathcal{P}_\mathcal{A}^\mathcal{B}$ defined by (we suppose that $\mathcal{B}$ is such that this program admits a feasible solution):

$$\text{Minimize} \sum_{a \in \mathcal{B}} z_a \tag{4}$$

s.t.

$$\forall i \in X, \sum_{a \in \mathcal{B} | i \in a} z_a = d_i \tag{5}$$

$$\forall a \in \mathcal{B}, z_a \geq 0 \tag{6}$$

If we denote the dual variables corresponding to constraints (5) by λ_i, then solving the related pricing problem $PRICE(\lambda)$ means computing some valid antichain a_0, such that

$$\sum_{i \in a_0} \lambda_i > 1 \tag{7}$$

Though this problem is NP-Complete, it may be efficiently handled through a combination of greedy search and Integer Linear Programming (ILP). A well-fitted ILP formulation of the $PRICE(\lambda)$ problem can be expressed as follows, where we have only one type of decision variable, $y_i \in \{0, 1\}$. This decision variable is defined so that $y_i = 1$ if and only if activity $i \in a_0$.

$$\text{Maximize} \sum_{i \in X} \lambda_i y_i \tag{8}$$

s.t.

$$\forall i, j \in X | i \prec j, y_i + y_j \leq 1 \tag{9}$$

$$\forall k \in K, \sum_{i \in X} r_{ik} y_i \leq R_k \tag{10}$$

$$\forall i \in X, y_i \in \{0, 1\} \tag{11}$$

3.3. Antichain linear program and feasible Schedules

Linear Program $\mathcal{P}_\mathcal{A}$ only provides a lower bound of Preemptive RCPSP instance I. If vector $z = (z_a, a \in \mathcal{A})$ is a feasible solution of $\mathcal{P}_\mathcal{A}$, it may not be possible to turn it into a feasible solution of I whose makespan is $\sum_{a \in \mathcal{A}} z_a$. We can link the valid antichain set $\mathcal{A}$ to an oriented graph structure $(\mathcal{A}, E_\prec)$ by specifying that there exists an arc $(a, b) \in (\mathcal{A}, E_\prec)$ from antichain a to antichain b if there exist activities $i \in a$ and $j \in b$ such that $i \prec j$.

Let z be some feasible solution of $\mathcal{P}_\mathcal{A}$ and $\mathcal{A}(z) \subseteq \mathcal{A}$ be the set $\mathcal{A}(z) = \{a \in \mathcal{A}$ such that $z_a \neq 0\}$ of active antichains according to z. Also, let $G_{\mathcal{A}(z)}$ be the subgraph of $(\mathcal{A}, E_\prec)$ induced by $\mathcal{A}(z)$.

For the instance described in Figure 1.a, Figure 1.d. shows an optimal solution z of the Antichain linear program where $\mathcal{A}(z) = \{a_1, a_2, a_3\}$ with $a_1 = \{1, 7, 8\}, a_2 = \{2, 3, 4\}$ and $a_3 = \{5, 6\}$. Note that since $1 \prec 3, 3 \prec 6$ and $5 \prec 7$, we have $(a_1, a_2), (a_2, a_3), (a_3, a_1) \in (\mathcal{A}, E_\prec)$ and $G_{\mathcal{A}(z)}$ contains a circuit.

The following result can easily be checked.

Theorem 3.1. *Let z be some feasible solution of $\mathcal{P}_\mathcal{A}$. Then there exists a feasible schedule σ such that $z(\sigma) = z$ if and only if the subgraph $G_{\mathcal{A}(z)}$ does not contain any circuit.*

It may be remarked that program $\mathcal{P}_\mathcal{A}$ provides an additional insight into Preemptive RCPSP. Let σ be any feasible Preemptive RCPSP schedule, and let $z(\sigma) = (z(\sigma)_a, a \in \mathcal{A})$ be the related vector associated with σ and $\mathcal{A}$. If $\mathcal{A}(z(\sigma))$ is the related active antichain set, then we see that solving the restricted linear

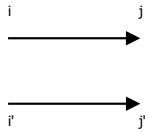


Figure 2: Forbidden structure for interval orders

program ($\mathcal{P}_{\mathcal{A}}^{\mathcal{A}(z(\sigma))}$) using the Primal Simplex Algorithm yields another feasible schedule σ^* with makespan no larger than the makespan of σ. Furthermore, Linear Programming Theory tells us that the number of active antichains related to σ^* (that is to say the cardinality of $\mathcal{A}(z(\sigma^*))$) does not exceed the number of constraints of ($\mathcal{P}_{\mathcal{A}}^{\mathcal{A}(z(\sigma^*))}$), which is equal to the cardinality of the activity set X. Preemptive RCPSP can thus be seen as a combinatorial problem related to the search for some acyclic subgraph $G_{\mathcal{B}}$ of the antichain graph $(\mathcal{A}, E_{\prec})$ such that $Card(\mathcal{B}) \leq Card(X)$ and the optimal value of the program $(\mathcal{P}_{\mathcal{A}}^{\mathcal{B}})$ is minimal.

