Branch and Price for Preemptive Resource Constrained Project Scheduling Problem Based on Interval Orders in Precedence Graphs

Aziz Moukrim
HEUDYASIC Laboratory
Technological University
COMPIEGNE
Email: aziz.moukrim@hds.utc.fr

Alain Quilliot
LIMOS CNRS UMR 6158
LABEX IMOBS3, Université
Blaise Pascal
Bat. ISIMA, BP 10125
Campus des Cézaux,
63173 Aubière, France
Email: quilliot@isima.fr

Hélène Toussaint
LIMOS CNRS UMR 6158
LABEX IMOBS3, CNRS
Bat. ISIMA, BP 10125
Campus des Cézaux,
63173 Aubière, France
Email: toussain@isima.fr

***Abstract*—This paper describes an efficient exact algorithm to solve Preemptive Resource Constrained Project Scheduling Problem (Preemptive RCPSP). We propose a very original and efficient branch and bound procedure based upon minimal interval order enumeration, which involves column generation as well as constraint propagation and which is implemented with the help of the generic SCIP software. We perform tests on the famous PSPLIB instances which provide very satisfactory results. To the best of our knowledge it is the first algorithm able to solve at optimality all the set of j30 instances of PSPLIB in a preemptive way. Moreover, this algorithm allows us to update several best known lower bounds for the j60, j90 and j120 instances of PSPLIB.**

I. Introduction

THIS paper deals with the *Preemptive Resource Constrained Project Scheduling Problem* (RCPSP: see [1], [2]). RCPSP aims at scheduling a set of activities, submitted to precedence 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 during all the time it is processed, these resources having limited capacities. This problem has been extensively studied in its non preemptive version ([3], [4]), which means that every activity has to be run as a whole, without any kind of interruption. There exist several variants of RCPSP (see [5], [6] for recent surveys). We talk about Preemptive RCPSP when an activity may be run in several steps: one may launch such an activity, interrupt it, keep on with this activity a little further, and so on. There exists few works on Preemptive RCPSP: [7] developed a branch and bound algorithm, [8] proposed a tree search procedure augmented with pruning rules (best-first tree search), [9] proposed an integer linear program which add preemption penalties, [10] and [11] dealt with preemption in an heuristic way and [12] 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. Still, one easily checks that such a hypothesis is very restrictive, and only allows to get an approximation of the optimal value of the problem. In this paper, we consider the problem under 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.

Our approach is a Branch and Bound one which involves constraint propagation, as well as the management of specific rational *Antichain* linear program whose variables are associated with subsets of activities which may be simultaneously processed during the schedule. This LP, which was was first introduced by [13], provides us with a lower bound of both Preemptive and Non Preemptive RCPSP. But dealing with it requires implementing a pricing or column generation scheme. It was proved in [14] that if the input RCPSP instance satisfies some ad hoc properties, then any optimal solution of the *Antichain* linear program may be turned into a feasible optimal schedule, without any increase of the *makespan* value. What we do here is to use this property in order to perform a tree search which may be viewed as being embedded into the enumeration process of all minimal extensions of the precedence relation which define *interval* orders. The resulting process happens to be very efficient, since it is able to solve in an exact way all 30 activity instances of the PSPLIB library, and to improve best existing lower bounds for several 60/120 activity instances of this library.

So the paper is organized as follows: we first recall what is Preemptive RCPSP (Section II), and next introduce the theoretical tools related to the *Antichain* LP and to interval orders (Section III), which will provide us with the basis of our algorithmic approach. Section IV describes the algorithm INT-ORD-ENUM and its implementation, and Section V is devoted to a presentation of experimental results.

II. Preemptive RCPSP

An instance $I = (X, K, \ll)$ 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 of activity i for resource k; those resources are given back to the system once the activity is over

- $\forall (i,j) \in X^2$, $i \ll j$ means that *i precedes j*: activity *j* cannot start before *i* is over (*Precedence* constraints)

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;
- 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, $Act(\sigma, t) = \{i$ such that $t_i < t < t_i + d_i\}$ denoting the set of the activities currently run at time t according to schedule σ.

So, solving *Non Preemptive* RCPSP means computing σ with a minimal *makespan* (total duration of the process).

In case 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 sense of standard RCPSP. Since there does not exist any "a priori" restriction either on the number of sub-activities or on their durations, which may be arbitrarily small, the existence of an optimal solution of Preemptive RCPSP has to be discussed.

III. Fundamental tools

A. The Antichain Linear Program: A Lower Bound

Let $I = (X, K, \ll)$ be some Preemptive RCPSP instance, defined according to notations of Section II. We suppose (we clearly may do it) that precedence relation $\ll$ is transitive. Then we define an *antichain* as being any subset a of X such that there does not exist $(i,j) \in a^2$ such that $i \ll j$. We say that such an antichain is *valid* if: $\forall k \in K, \sum_{i\in a} r_{ik} \le R_k$.

