Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

STEFAN HEINZ JENS SCHULZ**

Explanations for the Cumulative Constraint: an Experimental Study*

* Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.
** Technische Universität Berlin, Institut für Mathematik, Straße des 17. Juni 136, 10623 Berlin, Germany

ZIB-Report 11-13 (May 2011)

Explanations for the Cumulative Constraint: an Experimental Study*

Stefan Heinz[1] and Jens Schulz[2]

[1] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
heinz@zib.de
[2] Technische Universität Berlin, Institut für Mathematik, Straße des 17. Juni 136, 10623 Berlin, Germany
jschulz@math.tu-berlin.de

Abstract. In cumulative scheduling, conflict analysis seems to be one of the key ingredients to solve such problems efficiently. Thereby, the computational complexity of explanation algorithms plays an important role. Even more when we are faced with a backtracking system where explanations need to be constructed on the fly.
In this paper we present extensive computational results to analyze the impact of explanation algorithms for the cumulative constraint in a backward checking system. The considered explanation algorithms differ in their quality and computational complexity. We present results for the domain propagation algorithms time-tabling, edge-finding, and energetic reasoning.

1 Introduction

In cumulative scheduling we are given a set of jobs that require a certain amount of different resources. In our case, the resources are renewable with a constant capacity and each job is non-interruptible with a fixed processing time and demand request for several resources. A resource can be, for example, a group of worker with the same specialization, a set of machines, or entities like power supply.

Cumulative scheduling problems have been tackled with techniques from constraint programming (CP), integer programming (IP), or satisfiability testing (SAT). In recent years hybrid approaches are developed which combine methods from these areas. Currently, the best results are reported by a hybrid solver which uses CP and SAT techniques [13]. However, there are still instances with 60 jobs and four cumulative constraints published in the PSPLib [12] that resist to be solved to proven optimality.

Several exact approaches use a search tree to solve cumulative scheduling problems. The idea is to successively divide the given problem instance into smaller subproblems until the individual subproblems are easy to solve. The

* Supported by the DFG Research Center Matheon *Mathematics for key technologies* in Berlin.

best of all solutions found in the subproblems yields the global optimum. During the course of the algorithm, a *search tree* is created with each node representing one of the subproblems. These subproblems are usually generated by adding bound changes, called *branching decisions*, to the current problem. That means the feasibility region gets restricted. At each subproblem, mathematically sophisticated techniques exclude further values from the variables' domains. One of them is domain propagation which infers bound changes on the variables.

Recently it was discovered that *conflict analysis* plays an import role to solve cumulative scheduling problems efficiently [13, 7]. Conflict analysis is used to analyze infeasible subproblems which arise during the search in order to generate *conflict clauses* [10, 1] also known as *no-goods*. These conflict clauses are used to detect similar infeasible subproblems later in the search. In order to perform conflict analysis, a bound change which was performed during the search needs to be explained. Such an *explanation* is a set of bounds which infer the performed bound change. Note that bound changes which are branching decisions cannot be explained. There are two common ways to collect an explanations. One is to submit it directly with the bound change, called *forward checking*. The other way is to reconstruct an explanation on demand which means only if it is needed. This is known as *backward checking*. Both versions have their advantages and disadvantages. A forward checking system always constructs an explanation even if it is not needed, whereas the backward checking framework needs to be able to reconstruct an explanation for a bound change at any point during the search. In the latter case it can be computationally expensive to compute an explanation lazily, since the same bound changes might be explained multiple times.

In this paper we present extensive experimental results which give evidence that minimum-size explanations of bound changes are a crucial component for solving cumulative scheduling instances efficiently. We analyze the impact of constructing explanations in a backward checking system. For our study we consider the domain propagation algorithms time-tabling, edge-finding, and energetic reasoning, see [6]. In case of time-tabling, the complexity status of computing a minimum size explanation is still open. Thus, we evaluate three different heuristic approaches to deliver an appropriate explanation of small size. As benchmark set we use instances of the problem library PSPLib [12] and Pack instances from [4].

Related work. Scheduling problems have been widely studied in the literature. For an overview we refer to [4, 8]. The cumulative constraint was introduced by Aggoun and Beldicneau [3]. Current state-of-the-art propagation algorithms for cumulative are surveyed in [6, 4]. Best results on the instances we are focusing on are achieved by a solver combining CP and SAT techniques [13].