3.4. Interval Orders

A partially ordered set $(Z, \prec_{io})$ is an *interval order* if the elements i of Z may be represented as closed intervals $[b_i, e_i]$ of the real line, such that, for any pair $i, j \in Z$:

$$i \prec_{io} j \text{ if and only if } e_i < b_j$$

It is known (see [20]), that the partially ordered set $(Z, \prec_{io})$ is an interval order if and only if $(Z, \prec_{io})$ does not contain a suborder isomorphic to the structure described in Figure 2.

If we now consider our Preemptive RCPSP instance $I = (X, K, \prec)$, we see that:

Theorem 3.2. *If the partial order $(X, \prec)$ is an interval order, then the oriented antichain graph $(\mathcal{A}, E_{\prec})$ is acyclic.*

Proof. We suppose the converse, and consider some circuit Γ in $(\mathcal{A}, E_{\prec})$ with minimal length. Then we must distinguish two cases:

First case: $|\Gamma| = 2$, which means that Γ contains two antichains a and b. Here we see that $|a| \geq 2$ and $|b| \geq 2$, and there must exist $i, j' \in a$ and $j, i' \in b$ such that $i \prec j$ and $i' \prec j'$. From this it follows that $(\{i, j, i', j'\}, \{i \prec j, i' \prec j'\})$ defines a forbidden structure for interval orders (Figure 2), which entails a contradiction.

Second case: $|\Gamma| \geq 3$. Since $(X, \prec)$ is acyclic, Γ must contain 3 consecutive antichains a, b, c such that $|\Gamma \cap b| = 2$ and there exist $i \in a$, $j, i' \in b$ and $j' \in c$, such that $i \prec j$ and $i' \prec j'$. But from the minimality of Length(Γ) and from the fact that a, b and c are antichains it can be deduced that i, j, i' and j' must define a forbidden structure for interval orders (Figure 2), which also entails a contradiction. □

This result significantly impacts the design of the algorithm which will be presented in the next section. Clearly, if σ is a feasible schedule for the Preemptive RCPSP instance $I = (X, K, \prec)$, we remark that it is possible to extend the precedence relation $\prec$ into an interval order $\prec_{\sigma}$, so that σ remains consistent with $\prec_{\sigma}$. It is sufficient to define $\prec_{\sigma}$ such that for any activity pair $i, j \in X$:

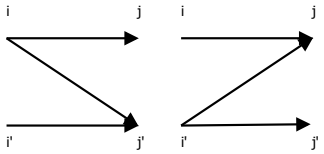


Figure 3: Minimal interval extensions for the forbidden structure for interval orders

$$i \prec_\sigma j \text{ if and only if } EndTime(\sigma, i) \leq StartTime(\sigma, j) \tag{12}$$

Putting this last remark and Theorem 3.2 together shows us that for the Preemptive RCPSP instance I it is sufficient to enumerate the extensions $\prec_{io}$ of the order relation $\prec$, which are interval orders. We can in fact restrict ourselves to those extensions $\prec_{io}$ which are minimal for inclusion, that is to say where there exists no extension $\prec'$ of $\prec$ that is an interval order and where $\prec' \subset \prec_{io}$ and $\prec' \neq \prec_{io}$ (this means that for any activity pair i, j in X: $i \prec' j$ implies $i \prec_{io} j$ and there exist $i', j' \in X$ such that $i' \prec' j'$ without $i' \prec_{io} j'$). Furthermore, it is easy to establish the following straightforward result.

Proposition 3.1. *Let $(X, \prec')$ be an interval order extension of $(X, \prec)$. Therefore, if $(X, \prec)$ contains a forbidden structure of interval orders $(\{i, j, i', j'\}, \{i \prec j, i' \prec j'\})$, then $(X, \prec')$ contains at least the constraints $i \prec' j'$ or $i' \prec' j$.*

The structures $(\{i, j, i', j'\}, \{i \prec'_1 j, i' \prec'_1 j', i \prec'_1 j'\})$ and $(\{i, j, i', j'\}, \{i \prec'_2 j, i' \prec'_2 j', i' \prec'_2 j\})$ are known as *minimal interval extensions* of $(\{i, j, i', j'\}, \{i \prec j, i' \prec j'\})$. These structures are described in Figure 3.

The next section is devoted to detailed description of how the enumeration of interval extensions is performed.

4. Interval Order Enumeration Algorithm

Sections 2 and 3 lead us to reformulate PRCPSP for any instance $I = (X, K, \prec)$ as follows. It is a matter of computing an extension $\prec_{io}$ of the precedence relation $\prec$ that is an interval order and where the optimal solution z_{io} of the related Linear Program $\mathcal{P}_{\mathcal{A}_{io}}$ corresponding to antichains $\mathcal{A}_{io}$ of $\prec_{io}$ is the smallest possible.