It comes that a subset $a \subseteq X$ of activities is a *valid antichain* iff activities in a may be simultaneously run inside some feasible schedule. We denote by A the set of all *valid antichains*.

Then we become able to set the following linear program, which we call *Antichain Linear Program* associated with Preemptive RCPSP instance $I = (X, K, \ll)$, which was already introduced in [13, 14], and which we denote by $(\mathsf{P})_{\mathsf{Ant}}$:

$$(\mathsf{P})_{\mathsf{Ant}} \quad \begin{array}{l} \text{Minimize } \sum_{a\in \mathsf{A}} z_a \\ \text{Subject to} \\ \quad \forall i \in X, \quad \sum_{a\in \mathsf{A} \mid i \in a} z_a = d_i \qquad \text{(C1)} \\ \quad \forall a \in \mathsf{A}, \quad z_a \ge 0 \end{array}$$

Explanation: if σ is any feasible schedule related to instance I, we may associate with σ and with any valid antichain a, the total amount of time $z(\sigma)_a$ during which the activities which are simultaneously run according to σ are exactly the activities of a. Then we see that $z(\sigma) = (z(\sigma)_a, a\in \mathsf{A})$ is a feasible solution of $(\mathsf{P})_{\mathsf{Ant}}$ since constraints (C1) express the fact that any activity i has to be completely done, or, equivalently, that the duration of all antichains containing i must be equal to the duration of i. It comes that the optimal value of $(\mathsf{P})_{\mathsf{Ant}}$ provides us with a lower bound of the optimal value of I, which we denote by $LB(I)$.

B. Dealing with $(P)_{Ant}$: Column generation

Since the set A may be very large, even when the activity set X is small, we need to handle the *Antichai*n LP $(\mathsf{P})_{\mathsf{Ant}}$ through *column generation*. Column generation is an usual technique to solve a LP which contains an exponential number of variables. It consists in initializing this LP with a few number of *active* variables (which may be obtained from application of some heuristic), and then in iteratively solving the induced *restricted problem* at optimality and using the dual variables to generate a new improving primal variable. The search for this improving primal variable is called the related *Pricing* Problem. The new variable is added to the restricted problem and the process goes on until no improving variable can be found: then the solution of the restricted problem is the optimal solution. When this technique is associated to a Branch and Bound process (usually for integer formulation) it gives rise to a *Branch and Price* process. In our case, let us consider some active antichain subset $B \subseteq \mathsf{A}$, together with some dual solution λ of the restricted LP $(\mathsf{P})^B_{\mathsf{Ant}}$ defined by (we suppose that B is such that this program admits a feasible solution):

$$(\mathsf{P})^B_{\mathsf{Ant}} \quad \begin{array}{l} \text{Minimize } \sum_{a\in B} z_a \\ \text{Subject to} \\ \quad \forall i \in X, \quad \sum_{a\in B \mid i \in a} z_a = d_i \qquad \text{(C1)} \\ \quad \forall a \in B, \quad z_a \ge 0, \end{array}$$

Then solving the related pricing problem PRICE(λ) means computing some valid antichain a, such that:

$$\sum_{i \in a} \lambda_i > 1.$$

Though this problem is NP-Complete, it may be efficiently handled through a combination of greedy search and Integer Linear Programming (LIP). A well-fitted LIP formulation of the PRICE(λ) problem comes as follows:

Pricing LIP Formulation L-PRICE(λ)

$$\begin{array}{l} \text{Maximize } \sum_{i\in X} \lambda_i y_i \\ \text{Subject to} \\ \quad \forall (i,j) \in X^2 \mid i \ll j, \quad y_i + y_j \le 1 \qquad \text{(C2)} \\ \quad \forall k \in K, \ \sum_{i\in X} r_{ik} y_i \le R_k \qquad \text{(C3)} \\ \quad \forall i \in X, \ y_i \in \{0,1\} \end{array}$$

C. Turning a Solution of $(P)_{Ant}$ into a Feasible Schedule ?

Unfortunately, Linear Program $(\mathsf{P})_{\mathsf{Ant}}$ only provides us with a lower bound of Preemptive RCPSP instance I: if vector $z = (z_a, a\in \mathsf{A})$ is a feasible solution of $(\mathsf{P})_{\mathsf{Ant}}$, it may not be possible to turn it into a feasible solution of I. As a matter of fact, we may provide the valid antichain set A with an oriented graph structure $(\mathsf{A}, E_{<<})$ by setting that there exists an arc $(a, b) \in E_{<<}$ from antichain a to antichain b, if there exist activities $i \in a$ and $j \in b$, such that $i \ll j$.

