A Constraint Programming Approach for the Resource-Constrained Project Scheduling Problem

Olivier Liess(*) • Philippe Michelon(**)

(*) Université d'Avignon et des Pays de Vaucluse, Laboratoire d'Informatique d'Avignon, 339 chemin des Meinajariès, Agroparc BP 1228, 84911, Avignon Cedex 9, France.

email: olivier.liess@univ-avignon.fr

(**) Universidade Estaduale do Ceará
MPCOMP
Av. Paranjana, 1700. Campus do Itaperi
Fortaleza - Ceará - CEP 601740-000
Brasil

email: philippe@uece.br

Abstract

A "pure" Constraint Programming Approach for the Resource-Constrained Project Scheduling Problem (RCPSP) is presented. Our basic idea was to substitute the resource constraints by a set of "sub-constraints" generated as needed. Each of these sub-constraints corresponds to a set of tasks that cannot be executed together without violating one of the resource constraints. A filtering algorithm for these sub-constraints has been developed. When applied to the initial resource constraints together with known filtering algorithms, this new filtering algorithm provides very good numerical results.

1 The Resource-constrained Project Scheduling Problem

The resource-constrained project scheduling problem (RCPSP) is one of the most general scheduling problems that is extensively studied in the literature (see e.g. [3] and the references there in). It consists in scheduling a project, i.e. a set of activities (or tasks) linked by precedence constraints and subject to resource constraints while minimizing the total duration of the project (the so called makespan). According to [3], it is classified under code $PS|prec|C_{max}$.

A project is made of a set of n activities (or tasks) linked by precedence constraints given by an activity-on-node network $G = (V, E)$. In such an acyclic network, each arc (i, j) represents a precedence constraint, i.e. activity j cannot start before the completion of activity i. It is assumed that $|V| = n + 2$ where 0 and $n + 1$ are dummy tasks representing the start and the end of the project

$((0,i) \in E$ and $(i,n+1) \in E, \quad \forall i)$. The processing time of activity i is noted p_i (with $p_0 = p_{n+1} = 0$). To each arc (i,j) of E is associated a "length" of p_i.

A set $\mathcal{R}$ of renewable resources is considered, each resource $k \in \mathcal{R}$ having a constant number R_k of units available at any time of the planning horizon T. Each activity i requires a non-negative amount r_{ik} of each resource $k \in \mathcal{R}$. Furthermore, the activities are non-preemptive, that is, once the activity has been started, it cannot be interrupted to be completed later.

As illustrated in figure 1, the (RCPSP) can be completely described by adding the resources consumptions on the activity-on-node network. In this example, 3 resources and 7 activities are considered. The time durations of the activities are reported on the on-going arcs (which can be considered as start-start relations with minimal time lags) and the resources consumptions are indicated on the nodes. The left part of the picture represents therefore the problem to be solved while the right part gives a feasible solution by reporting the activities on each resource using Gantt charts. For each resource, the horizontal axis represents the time and the vertical axis the resources consumptions.

One can note that, without the resource constraints, the problem would reduce to a longest path problem in an acyclic network and would then be easy to solve.

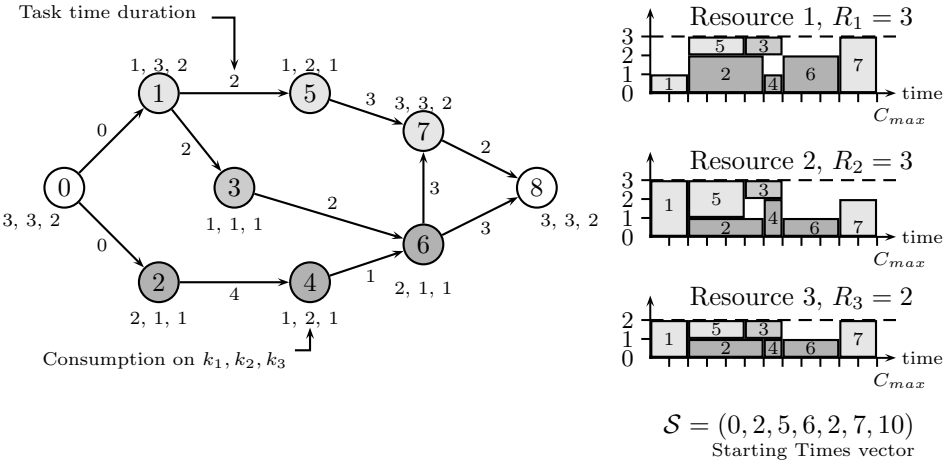

Figure 1: An example of (RCPSP)