Learning from infeasible subproblems is one of the key ingredients in modern SAT solvers. This technique is called conflict analysis. The basic idea is to conclude a *conflict clause* which helps to prune the search tree and enables the solver to use *non-chronological backtracking*. For a detailed description see for example [10]. There are two main differences of IP and SAT solving in the context of conflict analysis. First, the variables of an IP do not need to be of binary type. Therefore, we have to extend the concept of the conflict graph: it has to

represent bound changes instead of variable fixings, see [1] for details. Second, infeasibility cannot only arise by propagation but also due to an infeasible linear program relaxation [1]. To be able to perform a conflict analysis, it is essential to explain bound changes. This means to state a set of bounds which lead to the proposed bound change. From that a conflict graph is created, then a cut in this graph is chosen, which produces a conflict clause that consists of the bound changes along the frontier of this cut. For cumulative constraints, explanation algorithms to some of the known propagation algorithms are given in [13, 7].

Outline. Section 2 introduces the notation of the considered scheduling problem. In Section 3 we present the propagation and different explanation algorithms that are used in our experimental study, presented in Section 4.

2 Cumulative Scheduling

In cumulative scheduling, an instance is given by a set $\mathcal{J}$ of n non-preemptable jobs with processing times $p_j \in \mathbb{N}$ for each job $j \in \mathcal{J}$. Each job j requests a certain demand r_j of a cumulative resource with capacity $C \in \mathbb{N}$. In a contraint program, a *cumulative* constraint is given by $\texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, i.e., vectors of start times, processing times and demands, and the capacity. The cumulative constraint enforces that at each point in time t, the cumulated demand of the jobs running at t, does not exceed the given capacity, i.e.,

$$\sum_{j \in \mathcal{J} : t \in [S_j, S_j + p_j)} r_j \leq C \qquad \text{for all } t.$$

Depending on the tightness of the *earliest start times* (est_j), *earliest completion times* (ect_j), *latest start times* (lst_j), and *latest completion times* (lct_j) for each job $j \in \mathcal{J}$, propagation algorithms are able to update variable bounds. Since we use start time variables S_j, the lower bound corresponds to est_j and the upper bound corresponds to lst_j.

3 Propagation and Explanation Algorithms

Explanations tend to create stronger conflict clauses if they include only few variables since we could expect that the constructed conflict graph has a smaller width and size. Hence, one would like to search for minimum sized explanations. On the other side, we are facing a backward checking system which implies that bound changes have to be explained several times during the search. Therefore, explanation algorithms should have a small complexity. In case of the cumulative constraint, computing a minimum sized explanation stands in contrast to a reasonable complexity. In this section we briefly introduce the three propagation algorithms used for our experiments. For each algorithm we state three variants to generate an explanation for a bound change. These constructions differ in their quality (the size of the explanation) and their computational complexity.

We only consider lower bound (est_j) adjustments of the start variables S_j. Upper bound (lst_j) changes can be treated symmetrically. To keep the notation simple, we assume for each interval $[a, b)$ that the condition $a < b$ holds even it is not explicitly mentioned.

3.1 Energetic Reasoning

Energetic reasoning checks non-empty time intervals $[a, b)$, with $a < b$, whether the jobs contributing to that interval require more energy than available. That is why, it has also been considered under the name *interval consistency test*. There are $O(n^2)$ intervals to be checked, see [5]. The available energy of such an interval is given by $C \cdot (b-a)$. The energy of a job is the product of its processing time and its demand. For a job j the required energy $e_j(a, b)$ for such an interval is given by:

$$e_j(a, b) := \max\{0, \min\{b - a, p_j, \text{ect}_j - a, b - \text{lst}_j\}\} \cdot r_j.$$

Hence, $e_j(a, b)$ is the non-negative minimum of (i) the energy if it runs completely in the interval $[a, b)$, i.e., $(b-a)\cdot r_j$, (ii) the energy of job j, i.e., $p_j \cdot r_j$, (iii) the left-shifted energy, i.e., $(\text{ect}_j - a)\cdot r_j$, and (iv) the right-shifted energy, i.e., $(b-\text{lst}_j)\cdot r_j$.