Then we easily check that:

Theorem 1: *Let z be some feasible solution of $(\mathsf{P})_{\mathsf{Ant}}$, and $\mathsf{A}(z) \subseteq \mathsf{A}$ be the set $\mathsf{A}(z) = \{a \in \mathsf{A}$ such that $z_a \neq 0\}$ of active* *antichains according to z. Then there exists a feasible schedule σ such that $z = z(\sigma)$ if and only if the subgraph ($A(z)$, $E_{<<}$) does not contain any circuit.*

Proof: Left to the reader.

Still, we may notice that program $(P)_{Ant}$ provides us with additional understanding of Preemptive RCPSP: if σ is any feasible Preemptive RCPSP schedule, if $z(\sigma) = (z(\sigma)_a, a \in A)$ is the related solution of $(P)_{Ant}$, and if $A(z(\sigma))$ is the related active antichain set, then one sees that solving the restricted linear program $(P)^{A(z(\sigma))}_{Ant}$ through Primal Simplex Algorithm provides us with another feasible schedule σ^* with makespan no larger than the makespan of σ. Moreover, Linear Programming Theory tells us that the number of active antichains related to σ^*, that means the cardinality of $A(z(\sigma^*))$ does not exceed the number of constraints of $(P)^{A(z(\sigma))}_{Ant}$, which is equal to the cardinality of the activity set X. This makes appear Preemptive RCPSP as a combinatorial problem related to the search of some acyclic subgraph (B, $E_{<<}$) of the antichain graph (A, $E_{<<}$), such that:

- Card(B) $\leq$ Card(X);
- The optimal value of the program $(P)^{B}_{Ant}$ is minimal.

This confirms the existence of an optimal solution.

Also, we may notice that no activity which is not in the set $Min(X)$ of the activities which are minimal in the sense of the precedence relation $\ll$, may start before the time when at least one activity in $Min(X)$ is completed. We deduce that the lower bound which derives from the resolution of the $(P)^{A}_{Ant}$ program may be improved by adding the following constraint to $(P)^{A}_{Ant}$:

$\sum_{a \in A\text{-}Min} z_a \geq \text{Inf}(d_i, i \in Min(X))$, with $A\text{-}Min = \{a \in A$, such that $a \subseteq Min(X)\}$.

We denote by $LB^*(I)$ this improved lower bound.

D. Interval Orders

A partially ordered set (Z, $<$) is an *interval order* if the elements z of Z may be represented as closed intervals $[o(z), d(z)]$ of the real line, in such way that, for any pair z, z' in Z:

- $z < z'$ if and only if $d(z) < o(z')$.

It is known (see [20]), that the partially ordered set (Z, $<$) is an interval order if and only if there does not exist $x, y, z, t \in$ Z such that:

- $x < y$ and $t < z$; (C4)
- there does not exist any other pair $u < v$ with $u, v \in \{x, y, z, t\}$ than the pairs in (C4) above.

Figure 1 below shows the forbidden pattern associated with interval orders:

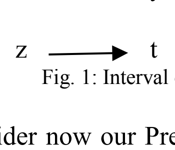


Fig. 1: Interval order forbidden pattern

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

Theorem 2: *If the partial order (X, $\ll$) is an interval order, then the oriented antichain graph (A, $E_{<<}$) is acyclic (does not contain any circuit).*

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

- *first case*: Length(Γ) = 2, which means that Γ contains two antichains a and b. Then we see that there must exist $i_1, j_2 \in a$, $i_2, j_1 \in b$ such that: $i_1 \ll j_1$ and $i_2 \ll j_2$. Then it becomes easy to check that i_1, j_1, i_2, j_2 define a forbidden pattern in the above sense, which induces a contradiction.
- *second case*: Length(Γ) $\geq$ 3, which means that Γ contains 3 consecutive antichains a, b, c, and that there must exist $x \in a$, y, $z \in b$, $t \in c$, such that $x \ll y$ and $z \ll t$. But we also deduce from the minimality of Length(Γ) and from the fact that a, b, c are antichains that x, y, z, t must define a forbidden pattern in the above sense, which induces again a contradiction. **End-Proof**.

This result will impact in a very significant way 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 $I = (X, K, \ll)$, it is possible to extend the precedence relation $\ll$ into an interval order $\ll_\sigma$, in such a way σ remains consistent with $\ll_\sigma$. In order to do it, we only need to set, for any activity pair i, j in X:

- $i \ll^* j$ iff End-Time(i) $\leq$ Start-Time(j).

Putting this last remark and Theorem 2 together makes appear that we only need, in order to deal with the Preemptive RCPSP instance I, to enumerate the extensions $\ll^*$ of the order relation $\ll$ which are interval orders. As a matter of fact, we may restrict ourselves to those extensions $\ll^*$ which are minimal for inclusion, that means which are such that there does not exist any extension $\ll'$ of $\ll$ which is an interval order and which is such that: $\ll' \subset \ll^*$, $\ll' \neq \ll^*$. So, next section is devoted to an accurate description of the way this enumeration process is performed.