The (RCPSP) being strongly NP-hard, most exact methods are of the branch-and-bound type (see e.g. [1, 3, 4, 5, 8, 9, 15]). The underlying principles of these methods are based on Integer Programming techniques, such as column generation ([13, 5, 15]) or cut generation ([17]), on Constraint Propagation ([1, 9, 10, 11]), on methods hybriding Integer and Constraint Programming ([2, 4, 7]) or on pure enumeration schemes([8]) Among these methods, it is generally assumed that the method of Demeulemeester and Herroelen ([8]) is the most efficient for small size problems. Nevertheless, this method, which uses cutsets (sets of unscheduled activities for which all their predecessors have al-

ready been scheduled), is also considered as not practical for medium or large size problems because of the high number of cutsets that have to be stored. In practice, none of these methods are able to solve 60 activities (RCPSP) on a regular basis. That's why many heuristic methods have also been developed (see [16] and the references there in).

Our aim here is to introduce a Constraint Propagation based branch-and-bound. Let us first introduce a clear definition of the (RCPSP).

Let S_i denote the starting time of activity i (with $S_0 = 0$). S_{n+1} represents thus the total project duration or makespan. The (RCPSP) can then be conceptually stated as follows :

$$\begin{array}{lll} \min & S_{n+1} & \\ \text{s.}t.: & S_j \geq S_i + p_i & \forall (i,j) \in E \\ & \sum_{i \in \mathfrak{S}(t)} r_{ik} \leq R_k & \forall k \in \mathcal{R} \quad \forall t \in \{0, ..., T\} \\ & S_i \in \{0, 1, ..., T - p_i\} & \forall i \end{array}$$

where $\mathfrak{S}(t)$ is the set of activities i such that $S_i \leq t \leq S_i + p_i$ and T an available upper bound to the optimal project duration (i.e. the makespan of a feasible solution).

Methods using Constraint Programming ([1, 2, 4, 7, 9, 10, 11, 13]) either for solving the whole problem or as preprocessing to another technique, are based on the same pattern. This pattern will be introduced in section 2. Section 3 explains how the general pattern is modified in our approach. Finally, section 5 provides some numerical experiments.

2 Constraint Programming for the (RCPSP)

Constraint Programming techniques associate to each decision variable a set of possible values. In scheduling problems, several quantities are generally associated to a given activity i:

- the earliest possible starting time es_i.
- the latest possible starting time ls_i.
- the earliest possible finishing time ef_i (note that $ef_i = es_i + p_i$).
- the latest possible finishing time lf_i (note that $lf_i = ls_i + p_i$).

Hence, the interval $[es_i, ls_i]$ is the set of possible values for S_i. Initially es_i is the length of the longest path in G from 0 to i, while ls_i is T minus the length of the longest path in G from i to $n + 1$.

Constraint Programming consists in applying "filtering" algorithms (each of them being associated to a constraint) for eliminating the values which became

impossible with respect to the current state of the solution process.

Constraint Programming literature for the (RCPSP) (see [1, 4, 7, 9, 10, 11] for examples) indicates that the precedence constraints are associated to a longest path algorithm, such as Bellman (see [12], p. 44) or Floyd-Warshall (see [12], p. 49). In practice, despite its higher worse case complexity ($O(n^3)$), the Floyd-Warshall is used since it computes the longest path between each pair of nodes, providing, in one shot, updated es_i, lf_i, but also a matrix B such that B_{ij} is the longest path between nodes i and j, that is a lower bound to the time duration that must occur between starting times of tasks i and j. Initially, the algorithm is called without taking into account the resources constraints (reducing from $\{0, 1, ..., T - p_i\}$ to $[es_i, ls_i]$ the possible values for S_i), but each time the filtering algorithms associated to the resources constraints increase a es_i or decrease a ls_i may imply that the set of possible values for the starting times of some other activities are also reduced so that the longest path algorithm has to be called again.