Our algorithm IOE based on Interval Order Enumerations is a branch-and-bound algorithm which performs some enumeration of the extensions $\prec_{io}$ of $\prec$. We will now specify the components of the tree search process. First we describe the extensions of Preemptive RCPSP instance $I = (X, K, \prec)$ which define the nodes of the related search tree. Then we describe in some detail how branching, bounding and related filtering are performed. We also explain the constraint propagation scheme and the branching strategy, before presenting a summary of the whole algorithm.

4.1. The nodes of the IOE search tree

A node in the search tree induced by a branch-and-bound algorithm is usually defined by a set of additional constraints imposed on the initial problem. In the case of the Preemptive RCPSP instance $I = (X, K, \prec)$, those constraints are:

- additional precedence constraints with the aim of obtaining an interval order extension of $\prec$. We denote by $(X, \rightarrow)$ the precedence graph containing the precedence relations of $(X, \prec)$ and the additional precedence constraints.

- *anti-precedence* constraints, denoted by $i \nrightarrow j$, meaning that $i \rightarrow j$ is forbidden.

The precedence graph $(X, \rightarrow)$ is managed so that it always remains transitive. If $(X, \rightarrow)$ is not an interval order, then it must contain some forbidden structure for interval orders $(\{i, j, i', j'\}, \{i \rightarrow j, i' \rightarrow j'\})$. In order to remove this forbidden structure, we enrich $(X, \rightarrow)$ with additional constraints. There are two cases (see Proposition 3.1):

- Case 1: $i \rightarrow j'$

- Case 2: $i' \rightarrow j$ and $i \nrightarrow j'$.

We can therefore identify any node v of the search tree using a pair $Add_{\rightarrow}(v)$ and $Add_{\nrightarrow}(v)$ where $Add_{\rightarrow}(v)$ and $Add_{\nrightarrow}(v)$ are respectively the sets of additional precedence constraints and anti-precedence constraints which constrain $\prec_{io}$ as follows:

- $\prec \cup Add_{\rightarrow}(v) \subseteq \prec_{io}$

- $\prec_{io} \cap Add_{\nrightarrow}(v) = \emptyset$.

Clearly, if the current order relation happens to define an interval order $(X, \rightarrow)$, the related node v is a terminal node (a leaf).

The forbidden structure for interval orders allows us to perform a binary branching process with immediate successors v_1 and v_2 by successively considering the two following alternatives:

- First alternative (successor v_1): $Add_{\rightarrow}(v_1) = Add_{\rightarrow}(v) \cup \{i \rightarrow j'\}$ and $Add_{\nrightarrow}(v_1) = Add_{\nrightarrow}(v)$

- Second alternative (successor v_2): $Add_{\rightarrow}(v_2) = Add_{\rightarrow}(v) \cup \{i' \rightarrow j\}$ and $Add_{\nrightarrow}(v_2) = Add_{\nrightarrow}(v) \cup \{i \nrightarrow j'\}$

Our branching scheme splits the set of feasible schedules into two disjoint subsets. Moreover, at each node of the search tree we are able to deduce additional information using *constraint propagation* to reduce computational effort.

4.2. Constraint Propagation

We apply several kinds of inference rules in order to detect inconsistencies or to deduce additional precedence constraints and/or anti-precedence constraints. These rules are based on the earliest and latest start times of the activities in the desired schedule. We also use transitive closure and forbidden structures for interval orders of the current relation $\rightarrow$ at each node.

4.2.1. Transitive closure

The first class of rules concerns transitivity, ensuring that at any time during the process, current relation $\rightarrow$ remains transitive:

$$\forall i, j, l \in X, i \rightarrow j \text{ and } j \rightarrow l \implies i \rightarrow l \tag{13}$$

We use the Floyd-Warshall algorithm for computing the transitive closure for the initialization at the root node of our search tree. This algorithm runs in $O(n^3)$ time. However, at the other nodes of the search tree, we update the transitive closure each time a new constraint $i \rightarrow j$ is found, according to the following equation, leading to $O(n^2)$ time complexity.

$$\forall i', j' \in X, i' \rightarrow i \text{ and } j \rightarrow j' \implies i' \rightarrow j' \tag{14}$$

Besides, any relation $i \rightarrow i$ induces an inconsistency, implying that the current search tree node must be pruned.