IV. The branch/bound algorithm INT-ORD-ENUM

A. A Reformulation of Preemptive RCPSP Instance I

Sections II and III lead us to reformulate Preemptive RCPSP instance $I = (X, K, \ll)$ as follows:

Preemptive RCPSP Reformulation: Compute an extension $\ll^*$ of the precedence relation $\ll$ which is an interval order and which is such that, if z^* is an optimal solution of the related LP $(P)_{Ant}$, obtained through Primal Simplex Algorithm and column generation, the optimal value $1.z^*$ is the smallest possible.

Remark: Clearly, program $(P)_{Ant}$ must be understood here with respect to $\ll^*$and to the related *Antichain* set $A^* \subseteq A$.

So, our algorithm INT-ORD-ENUM is a Branch/Bound algorithm, which performs some enumeration of the extensions $\ll^*$ of $\ll$. We must now specify the main components of such a tree search process, which are about:

- the extensions of Preemptive RCPSP instance $I = (X, K, \ll)$ which define the nodes of the related search tree;
- the way branching is performed;
- the way bounding and related filtering are performed;
- the way constraint propagation is performed;
- the branching strategy;
- the way the whole algorithm is implemented.

B. The Nodes of the INT-ORD-ENUM Search Tree

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

- additional precedence constraints $i \ll j$;
- *anti-precedence* constraints i **--->** j: i **--->** j means that $i \ll j$ is forbidden.

So, we may identify any node of the search tree with a pair $(Add_{\ll}, Add_{--->})$, where $Add_{\ll}$ and $Add_{--->}$ are respectively the sets of additional precedence constraints and anti-precedence constraints which constrain $\ll^*$ as follows:

- $(\ll \cup Add_{\ll}) \subseteq \ll^*$;
- $(Add_{--->} \cap \ll^*) = \text{Nil}$.

Explanation of the *anti-precedence* constraints: it must be understood that those constraints have sense only with respect to the reformulation of subsection IV.*A*. That means that they are not going to play any role either with respect to an eventual feasible schedule or with respect to the program $(P)_{Ant}$, but that they only will impact the way additional precedence constraints may be added to the initial ones.

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

C. The Branching Mechanism

If current precedence relation, which is managed in such a way it always remains transitive, is not an interval order, then it must contain some forbidden pattern i_1, j_1, i_2, j_2:

- $i_1 \ll j_1$ and $i_2 \ll j_2$; (C5)
- no other pair $u \ll v$ exists with $u, v \in \{i_1, i_2, j_2, j_1\}$ than the pairs in (C5) above.

This forbidden pattern allows us to perform a binary branching process by successively considering the 2 following alternatives:

- 1 th alternative (1 th son): insert $i_1 \ll j_2$ into $Add_{\ll}$;
- 2 th alternative (2 th son): insert $i_2 \ll j_1$ into $Add_{\ll}$ and insert i_1 **--->** j_2 into $Add_{--->}$;

D. Lower Bound, Upper Bound and Related Filtering

Lower Bound: The lower bound which derives from a current node defined by a pair $(Add_{\ll}, Add_{--->})$, is provided by the optimal value of the program $(P)_{Ant}$, where valid antichains are considered as deriving from $(\ll \cup Add_{\ll})$. This problem is handled through column generation, as explained in Section III.*C*, and the Pricing problem PRICE(λ) is handled while using the ILP model of Section III.*C*. Every column which has been generated at some time during the process is kept into memory.

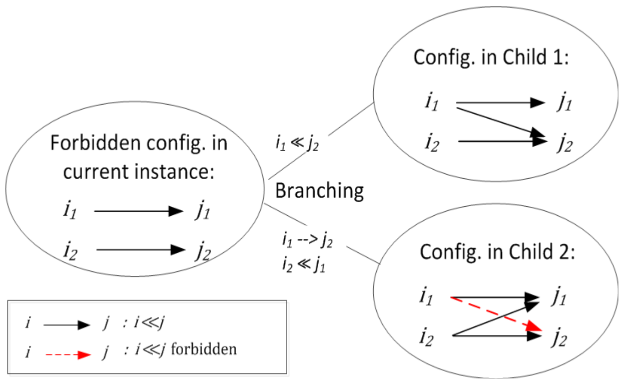


Fig. 2: the Branching Mechanism

Upper Bound: Also, we make in such a way that we are provided, as part of a pretreatment, with an initial upper bound *UB*: in order to get this initial upper bound, we apply to instance *I*, a greedy randomized algorithm designed for the Non Preemptive RCPSP (see [19]) and which, in case of 30 activity PSPLIB instances, approximates the optimal Non Preemptive RCPSP optimal value by less than 2% in average. Of course, *UB* is updated as soon as some feasible solution is computed by the INT-ORD-ENUM search process.