The resources constraints are much more difficult to tackle and generally necessitate several algorithms. The first (and most commonly used) algorithm is the so called "resource timetable" algorithm (see [1, 9, 10, 11] for instance). With each of the resources of the problem is associated a timetable reporting the number of units of the resource still available at each time $t \in [0, T]$. If, at a given t, these number of units is smaller than the requirement of a task i, then i cannot be executed at time t, so that es_i (respectively ls_i) can be increased (respectively decreased). Note that one of the drawbacks of this algorithm is that deductions can only be made after some tasks have been fixed. Theoretically, the resources timetables are sufficient to take into account the resources limitations. However, in practice, most of the Constraint Programming approaches for the (RCPSP) include more filtering algorithms.

These algorithms address some particular situations and are much more powerful than the resources timetables if the corresponding situation happens. Of particular interest is the disjunctive edge-finding algorithm of Carlier-Pinson ([6]) which reduces the intervals $[es_i, ls_i]$ for $i \in \mathcal{C}$ where $\mathcal{C}$ is a set of activities such that $\forall \ \ i, j \in \mathcal{C}$, i and j cannot be executed in parallel because of either the precedence constraints or the resources constraints. Edge-finding consists in considering a subset Ω of $\mathcal{C}$, an activity $i \in \mathcal{C}$, not in Ω, and to test several rules.
An example of such rules is to schedule i before any task of Ω. If an inconsistency (that is if $\max\{ef_j, j \in \Omega\} - \min\{\max\{es_j, es_i + p_i\}, j \in \Omega\} > \sum_{j\in\Omega} p_j$) is detected then, for sure, i can only start after a task of Ω (since none of these activities can be executed in parallel) so that es_i can be updated to $\max\{es_i, \min\{ef_j, j \in \Omega\}\}$.
Another example of rules is illustrated on figure 2. This rule consists in checking if

$$\max\left(lf_i, \max_{\omega\in\Omega} lf_\omega\right) - \min_{\omega\in\Omega} es_\omega < p_i + \sum_{\omega\in\Omega} p_\omega$$

If such a situation occurs, then i has to be scheduled before any task of Ω.

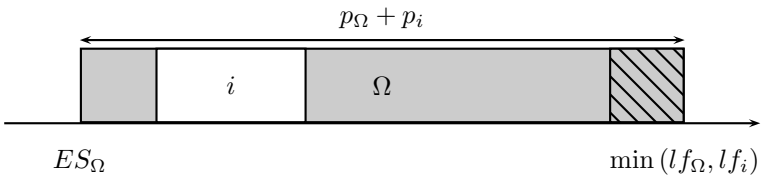

Figure 2: Example of edge-finding rule

A good survey of Constraint Programming techniques for the (RCPSP) can be found in [10]. Such techniques include the so called symmetric triple rules (initially proposed in [4]), the energetic rules (see [1]), the input/output, input-or-output, input/output negation consistency tests and shaving ([10, 11]).

Hence, Constraint Programming consists in iteratively calling all the filtering algorithms up to a state where it is not possible to reduce any of the intervals $[es_i, ls_i]$. When such a situation occurs, the current feasible set is partitioned into two (or more) subsets, each of the subproblems giving a new node of the branch-and-bound tree.

Our aim here is to design a Constraint Propagation scheme for the (RCPSP) where the resources timetables are not considered. Our basic idea is that this mechanism requires too much CPU time with respect to the interval reductions it deduces. More precisely, we want to take into account the resources constraints only when it is "necessary", that is whenever a subset of activities cannot be scheduled in parallel although the precedence constraints only would lead them to be scheduled in parallel. In some sense, the process described in the next section is similar to a cut generation process.

3 Substituting the resources constraints by sub-constraints

Our approach includes the Floyd-Warshall algorithm for the precedence constraints, the edge-finding algorithm and the symmetric triple algorithm for (partially) tackling the resources constraints. Hence, at a current node, we can find a partial solution, not detected as infeasible by these algorithms but which can actually violate one of the resources constraints (since they are not explicitly considered). In such a case, we just found a set of activities $\mathcal{T}$ that should not be executed in parallel (with respect to the resources capacities) but that are currently scheduled in such a way that there exists a time t such that all the tasks of $\mathcal{T}$ are in progress at t.

Obviously, in such a situation, the current node has to be pruned. However, since we would like to avoid to find again the same set of tasks scheduled in parallel, a constraint associated to this set of tasks only is generated and definitively included in the problem.