4.2.2. The earliest and latest start times

In this section we show how time windows based on the earliest and latest start times can be used to deduce further precedence constraints. Our branch-and-bound algorithm initially computes an upper bound UB at the root node, which is then updated each time an improvement is discovered during the tree search. The *earliest start time* ES_i of an activity i is a lower bound for the starting time of i. Also, using a lower bound for the time period between the end of activity i and the upper bound UB, we define a *latest start time* LS_i for any activity i. Using initializations $ES_0 = 0$ and $LS_{n+1} = UB$, these values are computed for any activity i using recursion:

$$ES_i = \max_{j \in X | j \rightarrow i} (ES_j + d_j) \tag{15}$$

$$LS_i = \min_{j \in X | i \rightarrow j} (LS_j - d_i) \tag{16}$$

Doing this allows us to implement the following classical inference rules, which tend to keep the current precedence relation $\rightarrow$ from inducing the existence of a largest path with length greater than or equal to UB.

$$\forall i, j \in X, ES_i + d_i > LS_j \text{ implies } i \nrightarrow j \tag{17}$$

$$\forall i, j \in X, LS_i + d_i \leq ES_j \text{ implies } i \rightarrow j \tag{18}$$

These equations insert into $Add_{\rightarrow}(v)$ and $Add_{\nrightarrow}(v)$ additional precedence relations at node v which should be satisfied in any schedule with makespan less than UB.

4.2.3. Forbidden structures for interval orders

The last class of rules concerns the forbidden structures for interval orders (see Section 3.4). We wish to prevent the current relation $\rightarrow$ from containing any such structure when new precedence or anti-precedence constraints are introduced. The following proposition is straightforward.

Proposition 4.1. *Let $i, j \in X$ such that $i \rightarrow j$. Then*

- $\forall i', j' \in X, i' \rightarrow j'$ *and* $i' \nrightarrow j \implies i \rightarrow j'$
- $\forall i', j' \in X, i' \rightarrow j'$ *and* $i \nrightarrow j' \implies i' \rightarrow j$
- $\forall i', j' \in X, i \nrightarrow j'$ *and* $i' \nrightarrow j \implies i' \nrightarrow j'$

Also, for any $i, j \in X$ such that $i \nrightarrow j$. Then

- $\forall i', j' \in X, i \rightarrow j'$ *and* $i' \rightarrow j \implies i' \rightarrow j'$
- $\forall i', j' \in X, i' \nrightarrow j'$ *and* $i \rightarrow j' \implies i' \nrightarrow j$
- $\forall i', j' \in X, i' \nrightarrow j'$ *and* $i' \rightarrow j \implies i \nrightarrow j'$

Here the role of constraints $i \nrightarrow j$ is obvious. They are useful for the insertion of additional precedence constraints into the $Add_{\rightarrow}(v)$ set with a significant impact on the antichain set and on the optimal value of the related linear program. Of course, whenever the forbidden structure for interval orders appears, the current node is pruned.

4.3. Lower Bound, Upper Bound and Related Filtering

The lower bound which derives from a current node v defined by a couple $(Add_{\rightarrow}(v), Add_{\nrightarrow}(v))$, is provided by the optimal value of the program $\mathcal{P}_{\mathcal{A}}$, where valid antichains are considered as deriving from $(\rightarrow, \nrightarrow)$. We denote this lower bound as $LB(v)$. It can be obtained using column generation and a heuristically solved pricing problem, or computed using the ILP model (see Section 3.2). Every column which has been generated at some time during the process is kept into memory.

In addition, an initial upper bound UB is obtained as part of a preprocessing step, and this is updated as soon as some feasible solution is computed by the IOE search process.

If the optimal solution z of the linear program $\mathcal{P}_{\mathcal{A}}$ is such that the subgraph $G_{\mathcal{A}(z)}$ does not contain any circuits, we consider that we have reached some terminal node of the search tree. If the related value $\sum_{a \in \mathcal{A}(z)} z_a$ is smaller than the value of the current solution (current upper bound UB), we update this current solution.

4.4. Branching Strategy

In Section 4.1 we described the branching mechanism, which is based on the extraction of a forbidden structure of interval orders. Where there is more than one forbidden structure, we need to specify the strategy used in order to decide which forbidden structure $(\{i, j, i', j'\}, \{i \prec j, i' \prec j'\})$ will define the branching.

We proceed by focusing on the shortest circuits of the subgraph $G_{\mathcal{A}(z)}$ and on the antichains in $\mathcal{A}(z)$ which are the *most involved* in these circuits. We recall that branching needs to be performed only if there exists some circuit in the subgraph $G_{\mathcal{A}(z)}$, where z is the optimal solution of $\mathcal{P}_{\mathcal{A}}$, solved after constraint propagation has been applied. We distinguish two cases:

- *First case*: a circuit of length 2 exists. Then consider any antichains $a, b \in \mathcal{A}(z)$ such that there exist $i, j' \in a$ and $j, i' \in b$ with $i \prec j$ and $i' \prec j'$. This means that $S = (\{i, j, i', j'\}, \{i \prec j, i' \prec j'\})$ is a forbidden structure for interval orders. For each such S involved in a circuit of length 2 we determine the weight w_S as follows:

$$w_S = \sum_{\{a \in \mathcal{A}(z) | i, j' \in a\}} z_a + \sum_{\{a \in \mathcal{A}(z) | j, i' \in a\}} z_a \tag{19}$$

  Branching should use the forbidden structure S such that w_S is maximized. If there are several such structures with the same maximum, one of these is simply chosen at random.