Related Filtering: Of course, if the optimal solution z^* of LP $(P)_{Ant}$, is such that the subgraph $(A(z^*), E_{\ll})$ does not admit any circuit, we consider that we have been reaching some terminal node of the search tree. In case related value $1.z^*$ is smaller than the value of the current solution (current upper bound *UB* of the forthcoming section IV.*E*, we update this current solution as a feasible schedule σ such that $z^* = z^*(\sigma)$. We notice that it is sufficient to consider the subgraph $(A(z^*), E_{\ll})$ as defined with respect to initial precedence relation $\ll$, since our goal is to make possible turning a solution of $(P)_{Ant}$, into a feasible schedule in the sense of initial precedence relation $\ll$.

E. Constraint Propagation

We apply several kind of inference rules $\alpha \models \beta$, whose semantics come as follows:

- α (precondition part) denotes constraints which are already associated with current node *S* of the three search;
- β (consequent part) denotes the additional relations which have to inserted into sets $Add_{\ll}$ and $Add_{--->}$..

The first class of rules deals with transitivity, and makes in such a way that, at any time during the process, current relation $\ll^* = (\ll \cup Add_{\ll})$ remains transitive:

Rule 1: $i \ll^* j, z \ll^* i \models z \ll^* j$;
Rule 1': $i \ll^* j, j \ll^* z \models i \ll^* z$;

Of course, any relation $i \ll^* i$ induces a *Failure* signal.

The second one deals in a classical way with largest paths and current upper bound *UB*. We add two dummy activities: *s* (source) and *p* (sink) defined as usual and, at every time during the process, we are provided, for every activity i, with:

- $\pi(i)$ = earliest finish time for i, which means the length of a largest path from s to i;
- $\Pi(i)$ = the length of the largest path between the beginning of i and p: $\Pi(i)$ = UB - LS(i) where LS(i) is the latest starting time f*or i*

Doing this allows us to implement the following classical inference rules, which tend to keep the current precedence relation ($\ll \cup Add_{<<}$) from inducing the existence of a largest path with length $\geq UB$:

Rule 2: $\pi(i) = \alpha,\ i \ll * y$ *and* $\alpha + d_y > \pi(y) \models \pi(y) = \alpha + d_y$;

Rule 2': $\Pi(i) = \alpha, y \lll^* i$ *and* $\alpha + d_y > \Pi(y) \models \Pi(y) = \alpha + d_y$;

Rules 2 and 2' update values $\pi(y)$ and $\Pi(y)$, $y \in X$, as soon as necessary. Of course, the existence of any path with length $\geq UB$ induces a Failure signal.

Rule 3: $\pi(i) = \alpha,\ \alpha + \Pi(y) > \text{UB} \models i$ **--->** y ;

Rule 3': $\Pi(i) = \alpha, \alpha + \pi(y) > \text{UB} \models y$ **--->** i;

Rules 3 and 3' forbid any additional precedence relation which would induce the existence of a largest path with length $\geq UB$ to be inserted into $Add_{<<}$.

Rule 4: $\pi(i) = \alpha$, $UB - \Pi(y) + d_y \leq \alpha - d_i \models y \ll * i$;

Rule 4': $\Pi(i) = \alpha$, $UB - \alpha + d_i \leq \pi(y) - d_y \models i \lll^* y$;

Rules 4 and 4' insert into $Add_{<<}$. additional precedence relations which should be satisfied in any schedule with makespan no more than UB.

The last class of rules deals with the forbidden patterns of Section III.*D*, and aims at keeping current relation ($\ll \cup Add_{<<}$) from containing any such a pattern:

Rule 5: $i \lll^* j, z \lll^* t$ *and* z **--->** $j \models i \lll^* t$;

Rule 5': $i \lll^* j, t \lll^* z$ *and* i **--->** $z \models t \lll^* j$;

Rule 6: i **--->** $j, z \lll^* j$ *and* $i \ll * t \models z \lll^* t$;

Rule 6': i **--->** $j, i \lll^* z$ *and* t **--->** $z \models t$ **--->** j.

We see here the true role of constraints i **--->** j, which help us in inserting additional precedence constraints into the $Add_{<<}$ set, with a strong impact on the antichain set A* and on the optimal value of the related linear program $(P)_{Ant}$,. Of course, any time such a pattern appears, it induces a Failure signal.

F. Branching Strategy

We described in IV.*B* the Branching mechanism, which relies on the extraction of some forbidden pattern i_1, j_1, i_2, j_2:

- $i_1 \ll j_1$ and $i_2 \ll j_2$; (C5)
- no other pair $u \ll v$ exists with $u, v \in \{ i_1, i_2, j_2, j_1\}$ than the pairs in (C5) above.

Since it is known that the way branching parameters are chosen is a critical issue as soon as Branch/bound and constraint propagation are performed. So we must now specify the strategy which is used here in order to compute a well-fitted 4-uple i_1, j_1, i_2, j_2.