The filtering algorithm that we associate to this constraint is based on the following observation: if an activity i is such that $ls_i \leq ef_i$ then i is necessarily in progress during the interval $[ls_i, ef_i]$ (see figure 3). As a consequence, to each activity i of $\mathcal{T}$ can be associated an interval $I_i = [ls_i, ef_i]$ if $ls_i \leq ef_i$ and $I_i = \emptyset$ otherwise.

It is then possible to define a graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$ where $(i, j) \in \mathcal{E}$ if and only if $I_i \bigcap I_j \neq \emptyset$. $\mathcal{G}$ is thus a graph of intervals (see [12], p. 27). The filtering algorithm is based on the following easy-to-show property:

Property 1 .
Let C *be a clique of* $\mathcal{G}$ *and let* I_i, $i \in$ C *be the intervals associated to the nodes of* C. *Let also* $I_{\mathcal{C}}$ *be equal to* $\bigcap_{i \in \mathrm{C}} I_i$.
Then:

- $I_{\mathcal{C}} \neq \emptyset$
- $I_{\mathcal{C}} = [a, b]$ *with* $a = \max\{ls_i, \quad i \in \mathcal{C}\}$ *and* $b = \min\{ef_i, \quad i \in \mathcal{C}\}$

Proof .
By induction on the size s of the cliques.

For $s = 2$, both properties are trivial: since C is a clique, by construction of $\mathcal{G}$, the 2 corresponding intervals intersect. It is also easy to check that the second property is true, i.e. that if $I_1 = [ls_1, ef_1]$ and $I_2 = [ls_2, ef_2]$ then $I_1 \cap I_1 = [\max\{ls_1, ls_2\}, \min\{ef_1, ef_2\}]$.

Let us now assume that both properties are true for a given s. We are going to show that it is still true for $s+1$. Let us consider C, a clique of size $s+1$ and $\mathrm{C}^- = \mathrm{C} \,/\{j\}$ for a given $j \in$ C. C^- is a thus a clique of size s. The induction hypothesis implies then:

$$I_{\mathcal{C}^-} = [a^-, b^-] = [\max\{ls_i, \quad i \in \mathcal{C}^-\}, \min\{ef_i, \quad i \in \mathcal{C}^-\}] \neq \emptyset$$

Since, $I_{\mathcal{C}} = I_{\mathcal{C}^-} \cap [ls_j, ef_j] = [a^-, b^-] \cap [ls_j, ef_j]$, we have

$$I_{\mathcal{C}} = [\max\{a^-, ls_j\}, \min\{b^-, ef_j\}]$$

which states the second property. For the first property, assume that $I_{\mathcal{C}} = \emptyset$. We then have either $ls_j > b^-$ or $ef_j < a^-$. Since a^- is the maximum taken over a finite set, it is actually equal to ls_l for a given $l \in \mathcal{C}$. Therefore $ef_j < a^-$ is impossible since it would imply that $I_j \cap I_l = \emptyset$, a contradiction to the fact that $\mathcal{C}$ is a clique !!! The same holds for $ls_j > b^-$, completing the proof. □

This result allows to establish the following rules:

Property 2 .
Consider a clique C *of* $\mathcal{G}$. *Let* $I_{\mathcal{C}} = [a, b]$ *be equal to* $\bigcap_{i \in \mathrm{C}} I_i$. *During* $I_{\mathcal{C}} = [a, b]$, *resource* k *can only provide* $R_k - \sum_{i \in \mathcal{C}} r_{ik}$ *for the remaining activities. As a consequence, for any activity* j *not in* C *(but in* $\mathcal{T}$*):*

i) if $[es_j, ef_j] \bigcap I_{\mathcal{C}} \neq \emptyset$ *and* $r_{jk} > R_k - \sum_{i \in \mathcal{C}} r_{ik}$ *for some resource* k*, then* j *cannot start before* b*.*

ii) if $[ls_j, lf_j] \bigcap I_{\mathcal{C}} \neq \emptyset$ *and* $r_{jk} > R_k - \sum_{i \in \mathcal{C}} r_{ik}$ *for some resource* k*, then* j *cannot end after* a*.*

Figure 3 illustrates some possibilities of filtering. In this example, $\mathcal{T}$ is composed of 4 tasks that we assumed to be disjunctive (i.e. we assumed that no couple of tasks can scheduled in parallel). The clique C we consider is composed of task 1 only, so that $I_{\mathcal{C}} = [a, b] = [ls_1, ef_1]$. Task 2 does not meet the filtering condition i) or ii) and no filtering can be performed. Task 3 meets condition i) and, hence, its earliest starting time has to be set to b. Task 4 allows to detect an inconsistency: it also meets condition i), but it must start before b, hence the current state of the solution process is infeasible and the current node has to be pruned !