We can make the following deductions, see Baptiste et.al. [5] for further readings. First, in case an interval is *overloaded*, i.e., the required energy $E(a, b) := \sum_j e_j(a, b)$ is larger than $C \cdot (b - a)$, the problem is infeasible. Second, the earliest start time (lower bound) of a job j can be updated using any non-empty interval $[a, b)$ which intersects with job j according to the following equation:

$$\text{est}'_j = a + \left\lceil \frac{1}{r_j}\big(E(a, b) - e_j(a, b) - (b - a) \cdot (C - r_j)\big) \right\rceil.$$

A proof is given in [5].

In the following lemmas we state conditions on a set of bounds to achieve the deductions. In case both bounds of a job are responsible, we say, the job is reported. Otherwise, we explicitly mention the bound of interest.

Lemma 1. *An overload of interval $[a, b)$, with $a < b$, can be explained by a set $\Omega \subseteq \mathcal{J}$ such that*

$$\sum_{j \in \Omega} e_j(a, b) > C \cdot (b - a). \tag{1}$$

Lemma 2. *A lower bound update of job j to est'_j due to interval $[a, b)$, with $a < b$, intersecting with $[\text{est}_j, \text{ect}_j)$ can be explained by the previous lower bound of job j and a set $\Omega \subseteq \mathcal{J} \setminus \{j\}$ such that*

$$\sum_{i \in \Omega} e_i(a, b) > (C - r_j)(b - a) + (\text{est}'_j - a) \cdot r_j - r_j. \tag{2}$$

To construct such a sub set of jobs Ω for a lower bound update we compare in our experimental study three different algorithms:

Variant 1
Report all jobs $i \in \mathcal{J} \setminus \{j\}$ with $e_i(a,b) > 0$.
Variant 2
Report jobs $i \in \mathcal{J} \setminus \{j\}$ with $e_i(a,b) > 0$ until the Condition (2) is satisfied.
Variant 3
First, sort the jobs with respect to their energies $e_i(a,b)$ in non-increasing order and report jobs until Condition (2) is satisfied.

If the interval $[a,b)$, which inferred the lower bound change, is known, Variant 1 runs in linear time as Variant 2, which additionally needs a pre-computation of the necessary energy. Because of the sorting, Variant 3 runs in $O(n \log n)$. Observe that Variant 3 reports a minimum sized explanation with respect to interval $[a,b)$.

Note that in case of an overloaded interval (Lemma 1) the above explanation algorithms can be easily adjusted by considering the complete set of jobs $\mathcal{J}$ as basis and use Condition (1) as stopping criterion in Variants 2 and 3.

3.2 Edge-Finding

Edge-finding can be seen as a special variation of energetic reasoning. In that version the energy requirement of a job is only considered if the job lies completely in the interval $[a,b)$, i.e., $\text{est}_j \geq a$ and $\text{lct}_j \leq b$. This clearly leads to weaker bound updates, but can be executed with a smaller computational complexity using sophisticated data structures, see [14]. We use the same explanation algorithms as for energetic reasoning. Note that besides the jobs which lie completely in the interval $[a,b)$, we can also consider jobs which partly intersect with $[a,b)$. In case of constructing a suitable set of jobs Ω this has no influence on the computational complexity.

3.3 Time-Tabling

Time-tabling can be seen as a unit-interval capacity consistency test. For each interval of size one, a test is performed as by energetic reasoning. Since this attempts to be too time-consuming, implementations focus on a profile-based view. This profile is constructed using the cores of each job, see [9]. For a job j a *core* γ_j is given by the interval $[\text{lst}_j, \text{ect}_j)$. This is the interval where parts of the job j must be executed. Note that this interval might be empty.

We denote by $\text{peak}_t(\mathcal{J})$ the height of the resource profile at time t which is generated by the cores of jobs $\mathcal{J}$. Obviously, if $\text{peak}_t(\mathcal{J}) > C$ holds for some t then the corresponding cumulative constraint is infeasible. On the other hand, variable bound adjustments can be made as follows. Consider a job j. First, remove the core from job j out of the profile. Search in the interval $[\text{est}_j, \text{lst}_j]$, starting from est_j, the first time point such that job j can be scheduled without creating a profile peak exceeding the capacity. That time point est'_j defines a lower bound on the start time for job j. The bounds that are responsible in either case are stated in the following lemmas. Proofs are omitted.