- *Second Case*: there are no circuits of length 2. From the proof of Theorem 3.2 we know that there exist 3 antichains $a, b, c \in \mathcal{A}(z)$ such that $i \in a$, $j, i' \in b$ and $j' \in c$, with $i \prec j$, $i' \prec j'$ and $(\{i, j, i', j'\}, \{i \prec j, i' \prec j'\})$ is a forbidden structure for interval orders. Therefore, there exist at least a forbidden structure S and an antichain $b \in \mathcal{A}(z)$ that contains two activities j and i' from S. For any such structure S, we determine the weight w_S as follows:

$$w_S = \sum_{\{a \in \mathcal{A}(z) | j, i' \in a\}} z_a \tag{20}$$

  Again, branching should use the forbidden structure S such that w_S is maximized. As in the first case, if there are several such structures with the same maximum, we choose one of them randomly.

4.5. The IOE branch-and-bound algorithm

We now present the general outline of our IOE branch-and-bound algorithm IOE based on Interval Order Enumeration. It is implemented using a Breadth First Search strategy where the node list, denoted as L, is ordered by increasing optimal value of the antichain linear program, where antichains are computed with precedence graph $(X, \rightarrow)$. We also recall that each node v of the search tree

is defined by $Add_{\rightarrow}(v)$ and $Add_{\nrightarrow}(v)$. Our IOE algorithm may be summarized as described in Algorithm 1.

To the best of our knowledges, the only exact method in the literature is the basic branch-and-price algorithm proposed by Damay et al. [7], whose aim is to assess the performance of a neighbourhood search algorithm. The exact algorithm is used by the authors for determining optimal solutions for J30. A node v of the search tree consists of a set of forbidden disjunctions, implying that optimal solution z of the antichain linear program does not contain these couples of activities. If G_z is not acyclic, the serach tree algorithm determines a minimal length circuit in order to develop the current node. Assume for example that a circuit ({1, 2, 7}, {3, 4, 8, 9}, {5, 6}) is detected in G_z due to precedence constraints (2, 3), (4, 5) and (6, 1). Therefore, the corresponding forbidden disjunctions added to the three descendant nodes are respectively {1, 2}, {3, 4} and {5, 6}. This branching scheme gives rise to a huge number of nodes and does not deal with symmetries.

Our IOE algorithm is based on a new branching scheme in which the search space is reduced to interval order extensions. Furthermore, each node of our branch and bound tree has no more than two children, and constraint propagation is used extensively to deduce additional constraints.

4.6. An Example

The IOE Algorithm is illustrated using the instance shown in Figure 1. We are seeking a preemptive schedule with makespan $UB \leq 8$. To this end, we initialize our branch and bound by creating a single node v_0 such that $Add_{\rightarrow}(v_0) = \emptyset$ and $Add_{\nrightarrow}(v_0) = \emptyset$ and $L = (v_0)$. From the earliest and latest start times we deduce the additional precedence relations $\{1 \rightarrow 7, 1 \rightarrow 8\}$ and $Add_{\rightarrow}(v_0) = \{1 \rightarrow 7, 1 \rightarrow 8\}$.

Evaluation of node v_0. The optimal solution value of the antichain linear program is $5 + 1/3$ and then $LB(v_0) = 5 + 1/3$ with $(a_1 = \{1, 2\}, z_{a_1} = 2/3), (a_2 = \{2, 3, 4\}, z_{a_2} = 1/3), (a_3 = \{1, 5\}, z_{a_3} = 1/3), (a_4 = \{3, 7, 8\}, z_{a_4} = 1/3), (a_5 = \{4, 7, 8\}, z_{a_5} = 1/3), (a_6 = \{3, 4, 8\}, z_{a_6} = 1/3), (a_7 = \{5, 6\}, z_{a_7} = 2 + 2/3), (a_8 = \{6, 7\}, z_{a_8} = 1/3)$. Since $3 \prec 6$ and $5 \prec 8$, (a_4, a_7, a_4) is a circuit which induces a forbidden structure for interval orders $(\{3, 5, 6, 8\}, \{3 \prec 6, 5 \prec 8\})$. We develop two successors :

- Node v_1: $Add_{\rightarrow}(v_1) = Add_{\rightarrow}(v_0) \cup \{5 \rightarrow 6\}$ and $Add_{\nrightarrow}(v_1) = Add_{\nrightarrow}(v_0)$.
- Node v_2: $Add_{\rightarrow}(v_2) = Add_{\rightarrow}(v_0) \cup \{3 \rightarrow 8\}$ and $Add_{\nrightarrow}(v_2) = Add_{\nrightarrow}(v_0) \cup \{5 \nrightarrow 6\}$.