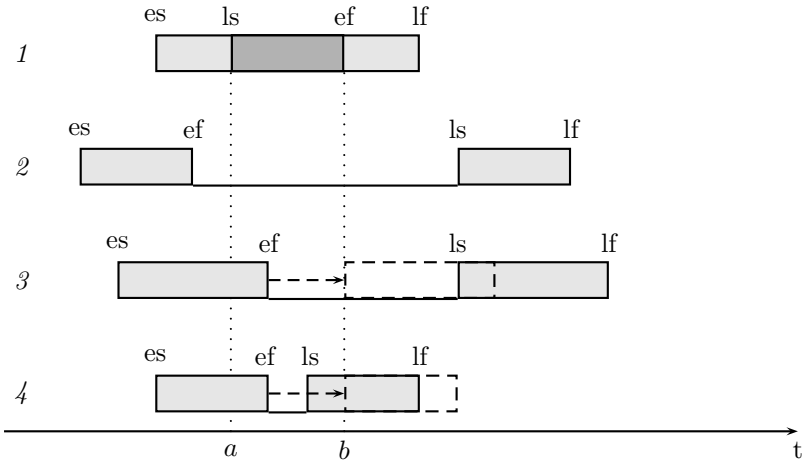

Figure 3: Filtering the sub constraint

In practice, for each "forbidden" set $\mathcal{T}$, we chose to heuristically generate a set of cliques such that each of the tasks of $\mathcal{T}$ is in, at least, one clique. Note however that the preliminary numerical results indicate that the forbidden set rarely includes more than 5 or 6 activities (so that it should be possible to generate all the cliques of $\mathcal{G}$). Algorithm 1 indicates with more details how this algorithm can be implemented. It fully takes advantage of the fact that $I_{\mathcal{C}} = [a, b]$ with $a = \max\{ls_i, \quad i \in \mathcal{C}\}$ and $b = \min\{ef_i, \quad i \in \mathcal{C}\}$.

Algorithm 1 first generates the cliques (lines 1 to 20). For this purpose, the clique generation consists in passing through the activities of $\mathcal{T}$ and, for each activity i, in computing a clique C, that is initialized with i, and to increase it

successively. Lines 21 to the end update the es_j and ls_j for the remaining tasks of $\mathcal{T}$, according to the rules of property 2.

Hence, our idea was to generate such "local resources constraints" each time a forbidden set is met, just like a cut generation process in Integer Programming. Therefore, at each node of the branch-and-bound tree, this solution method checks if no new forbidden set has appeared. Some numerical experiments were performed and they showed that this method was quite competitive, sometimes better, sometimes worst, than the results published in the literature. Nevertheless, as one can note, algorithm 1 can be applied on any set of tasks $\mathcal{T}$, and, in particular on the set constituted by all the tasks of the problem. In such a case, it becomes an alternative to the timetable filtering algorithm from which it differs mainly by the fact that it can make deductions before any task fixation.

4 Summary of the method

The whole approach is a classical Constraint Programming approach for the (RCPSP) ([11]), except that the Time Table algorithm is not considered. At each node of the branch-and-bound, the Floyd-Warshall algorithm (see [12], p. 49), the edge-finding ([6]), the symmetric triple rules ([4]) and the algorithm 1 of the previous section for each of the forbidden set known at the current node are successively performed (in this order). These algorithms are called until the problem reaches a state where either it is not possible to reduce any of the interval $[es_i, ls_i]$ or the current node is proved inconsistent or a feasible solution has been found.In the first case (no more reductions), the current feasible set is partitioned into two subsets, each of the subproblems giving a new node of the branch-and-bound tree.

The branching rule is as follows. We first branch on S_{n+1} by fixing it to $T-1$ (in practice, like many other authors, we took T equal to the best known solution). If the exploration of the subproblem results in showing that no solution exists for this makespan, then it turns out that T is a lower bound to the optimal value, hence the optimal value. Otherwise, the process can be repeated with $T = T-1$. Therefore, the initial optimization problem is transformed, at the first level of the branch-and-bound, into a Constraint Satisfaction Problem. At the subsequent levels, we branch first on the task i not yet fixed and which maximizes $\sum_k r_k^i p_i$. In the first subproblem to be explore, we fix the starting time of this task to its current lower bound (es_i). In the second subproblem, the starting time of the same task belongs to $[es_i + 1, ls_i]$. The tree exploration is realized according to the Last In First Out strategy.