As a matter of fact, we apply here the well-known "most constraint variable" principle, and focus on the shortest circuits of the subgraph (A(z*), $E_{<<}$) and on the antichains in A(z*) which are the most involved in those circuits.

As told in Section IV.*D*, branching has to be performed only if there exists some circuit in the subgraph (A(z*), $E_{<<}$), where z* is the optimal solution of the LP $(P)_{Ant}$, solved after constraint propagation has been performed. Then we distinguish two cases:

- **First case**: there exists a circuit with length 2. In such a case, circuits with length 2 define in a natural way a non oriented graph (A(z*), F) on the set A(z*): two antichains a, a' in A(z*) define an edge of this graph if they also define a circuit of the oriented graph (A(z*), $E_{<<}$), We consider an antichain a_0 which is with maximal degree $D_F(a_0)$ in the graph (A(z*), F), together with some antichain a_1, with maximal degree $D_F(a_1)$ among the F-neighbours of a_0. Then we derive the forbidden pattern i_1, j_1, i_2, j_2, according to the proof of Theorem 2 in Section III.*D* and to Figure 3 (a).
- **Second Case**: there does not exist any circuit with length 2. Then we compute the largest strongly connected component A_0 of the oriented graph (A(z*), $E_{<<}$), together with the antichain a_0, which is such that:
  - There exists at least one pair a_1, a_2, such that (a_1, a_0) and (a_0, a_2) are in the arc set $E_{<<}$, while (a_1, a_2) $\notin E_{<<}$;
  - The sum $D^-_F(a_0) + D^+_F(a_0)$ of the inner and outer degrees of a_0 in the subgraph (A_0, $E_{<<}$) induced from by A_0 is maximal.

Finally we compute some circuit Γ which contains a_0 as well as a_1, a_2 above and which is with minimal length, and we derive the forbidden pattern i_1, j_1, i_2, j_2, according to the proof of Theorem 2 in Section III.*D* and to Figure 3 (b).

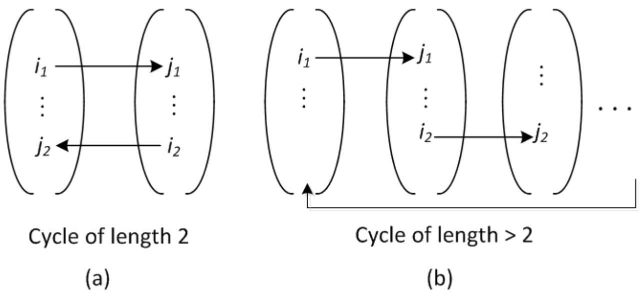


Fig. 3: Extracting a forbidden pattern

G. Implementation

The global Branch/bound algorithm INT-ORD-ENUM Branch/Bound is implemented according to a Breadth First Search strategy which may be summarized as follows:

INT-ORD-ENUM *Algorithmic Scheme*.

1. **Pretreatment**: Compute a feasible Non Preemptive RCPSP schedule σ, while using a greedy randomized insertion flow heuristic as in [19]. Derive an upper bound UB, together with an initial antichain subset $B \subset A$, such that the linear program $(P)^B_{Ant}$ admits a feasible solution; Initialize the breadth search node list L as the list {($Add_{<<}$ = Nil, $Add_{--->}$ = Nil)};

2. **Main Process: Breadth First Tree Search**.
   Let L be the current node list, ordered according to LP $(P)^{B}_{Ant}$ related values, and S be the first node in L; S is defined by two additional constraint sets $Add_{<<}$ and $Add_{--->}$; Delete S from L; Perform Constraint Propagation and extend $Add_{<<}$ and $Add_{--->}$; If *Failure* then go back to 2. Else go to 3.;
3. Solve the LP $(P)^{B}_{Ant}$ related to S through column generation and test the oriented graph $(A(z^*), E_{<<})$ deriving from the obtained optimal solution z^*; If $1.z^* \geq$ UB then go to 2. Else go to 4.;
4. If the graph $(A(z^*), E_{<<})$ is acyclic then derive from z^* a feasible schedule σ, update the upper global bound UB and go back to 2. Else go to 5.;
5. Compute branching parameters i_1, j_1, i_2, j_2, according to Section IV.*F* and create both related children:
   - 1 th son: insert $i_1 << j_2$ into the set $Add_{<<}$;
   - 2 th son: insert $i_2 << j_1$ into the set $Add_{<<}$; and i_1 **--->** j_2 into the set $Add_{--->}$;

   Insert those two children nodes in L, according to their related LP $(P)^{B}_{Ant}$ value; Go back to 2.;

Process ends as soon as the LP value related to the first element of S is no smaller than UB. Then current value UB provides the optimal makespan value;

This algorithm is implemented in C++, and linear programs $(P)^{B}_{Ant}$ and *L-PRICE(λ)* are handled by CPLEX.12 linear solver. But the global INT-ORD-ENUM process is embedded into the SCIP framework for branch cut and price algorithms [16]. The SCIP framework consists in a template library which implements through breadth first search generic branch and bound schemes involving Linear Programming together with pricing scheme. In the present case, what we mainly had to do was providing the C++ procedures which performed, for every node S, the construction of the $(P)^{B}_{Ant}$ and *L-PRICE(λ)* programs, the constraint propagation process and the branching strategy, and assembly them inside SCIP.