Evaluation of node v_1. The optimal solution value of the antichain linear program is 7 and then $LB(v_1) = 7$ with $(a_1 = \{1, 5\}, z_{a_1} = 1), (a_2 = \{2, 3, 4\}, z_{a_2} = 1), (a_3 = \{5\}, z_{a_3} = 2), (a_4 = \{6\}, z_{a_4} = 1), (a_5 = \{6, 7\}, z_{a_5} = 1), (a_6 = \{6, 8\}, z_{a_6} = 1)$.

Evaluation of node v_2. The optimal solution value of the antichain linear program is 5.5 and then $LB(v_2) = 5.5$ with $(a_1 = \{1, 5\}, z_{a_1} = 1), (a_2 =$

Algorithm 1: The branch-and-bound algorithm IOE

begin

Initialize at root node the breadth search list and the additional constraint sets: $L = (v_0)$, $Add_{\rightarrow}(v_0) = \emptyset$ and $Add_{\nrightarrow}(v_0) = \emptyset$;

Compute a feasible non-preemptive RCPSP schedule and derive an upper bound UB (see Section 4.3);

Perform Constraint Propagation and extend $Add_{\rightarrow}(v_0)$ and $Add_{\nrightarrow}(v_0)$ (see Section 4.2);

Using column generation, compute an optimal solution $z(v_0)$ of the antichain linear program and the lower bound $LB(v_0)$ (see Section 4.3);

while $L \neq \emptyset$ **do**

let v be the first node in L; Delete v from L;

if $LB(v) < UB$ **then**

if $G_{\mathcal{A}(z(v))}$ *is acyclic* **then**

build a feasible schedule and update the upper bound UB;

else

Compute a forbidden structure for interval orders derived from $z(v)$, $S = (\{i, j, i', j'\}, \{i \prec j, i' \prec j'\})$ according to Section 4.4 and create both related children:

Node v_1: $Add_{\rightarrow}(v_1) = Add_{\rightarrow}(v) \cup \{i \rightarrow j'\}$ and $Add_{\nrightarrow}(v_1) = Add_{\nrightarrow}(v)$

Node v_2: $Add_{\rightarrow}(v_2) = Add_{\rightarrow}(v) \cup \{i' \rightarrow j\}$ and $Add_{\nrightarrow}(v_2) = Add_{\nrightarrow}(v) \cup \{i \nrightarrow j'\}$

foreach u *in* $\{v_1, v_2\}$ **do**

Perform Constraint Propagation and extend $Add_{\rightarrow}(u)$ and $Add_{\nrightarrow}(u)$ (see Section 4.2);

Using column generation, compute an optimal solution $z(u)$ of the antichain linear program and the lower bound $LB(u)$ (see Section 4.3);

Insert node u in L according to its related optimal value;

$\{2,3,4\}, z_{a_2} = 1), (a_3 = \{5,6\}, z_{a_3} = 2), (a_4 = \{6,7\}, z_{a_4} = 1/2), (a_5 = \{6,8\}, z_{a_5} = 1/2), (a_6 = \{7,8\}, z_{a_6} = 1/2)$.

Considering $LB(v_2) < LB(v_1)$, we first develop node v_2. Since $1 \prec 3$ and $2 \prec 5$, (a_1, a_2, a_1) is a circuit which induces a forbidden structure for interval orders $(\{1,2,3,5\}, \{1 \prec 3, 2 \prec 5\})$. We develop two successors :

- Node v_3: $Add_{\rightarrow}(v_3) = Add_{\rightarrow}(v_2) \cup \{1 \rightarrow 5\}$ and $Add_{\nrightarrow}(v_3) = Add_{\nrightarrow}(v_2)$.
- Node v_4: $Add_{\rightarrow}(v_4) = Add_{\rightarrow}(v_2) \cup \{2 \rightarrow 3\}$ and $Add_{\nrightarrow}(v_4) = Add_{\nrightarrow}(v_2) \cup \{1 \nrightarrow 5\}$.

Evaluation of node v_3. The optimal solution value of the antichain linear program is 5.5 and then $LB(v_3) = 5.5$ with $(a_1 = \{1,2\}, z_{a_1} = 1), (a_2 = \{3,4,5\}, z_{a_2} = 1), (a_3 = \{5,6\}, z_{a_3} = 2), (a_4 = \{6,7\}, z_{a_4} = 1/2), (a_5 = \{6,8\}, z_{a_5} = 1/2), (a_6 = \{7,8\}, z_{a_6} = 1/2)$. We obtain a feasible schedule whose makespan is equal to the smallest lower bound. This schedule is then optimal and the search tree is stopped.