The following section reports the numerical results of this method which happens to be very efficient.

5 Numerical results

The method has been implemented in C++ and numerical experiments were performed on the classical benchmarks of the literature, the so called KSD in-

Algorithm 1 filtering $\mathcal{T}$

for all $i \in \mathcal{T}$ such that $ls_i < ef_i$ **do**
  $C \leftarrow \{i\}$
  $Min \leftarrow ls_i$
  $Max \leftarrow ef_i$
  for all $k \in K$ **do**
    $R_k^t \leftarrow R_k$
  end for
  for all $t \in [ls_i, ef_i[$ **do**
    for all $j \in \mathcal{T} \setminus \{i\}$ such that $ls_j \leq t < ef_j$ **do**
      $Min \leftarrow \max(Min, ls_j)$
      $Max \leftarrow \min(Max, ef_j)$
      $C \leftarrow C \cup \{j\}$
      for all $k \in K$ **do**
        $R_k^t \leftarrow R_k^t - r_k^j$
      end for
    end for
    if $R_k^t < 0$ **then**
      End : the current node is infeasible
    else
      if C has not been generated before **then**
        for all $j \in \mathcal{T} \setminus \{C\}$ **do**
          for all $k \in K$ **do**
            if $r_k^j > R_k^t$ **then**
              if $(es_j < Max)$ and $(ef_j > Min)$ **then**
                if $ls_j < Max$ **then**
                  End : the current node is infeasible
                else
                  $es_j \leftarrow Max$
                end if
              else if $(ls_j < Max)$ and $(lf_j > Min)$ **then**
                if $ef_j > Min$ **then**
                  End : the current node is infeasible
                else
                  $lf_j \leftarrow Min$
                end if
              end if
            end if
          end for
        end for
      end if
    end if
  end for
end for

stances. These instances are available via a web site, maintained by Kolisch et al [14] (http://www.bwl.uni-kiel.de/Prod/psplib/datasm.html) where the best known upper and lower bounds are also reported. They have been randomly generated and contain problems of size 30, 60, 90 and 120 activities. Many of them are very easy to solve while a lot of others are still open problems (i.e. the best known upper and lower bounds do not coincide). For each problem size, the instances are divided in series of 10 problems. The series differ by several characteristics such as the density of the activity-on-node graph (i.e. the number of precedence constraints), the average resources consumption of the activities, etc... There are 48 series of 10 problems of size 30 (i.e. 480 problems), 48 series of 10 problems of size 60, 48 series of 10 problems of size 90 and 60 series of 10 problems of size 120. Each problem name is of the form jns-p, where n is the problem size, s the serie and p the instance in the serie for the size. For example, a problem named j12034-4 corresponds to the 4th problem of the 34th serie and is of size 120.

Table 1 reports the numerical results we obtained and compare with several other published results. It is reported the percentage (over the 480 problems for sizes 30, 60, and 90 and over the 600 problems for size 120) of problems solved to the optimum after a given amount of time (namely 30, 300, 360 and 3600 excepted for problems of size 120 for which we limited ourself to 900 seconds). Nevertheless, the machine we used (PC NEC PowerMate 2GHz, under Debian GNU/Linux) is faster than the machines on which the other methods were run. For this reason, we indicated in bold the results which should be compared, taking into account the differences in the machines. For example, we estimated that our machine is about ten times faster than the machine used by Dorndorf et al [9]. This estimation is the rate of the frequencies of the machines (note however that we always rounded in our disfavor). A "-" indicates that the result is not reported in the original paper.

From these results, it can be noted that our method is not able to solve all the instances of size 30 in a reasonable amount of time while the methods due to Demeulemeester-Herroelen [8] and to Klein et al [13] solve all the 480 instances quite quickly. Nevertheless, as the problem size increases, our method outperforms all the previous methods.