Lemma 3. *An infeasibility due to* $\text{peak}_t(\mathcal{J}) > C$ *at a time point* t *can be explained by a set* $\Omega \subseteq \mathcal{J}$ *such that*

$$\sum_{j \in \Omega : t \in \gamma_j} r_j > C.$$

Lemma 4. *A lower bound update of job* j *to* est'_j *can be explained by the previous lower bound of* j *and a set* $\Omega \subseteq \mathcal{J} \setminus \{j\}$ *such that for all intervals* $I \in \{[\text{est}'_j - 1, \text{est}'_j)\} \cup \{[a, b) \subseteq [\text{est}_j, \text{est}'_j) \mid b - a = p_j\}$ *the following condition holds*

$$\exists\, t \in I : \sum_{i \in \Omega : t \in \gamma_i} r_i > C - r_j. \tag{3}$$

Note that even in the special case of jobs with unit processing times, it is an open question whether it is $\mathcal{NP}$-hard to find an explanation of minimum size. In the following we describe three different techniques to derive an explanation for a lower bound updated by the time-tabling algorithm. These approaches differ in their computational effort. Consider the lower bound update of job j from est_j to est'_j.

Variant 1

Report all variables whose core intersect with the interval $[\text{est}_j, \text{est}'_j)$.

Variant 2

(i) Sort jobs in non-decreasing order w.r.t. their demands.
(ii) For each $t \in [\text{est}_j, \text{est}'_j)$ with $\text{peak}_t(\mathcal{J} \setminus \{j\}) > C - r_j$ report jobs $i \in \mathcal{J} \setminus \{j\}$ with $t \in \gamma_i$ until Condition (3) is satisfied.

Variant 3

(i) Sort jobs in non-decreasing order w.r.t. their demands.
(ii) Set $t = \text{est}'_j - 1$.
(iii) If $t < \text{est}_j$ stop.
(iv) Explain $\text{peak}_t(\mathcal{J} \setminus \{j\})$.
(v) Find smallest time point $t' \in [t - p_j, t)$ such that $\text{peak}_{t'}(\mathcal{J} \setminus \{j\}) > C - d_j$ holds.
(vi) Set $t = t'$ and goto (iii).

Note that in Variants 2 and 3 we are starting with the largest time point, i.e., $\text{est}'_j - 1$, and report all cores until we satisfy Condition (3). For the remaining peaks we first compute the contribution of previously stated jobs and only add as many new jobs to the explanation until we fulfill Condition (3). Variant 1 runs in linear time, Variant 2 explains each peak larger than $C - d_j$, and Variant 3 tries to report only a few peaks. For the two latter once we need $O(n \log n)$ for sorting the jobs in non-decreasing order w.r.t. their demands. The number of time points that need to be considered is linear in the number of jobs.

4 Experimental Study

In this section we describe the computational environment, introduce the selected test instances, and finally present and discuss the computational results.

4.1 Computational Environment

For performing our experimental study we used the non-commercial constraint integer programming framework SCIP [2], version 2.0.1.1. We integrated CPLEX release version 12.20 as underlying linear programming solver. All computations reported were obtained using Intel Xeon 5150 core 2.66 GHz computers (in 64 bit mode) with 4 MB cache, running Linux, and 8 GB of main memory. A time limit of one hour was always enforced.

SCIP has a SAT-like conflict analysis mechanism and is a backtracking system. To avoid an overhead by constructing explanations for bound changes, it is possible to store additional information for each bound change. Since the number of stored bound changes is quite large during the search, the space for these information are restricted to 32 bits each. In case of energetic reasoning and edge finding we use these bits to store the responsible interval. Otherwise, we would need to search in worst case over $O(n^2)$ interval candidates. Hence, we can use the explanation algorithms stated in the previous section without any additional effort.

The basic version of SCIP also supports solving cumulative scheduling problem. For this study, we enhanced the capability for cumulative constraints and implemented the different explanation algorithms discussed in the previous section. We additionally used a scheduling-specific series-generation scheme based on α-points in order to generate primal solutions, see [11].

For our study we are interested in instances which are not trivial to solve on the one side and solvable on the other side. For all test sets we used the following criteria to restrict the test set to reasonable instances. We kept all instances which:

(i) could be solved to optimality by at least one solver,
(ii) at least one solver needed more than one search node, and
(iii) at least one solver needed more that one second of computational running time.