5. Numerical results

IOE is coded in C++ on linux CentOS, with an Intel(R) Xeon(R) 2.40GHz processor. ILP formulation of the related pricing problem is handled by CPLEX 12 linear solver, and the global IOE process is embedded into the SCIP framework for branch-and-cut-and-price algorithms SCIP [19]. The SCIP framework consists of a template library which implements via breadth first search generic branch-and-bound schemes involving linear programming together with pricing scheme.

Different benchmarks were used to evaluate the performance of our branch-and-bound algorithm. We used the PSPLib ([13]) that contains four classes J30, J60, J90 and J120 with 30, 60, 90 and 120 activities respectively. Each of the first three classes contains 480 instances, whereas class J120 has 600 instances. We also used the 40 instances (BL) with either 20 or 25 activities proposed by [3]. In addition, we tested our algorithm using the 55 instances (Pack) with a number of activities varying from 17 to 35 proposed by [6]. We give a summary of our computational experiments bellow. Detailed results are available on http://www.isima.fr/~toussain/ [11].

In order to get an initial upper bound we apply to instance I a greedy randomized algorithm designed for the non-preemptive RCPSP (see [17, 1]) and which, in the case of 30 activity PSPLIB instances, approximates the optimal non-preemptive RCPSP optimal value to within 2% in average.

5.1. Results on J30, BL and Pack instances

Our main achievement here is solving Preemptive RCPSP both exactly and rapidly on all J30, BL and Pack instances. The results are presented in Table 1, where for each instance classes (J30, BL and Pack) the mean, minimum, maximum and standard deviation are given for:

Table 1: Results on J30, BL and Pack instances

		Non Preemp. opt.	Preemp. opt.	#nodes	cpu (s)
J30	mean	58.99	58.07	80.06	1.75
	min	34	34	0	< 0.01
	max	129	129	1930	65
	std dev.	14.09	13.80	217.2	5.80
BL	mean	20.90	20.27	34.68	0.08
	min	13	13	2	<0.01
	max	33	32.25	185	0.58
	std dev.	5.30	5.21	46.52	0.1
Pack	mean	?	64.87	3.84	0.02
	min	?	19.71	1	<0.01
	max	?	137.5	46	0.25
	std dev.	?	29.08	7.13	0.04

- *Non Preemp. opt.*: optimal values for non-preemptive RCPSP, when available
- *Preemp. opt.*: optimal values for preemptive RCPSP (our results)
- *#nodes*: number of nodes created (0 means that an optimal value was found by a heuristic in preprocessing and proved to be optimal by the first constraint propagation)
- *cpu (s)*: cpu time in seconds

In the case of J30, we noticed that for 236 instances out of 480, the optimal values for the antichain linear program at the root node, for the preemptive RCPSP and for the non-preemptive RCPSP coincide. Also, the optimal value of the antichain linear program approximates in average the Non-Preemptive RCPSP optimal value to within 6% on average.

5.2. Results on J60, J90 and J120 instances, new lower bounds for non preemptive RCPSP

The computational results for J60, J90 and J120 are presented in Table 2 where we give for each instance class (J60, J90 and J120):

- *#inst*: number of class instances
- *#svd_inst*: number of instances solved
- *Avg_LB*: average lower bounds
- *Avg_UB*: average upper bounds
- *Avg* Δ*(UB/CP)*: average deviation in percent from the critical path lower bound

Table 2: Results on J60, J90 and J120 instances

	j60	j90	j120
#inst.	480	480	600
#svd_inst	383	299	21
Avg_LB	78.26	92.29	114.13
Avg_UB	80.91	100.19	139.35
Avg Delta(UB/LB)	2.94	7.60	21.00
Avg Delta(UB/CP)	12.09	15.61	47.26
Avg cpu (s)	2540.9	4650.4	10449.5
Avg #noeuds	40998.9	53472.2	35082.3

- *Avg Δ(UB/LB)*: average deviation in percent from our best lower bound LB
- *Avg cpu (s)*: average CPU time in seconds
- *Avg #noeuds*: average numbers of visited search tree nodes

The results are given with a time limit of 3 hours. Our IOE algorithm is able to solve to optimality 383 out of 480 instances for J60, 299 out of 480 instances for J90, but only 21 out of 600 instances for J120. Though we were not able to handle all 60/90/120 activity instances of the PSLIB library in an exact fashion, we were nevertheless able to derive new lower bounds for 33 non-preemptive RCPSP instances of the PSPLIB library. The results are summarised in Table 3, where

- *best non-preemp. UB*: best known upper bound for non-preemptive RCPSP (available in PSPLIB website)
- *Preemp. LB*: lower bound for preemptive RCPSP (our method)
- *deduced no preemp. LB*: lower bound for non-preemptive RCPSP which we deduce from *Preemp. LB*
- *Best known LB*: the best known lower bound currently available from the PSPLIB website and updated with recent results of (see Schutt et al. [18]).