H. An Example

Let us consider an instance of 6 activities and 1 resource. Each activity has a duration equal to 1 and a resource requirement equal to 1. The precedence constraints are given by the following precedence graph:

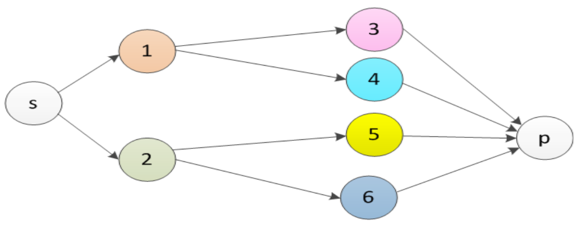
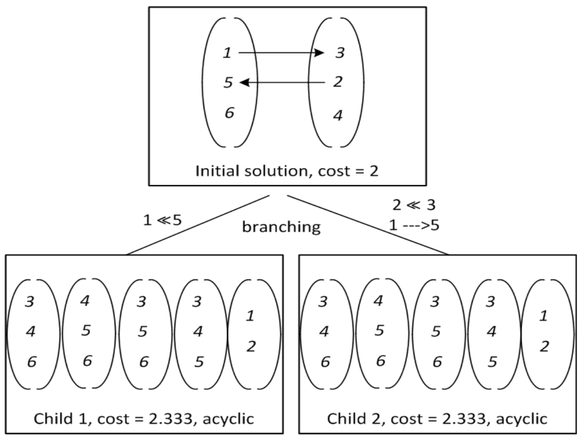


Fig. 4: Precedence graph

We initialize the set of antichains with the 6 singleton antichains. The tree constructed by our method and the branching decisions are given as below:


1
5
6
3
2
4
Initial solution, cost = 2
1 ≪5
branching
2 ≪ 3
1 --->5
Child 1, cost = 2.333, acyclic
Child 2, cost = 2.333, acyclic
Fig. 5: Solving instance of figure 4

The resulting optimal solution is given according to the following Gantt chart.

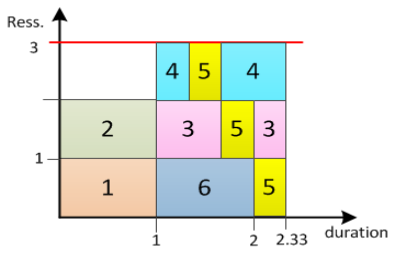


Fig. 6: Gantt chart of optimal preemptive solution

V. Numerical Experiments

Experiments were carried on in C++, on linux CentOS proc. Intel(R) Xeon(R) 2.40GHz. The instances which we used were PSPLIB instances ([17]).

Our main achievement here was to solve in an exact way and in a rather short time (never more that 95 CPU seconds) Preemptive RCPSP on all 30 activity instances of the PSPLIB library, which had been, until now, never done. This first experiment is described in coming section V.*A*.

Also, we could get an evaluation of the bounding process related to linear program $(P)^{B}_{Ant}$, and check that in average, $LB(I)$ approximates the optimal Non Preemptive RCPSP optimal value by less than 6%. By the same way, we checked that the augmented lower bound $LB^*(I)$ hardly improve $LB(I)$ by less than 0.5%.

Finally, though we were not able to handle in an exact way all 60/120 activity instances of the PSLIB library, we could derive from the instances which we were able to handle new lower bounds for several Non Preemptive RCPSP instances of the PSPLIB library. This second part of the experiment will be described in Section V.*B*.

A. Exact results on j30

The columns of table I have the following meaning:

- *No Preemp. opt.*: optimal value for non preemptive RCPSP (available in PSPLIB website)
- *Preemp. opt.*: optimal for preemptive RCPSP (our results)
- *#nodes*: number of nodes created (0 means optimal value was found by heuristic in preprocessing and prooved to be optimal by the first constraint propagation)
- *cpu* (*s*): cpu time in seconds

TABLE I
RESULTS ON J30 INSTANCES OF PSPLIB

	No Preemp. opt.	*Preemp. opt.*	*#nodes*	*cpu*(s)
Mean	58.99	58.07	72.73	2.04
Min	34.00	34.00	0.00	< 0.1
Max	129	129	2130	94.11
std dev.	14.09	13.80	214.87	8.03

B. Comparative Analysis of LB(I), MB(I)*

The following table II provides us with average values for the 480 instances of PSPLIB with 30 activities:

TABLE II: EVALUATION OF THE BOUND *LB*(*I*)

Mean LB(*I*)	*Mean LB**(*I*)	*Mean Premp. opt.*	*Mean No Premp. opt.*
56.73	56.79	58.07	58.99

Remark: in almost 50% of the cases (exactly 236 instances among 480), the values *LB*(*I*), *Premp. Op*. and *No Premp. Opt*. coincide.