We collect resource-constrained project scheduling problem (RCPSP) instances from the problem library PSPLIB [12]. As bases we only choose test set J30 and J60, which are RCPSP instances with 30 and 60 jobs, respectively. Each test set has 480 instances. Applying the above criteria, we are left with 115 and 71 instances for the test sets J30 and J60. We omit the larger test sets J90 and J120 since for these cases the remaining set after filtering are too small. The collection of RCPSP instances in the PSPLIB is criticized for containing rather disjunctive problems. Therefore, we additionally considered the 55 Pack instances which are introduced by Artigues et.al [4]. The restricted set contains 28 instances.

4.2 Computational Results

In Section 3 we stated for the propagation algorithms time-tabling, edge-finding, and energetic reasoning three different explanation algorithms. We additionally

Table 1. Evaluation of time spend in conflict analysis for time-tabling on 115 instances from J30 and 71 instances from J60 and for energetic reasoning and edge-finding on 28 Pack instances.

test set	setting	solved	outs	better	worse	totaltime	expl. time	allopt	shnodes	shtime
time-tabling										
J30	no conflict	111	4	–	–	27329.3	–	105	2267 k	6.0
	no explanation	105	10	28	29	41182.8	–	105	2477 k	6.4
	Variant 1	115	0	55	6	15739.5	0.7%	105	871 k	2.5
	Variant 2	115	0	56	4	12328.1	0.9%	105	797 k	2.5
	Variant 3	115	0	55	3	9998.9	1.02%	105	791 k	2.4
J60	no conflict	69	2	–	–	19334.3	–	60	3815 k	10.6
	no explanation	60	11	8	38	55037.5	–	60	9212 k	25.6
	Variant 1	70	1	38	10	13420.4	1.3%	60	2008 k	7.1
	Variant 2	70	1	42	5	10207.9	1.64%	60	1759 k	5.7
	Variant 3	71	0	40	5	8800.0	1.76%	60	1510 k	5.6
energetic reasoning										
Pack	no conflict	23	5	–	–	21064.0	–	16	375 k	8.2
	no explanation	21	7	9	6	29267.6	–	16	467 k	14.2
	Variant 1	21	7	3	9	30028.5	0.27%	16	641 k	18.0
	Variant 2	19	9	4	9	39323.8	0.4%	16	677 k	18.9
	Variant 3	24	4	11	3	16869.6	0.35%	16	106 k	4.3
edge-finding										
Pack	no conflict	21	7	–	–	35921.1	–	16	471 k	7.7
	no explanation	17	11	3	8	41658.3	–	16	660 k	13.1
	Variant 1	19	9	7	4	35194.8	0.018%	16	388 k	5.9
	Variant 2	19	9	6	5	36442.5	0.032%	16	378 k	5.8
	Variant 3	19	9	6	4	37720.4	0.026%	16	385 k	5.5

consider two further settings. One in which the conflict analysis is globally disabled ("no conflict"). That means no infeasible problem is analyzed. Second, the cumulative constraint does not explain the bound changes such that all bound changes made by the cumulative constraint are considered as branching decisions ("no explanation").

For each run we only used the propagation algorithm of interest for retrieving domain reductions due to the cumulative constraints. All other scheduling specific techniques were disabled. The computational results showed that the effort spent by edge-finding and energetic reasoning compared the amount of reduction detected for the RCPSP instances are rather small. Most of the running time was used within these propagation algorithms. As a result, the running time of these propagation algorithms was dominating the time needed for constructing explanation in such a way that no differences between the explanation algorithms could be made. In contrast the time-tabling algorithm is too weak for the Pack instances which ended up in similar behavior. Therefore, we only present the results of the time-tabling explanation algorithms for the rather disjunctive instances of the test set J30 and J60 and the results of the propagation algorithms edge-finding and energetic reasoning for the Pack instances.

Table 1 presents the computational results for the three test sets. Column "solved" shows how many instances were solved to proven optimality whereas column "outs" states the number of instances which timed out. The next two

columns "better" and "worse" indicate how often a setting was 10% faster or slower than the reference solver which is the one performing no conflict analysis at all. To evaluate how much time was spent by the various explanation algorithms, column "totaltime" displays the total solving time in seconds and column "expl. time" the percentage of the total solving time used for the considered explanation algorithm. The column "allopt" gives the number of instances which were solved to proven optimality by all settings. These instances are used to compute the shifted geometric mean[3] of all nodes ("shnodes") in thousands and of the running time in seconds ("shtime"), respectively.