As a consequence of this efficiency, the method was able to close several instances still indicated as open on the web site. Tables 2, 3 and 4 report the instances that we closed (needless to say that much more problems were solved to optimality, we report in these tables only the problems which have never been solved before). It should also be emphasized that we unfortunately were not able to reduce any of the best known solution. In these tables, the first column indicates the problem name, the second the best known lower bound, the third the best known upper bound and the last one the CPU time we used for closing the instance. For all these problems, the optimal value corresponds to the best known upper bound (which is hardly surprising since these instances have been intensively studied). It can be noted that even for a given size, the times may considerably vary (for example for 60 tasks problems, j6014-3 has been solved in a few milliseconds, while problem j6042-3 necessitates more than 6000 seconds). This is a characteristic of the KSD instances which contain both very difficult

Table 1: Numerical results for KSD instances

j30	time (s)	This method	Dorndorf[a]	Klein[a]	Demeulemeester[a]	
	30	**96,0**	-	-	-	
	300	97,7	**95,4**	-	-	
	360	-	-	**100,0**	-	
	3600	98,1	-	-	**99,8**	
j60	time (s)	This method	Dorndorf	Klein	Sprecher[a]	Brucker[a]
	30	**79,4**	-	-	-	-
	300	81,2	**78,5**	**76,0**	**72,7**	-
	1800	81,9	80,0	80,2	75,8	-
	3600	82,1	-	80,9	-	67,9
j90	time (s)	This method	Dorndorf	Klein	Sprecher	Brucker
	30	**78,3**	-	-	-	-
	300	78,5	**76,0**	-	**61,5**	-
	1800	78,8	-	-	-	-
j120	time (s)	This method	Dorndorf	Klein	Sprecher	Brucker
	30	**39,2**	-	-	-	-
	300	39,8	**33,3**	-	-	-
	900	40,0	-	-	-	-

[a]as reported in [9]

and easy problems (however, all the problems reported in the tables below are supposed to be difficult since they were not solved before).

Aknowledgements

The second author thanks the Fundação Cearense de Apoio a Pesquisa (FUNCAP) and the Conselho Nacional de Pesquisa (CNPq) from Brazil for their financial support as well as 3 anonymous referees who considerably helped us in improving the presentation of this paper.

Name	LB	UB	Time (s)
j601-7	71	72	0.01
j603-8	54	55	0.01
j605-5	94	108	4593.04
j605-6	71	74	13.57
j605-8	72	78	27.16
j605-9	82	83	0
j606-6	53	55	1.77
j6014-3	61	62	0
j6021-1	91	103	186.42
j6021-2	103	108	0.53
j6021-3	81	87	1.28
j6021-5	81	89	465.92
j6021-6	77	84	18.8
j6021-7	95	103	9.92
j6021-8	101	110	46.23
j6021-10	77	80	3.37
j6022-4	70	73	14
j6026-2	65	66	0.33
j6026-4	65	67	225.35
j6026-6	73	74	0
j6026-9	63	65	98.81
j6030-3	80	82	8.5
j6033-6	73	75	1.31
j6037-1	89	97	83.12
j6037-2	82	95	7121.78
j6037-3	134	139	288.9
j6037-4	96	101	9.67
j6037-5	91	98	2.16
j6037-7	101	110	172.16
j6037-8	88	93	8.36
j6037-9	89	96	34.23
j6037-10	94	96	0.09
j6038-2	72	76	652.27
j6038-8	70	71	0.03
j6038-10	64	66	3.38
j6041-4	121	133	2972.55
j6041-7	124	132	5726.95
j6042-3	73	78	6360.82
j6042-4	100	103	4.66
j6042-8	80	82	602.74
j6046-1	78	79	0
j6046-8	73	75	78.72

Table 2: Closed KSD instances of size 60

Name	LB	UB	Time (s)
j901-1	72	73	0.06
j901-6	70	74	8.41
j901-7	90	91	0.03
j901-8	93	95	6.4
j901-9	70	72	138.5
j901-10	89	90	0.04
j906-3	75	77	1.08
j906-8	67	68	0
j9017-1	90	92	0.03
j9017-3	88	89	0.02
j9017-9	93	96	0.13
j9017-10	85	89	320.73
j9026-4	96	97	0
j9026-8	82	83	0
j9030-5	83	84	0.01
j9030-7	84	85	0
j9033-1	94	99	0.45
j9033-4	91	92	0.15
j9033-7	105	109	0.05
j9033-8	107	110	0.27
j9034-5	81	83	0.03
j9038-1	84	85	0.15
j9038-3	88	89	0
j9038-5	84	86	0.02
j9042-9	82	83	0.01
j9046-1	103	104	0.01
j9046-3	112	113	0.01