C. New best lower bounds

Our method gives new best lower bound for j60, j90 and j120 instances (in a limit of time of 3 hours).

The columns of table III have the following meaning:

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

TABLE III
NEW BEST LOWER BOUNDS

instance	best no preemp. UB	Preemp. LB	deduced no preemp. LB	Best known LB
j6013_1.sm	112	106.41	107	105
j6029_2.sm	133	126.20	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	145.30	146	144
j6029_7.sm	123	116.00	116	115
j6029_9.sm	112	106.83	107	105
j6045_1.sm	96	91.00	91	90
j6045_2.sm	144	137.32	138	134
j6045_3.sm	143	137.50	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	118.20	119	114
j6045_10.sm	114	106.48	107	104
j9045_6.sm	175	163.26	164	163
j12036_4.sm	236	217.35	218	217
j12056_3.sm	241	222.12	223	220
j12056_4.sm	222	206.62	207	205
j12056_5.sm	280	261.80	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

VI. CONCLUSION

Our method is very efficient. Besides exactly solving small size Preemptive RCPSP instances, it is also able to provide us with very good lower bounds for larger scale Non Preemptive RCPSP. We are looking for adapting this method to the non preemptive RCPSP.

REFERENCES

[1] R. Kolisch, R. Padman. Deterministic project scheduling, *Omega*, 48, pp. 249-272 (1999)

[2] P. Brucker, A. Drexl, R. Mohring, K. Neumann, E. Pesch. Resource-constrained project scheduling: notation, classification, models and methods, *EJOR* 112, pp. 3-41 (1999)

[3] W. Herroelen. Project Scheduling-Th./Pract., *Prod./Op. Management*, 14, 4, pp. 413-432 (2006)

[4] S.S. Liu, C.J. Wang. RCPSP profit max with cash flow, *Aut. Const.* 17, pp. 966-74 (2008)

[5] S. Hartmann, D. Briskorn. A survey of variants of RCPSP. *EJOR* 207, pp.1-14 (2010)

[6] M.J. Orji, S. Wei. Project Scheduling Under Resource Constraints: A Recent Survey. *International Journal of Engineering Research & Technology (IJERT)* Vol. 2 Issue 2, (2013)

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

[8] S. Verma. Exact methods for the preemptive resource-constrained project scheduling problem, research and publication, *Indian institute of management* 2006, ahmedabad, india,w.p.no. 2006-03-08.

[9] B.A. Nadjafi, S. Shadrokh. The preemptive resource-constrained project scheduling problem subject to due dates and preemption penalties: an integer programming approach, *Journal of Industrial Engineering*, 1 pp. 35-39 (2008)

[10] F. Ballestin, V. Valls, S. Quintanilla, Preemption in resource- constrained project scheduling, *European Journal of Operational Research*, 189 pp.1136-1152 (2008)

[11] M. Vanhoucke, D. Debels, The impact of various activity assumptions on the lead time and resource utilization of resource- constrained projects, *Computers and Industrial Engineering*, 54 pp.140-154J (2008)

[12] M. Vanhoucke, A genetic algorithm for the net present value maximization for resource constrained projects, EVOComp; *LNCS* 5482 pp. 13-24 (2009)

[13] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the resource-constrained project scheduling based on a new mathematical formulation. *Management Science*, 44 pp. 714–729, (1998).

[14] A. Damay, A. Quilliot, E. Sanlaville. Linear programming based algorithms for preemptive and non preemptive RCPSP, *EJOR*, 182, 3, pp. 1012-1022 (2007)

[15] A. Mehrotra, and M. A. Trick. A column generation approach for exact graph coloring, INFORMS Journal on Computing, 8:4, pp. 133-151 (1996)

[16] http://scip.zib.de/

[17] http://www.om-db.wi.tum.de/psplib/

[18] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey Explaining Time-Table-Edge-Finding Propagation for the Cumulative Resource Constraint. *Lecture Notes in Computer Science* Volume 7874, pp. 234-250 (2013) (last results on http://ww2.cs.mu.oz.au/~pjs/rcpsp/)

[19] A.Quilliot, H.Toussaint: Flow Polyedra and Resource Constrained Scheduling Problem, *RAIRO-RO*, 46-04, p 379-409, (2012)

[20] F.S.Roberts: *Discrete Maths Models*; Prentice Hall, Englewood Cliffs, N.Y, (1976).