6. Conclusion

An effective branch-and-price algorithm for the preemptive RCPSP scheduling problem based on minimal interval order enumeration has been developed incorporating a number of constraint propagation techniques. As well as solving exactly small Preemptive RCPSP instances for three classes (J30, BL and Pack), it is also able to improve the lower bounds for 33 non-preemptive RCPSP instances. We are currently seeking to adapt this method to the non-preemptive RCPSP.

Table 3: New best lower bounds

instance	best non preemp. UB	premptive LB	deduced non preemp. LB	Best known LB
j6013_1.sm	112	106.41	107	105
j6029_2.sm	133	126.2	127	123
j6029_3.sm	121	117.29	118	115
j6029_4.sm	134	129.29	130	126
j6029_5.sm	110	104.04	105	102
j6029_6.sm	154	145.3	146	144
j6029_7.sm	123	116	116	115
j6029_9.sm	112	106.83	107	105
j6045_1.sm	96	91	91	90
j6045_2.sm	144	137.32	138	134
j6045_3.sm	143	137.5	138	133
j6045_4.sm	108	102.49	103	101
j6045_5.sm	106	100.41	101	100
j6045_6.sm	144	136.42	137	132
j6045_7.sm	122	116.04	117	113
j6045_8.sm	129	122.17	123	119
j6045_9.sm	123	118.2	119	114
j6045_10.sm	114	106.48	107	104
j9041_1.sm	142	129.18	130	129
j9045_3.sm	154	144.43	145	144
j9045_6.sm	175	163.26	164	163
j9045_8.sm	160	150.26	151	150
j9045_9.sm	158	145.12	146	145
j12036_4.sm	236	217.35	218	217
j12051_2.sm	221	200.37	201	200
j12051_5.sm	230	205.88	206	205
j12056_1.sm	237	218.17	219	218
j12056_3.sm	241	222.12	223	220
j12056_4.sm	222	206.62	207	205
j12056_5.sm	280	261.8	262	261
j12056_7.sm	283	263.29	264	260
j12056_8.sm	289	268.04	269	265
j12056_9.sm	288	266.34	267	264

Acknowledgements

This work was carried out within the framework of the Labex IMOBS3 and Labex MS2T, which were funded by the French Government, through the program Investments for the future managed by the National Agency for Research.

References

[1] C. Artigues, P. Michelon, and S. Reusser. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2):249 – 267, 2003.

[2] F. Ballestín, V. Valls, and S. Quintanilla. Pre-emption in resource-constrained project scheduling. *European Journal of Operational Research*, 189(3):1136 – 1152, 2008.

[3] P. Baptiste and C. L. Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1/2):119–139, 2000.

[4] P. Brucker and S. Knust. A linear programming and constraint propagation-based lower bound for the {RCPSP}. *European Journal of Operational Research*, 127(2):355 – 362, 2000.

[5] P. Brucker, A. Drexl, R. Mhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3 – 41, 1999.

[6] J. Carlier and E. Néron. On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research*, 149 (2):314 – 324, 2003.

[7] J. Damay, A. Quilliot, and E. Sanlaville. Linear programming based algorithms for preemptive and non-preemptive {RCPSP}. *European Journal of Operational Research*, 182(3):1012 – 1022, 2007.

[8] E. L. Demeulemeester and W. S. Herroelen. An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. *European Journal of Operational Research*, 90(2):334 – 348, 1996.

[9] S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1 – 14, 2010.

[10] W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2): 289 – 306, 2005.

[11] http://www.isima.fr/˜toussain/.

[12] R. Kolisch and R. Padman. An integrated survey of deterministic project scheduling. *Omega*, 29(3):249 – 272, 2001.

[13] R. Kolisch and A. Sprecher. {PSPLIB} - a project scheduling problem library: {OR} software - {ORSEP} operations research software exchange program. *European Journal of Operational Research*, 96(1):205 – 216, 1997.

[14] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Manage. Sci.*, 44(5):714–729, May 1998.

[15] A. Moukrim, A. Quilliot, and H. Toussaint. Branch and price for preemptive resource constrained project scheduling problem based on interval orders in precedence graphs. *6th Workshop on Computational Optimization, September 811, 2013. Kraków, Poland.*

[16] V. V. Peteghem and M. Vanhoucke. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 201(2):409 – 418, 2010.

[17] A. Quilliot and H. Toussaint. Flow polyhedra and resource constrained project scheduling problems. *RAIRO - Operations Research*, 46(4):373–409, 2012.

[18] A. Schutt, T. Feydy, P. Stuckey, and M. Wallace. Explaining the cumulative propagator. *Constraints*, 16(3):250–282, 2011. ISSN 1383-7133.

[19] SCIP. URL `http://scip.zib.de`.