Table 3: Closed KSD instances of size 90

Name	LB	UB	Time (s)
j1201-5	108	112	1.69
j1201-7	109	117	253.94
j1201-9	109	112	0.06
j1202-4	94	95	0.07
j1202-6	90	92	0.08
j1202-8	81	83	296.14
j1203-1	79	80	0.02
j1208-7	87	88	0.02
j1208-8	87	88	0.02
j12021-4	131	135	0.13
j12021-6	107	109	0.24
j12021-9	97	102	0.63
j12022-9	107	109	0.08
j12024-7	110	112	0.01
j12025-6	91	92	0.02
j12025-8	78	80	0.05
j12033-6	115	116	0.01
j12034-4	94	95	0.02
j12035-5	92	93	0.02
j12038-9	134	135	0.02
j12041-5	137	138	707.86
j12041-10	128	136	37.29
j12042-3	104	106	106.57
j12042-9	102	104	0.25
j12043-4	101	105	19.84
j12043-7	121	122	0.17
j12044-8	107	108	0.09
j12045-5	114	116	0.06
j12049-8	112	113	0.02
j12049-9	97	98	0.01
j12054-4	119	120	0.02
j12059-3	108	109	0.02

Table 4: Closed KSD instances of size 120

References

[1] Baptiste, P., Le Pape, C.: Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problems, Constraints **5** (2000) 119–139.

[2] Baptiste, P., Demassey, S.: Tight LP Bounds for Resources Constrained Project Scheduling, OR Spectrum **26** (2004)251–262.

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

[4] Brucker, P., Knust, S., Schoo, A., Thiele, O.: A branch and bound algorithm for the resource-constrained project scheduling problem, European Journal of Operational Research **127** (1998) 272–288.

[5] Brucker, P., Knust, S.: A Linear programming and constraint propagation for the RCPSP, European Journal of Operational Research **127** (2000) 355–362.

[6] Carlier, J., Pinson, E.: A practical use of Jacson's preemptive schedule for solving the job-shop problem. European Journal of Operational Research **78** (1991) 146–161.

[7] Demassey, S., Artigues, C., Michelon, P.: Constraint Propagation based Cutting Planes: an Application to the Resource-constrained Project Scheduling Problem, To appear in INFORMS Journal on Computing (2005).

[8] Demeulemeester, E.L., Herroelen, W.S.: New benchmark results for the resource-constrained project scheduling problem, Management Sciences **43** (1997) 1485–1492.

[9] Dorndorf, U., Pesch, E., Phan-Huy, T.: A branch and bound algorithm for the resource-constrained project scheduling problem, Mathematical Methods of Operations Research **52** (2000) 413–439.

[10] Dorndorf, U., Pesch, E., Phan-Huy, T.: A survey of interval capacity consistency tests for time and resource constrained scheduling, *Project Scheduling - Recent Models, Algorithms and Applications* (Kluwer Academic Publ., 1999) 213–238.

[11] Dorndorf, U., Pesch, E., Phan-Huy, T.: Constraint propagation techniques for disjunctive scheduling problems, Artificial Intelligence **122** (2000) 189–240.

[12] Gondran, M., Minoux, M.: *Graphes et algorithms* (Eyrolles, 1990).

[13] Klein, R., Scholl., A.: Computing lower bounds by destructive improvement - an application to RCPSP, European Journal of Operational Research, **112** (1999) 322–346.

[14] Kolisch, R., Sprecher, A.: PSPLIB - A project scheduling library, European Journal of Operational Research, **96** (1996) 205–216.

[15] Mingozzi, A., Maniezzo, S., Ricciardelli, S., Blanco, L.: An exact algorithm for project scheduling with resource constraint based on a new mathematical formulation, Management Sciences **44** (1998) 714–729.

[16] Palpant, M., Artigues, C., Michelon, P.: LSSPER: a Large Neighborhood Search for the Resource-constrained Project Scheduling Problem, Annals of Operations Research **31** Special issue on Metaheuristics (2004) 237–257.

[17] Sankaran, J., Bricker, D., Juang, S.: A strong fractionnal cutting-plane algorithm for RCPSP, International Journal of Industrial Engineering **6** (1999) 99–111.