Treating domain reductions as branching decisions ("no explanation") performs worst in all cases. The number of solved instances decreases and the shifted number of nodes and the computation times increase. This is an interesting result since overall, we experience that using conflict analysis usually helps to solve instances faster. An explanation for this behavior is that the resulting conflict clauses are large and in most case the solver discards them. This means the time spent for constructing them was in these cases useless.

In case of time-tabling we observe that Variant 3 yields the best results on instances from J30, and J60 as well. Considering only the 60 instances of the test set J60 which are solved to optimality by all settings, Variant 1 of explanation algorithms for time-tabling decreases the average running time in the shifted geometric mean by 30% and Variant 3 even to 53%, the number of nodes are decreased by 47% and 40%, respectively. Column "expl. time" reveals that the more precise the explanation algorithm is, the more percentage of the total running time is spent on explaining the bound changes. In total the time spent on explaining the cumulative propagations, is negligible. For the test set J30 the results are similar, again the strongest variant of explanations yields the best results.

In case of energetic reasoning on the highly cumulative Pack instances, we observe that only Variant 3 performs better than the "no conflict" setting. Variants 1 and 2 show that greedily explaining bound changes may mislead the search and are worse than not using conflict analysis at all. For Variant 3 the shifted solving time reduced to 50% and the shifted number of nodes decreased to 28%.

The edge-finding algorithm performs worse w.r.t. the number of solved instances when using conflict analysis in any form compared to disabling it. Two instances were not solved anymore. Recall that we explain edge-finding via the demands as defined in energetic reasoning. One could expect this to be a good counterpart, but it does not pay. There are less instances solved then by energetic reasoning. Nevertheless, in case of the instances solved to optimality by all settings, we experience that all variants need roughly the same amount of shifted nodes and shifted time which decrease by 25% and 20%, respectively, in contrast to turning conflict analysis off.

[3] The shifted geometric mean of values $t_1, \ldots, t_n$ is defined as $\left(\prod(t_i + s)\right)^{1/n} - s$ with shift s. We use a shift $s = 10$ for time and $s = 100$ for nodes in order to decrease the strong influence of the very easy instances in the mean values.

5 Conclusions

We studied explanation algorithms for the cumulative constraint in a backward checking system. We presented extensive computational results. These show, that minimum sized explanations of bound changes are crucial in order to solve hard scheduling problem instances efficiently. Future research should focus on the complexity status of explanation algorithms for time-tabling or deliver approaches with reasonable computational complexity for at least some special cases.

References

1. Achterberg, T.: Conflict analysis in mixed integer programming. Discrete Optimization 4(1), 4–20 (2007), special issue: Mixed Integer Programming
2. Achterberg, T.: SCIP: Solving Constraint Integer Programs. Math. Programming Computation 1(1), 1–41 (2009)
3. Aggoun, A., Beldiceanu, N.: Extending chip in order to solve complex scheduling and placement problems. Mathematical and Computer Modelling 17(7), 57 – 73 (1993)
4. Artigues, C., Demassey, S., Neron, E.: Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. ISTE (2007)
5. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-based scheduling: applying constraint programming to scheduling problems. International Series in Operations Research & Management Science, 39, Kluwer Academic Publishers, Boston, MA (2001)
6. Baptiste, P., Pape, C.L.: Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. Constraints 5(1/2), 119–139 (2000)
7. Berthold, T., Heinz, S., Lübbecke, M.E., Möhring, R.H., Schulz, J.: A constraint integer programming approach for resource-constrained project scheduling. In: Lodi, A., Milano, M., Toth, P. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, LNCS, vol. 6140, pp. 313–317. Springer Berlin / Heidelberg (2010)
8. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. Eur. J. Oper. Res. 207(1), 1–14 (2009)
9. Klein, R., Scholl, A.: Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. European Journal of Operational Research 112(2), 322–346 (1999)
10. Marques-Silva, J.P., Sakallah, K.A.: Grasp: A search algorithm for propositional satisfiability. IEEE Transactions on Computers 48, 506–521 (1999)
11. Möhring, R.H., Schulz, A.S., Stork, F., Uetz, M.: Solving project scheduling problems by minimum cut computations. Manage. Sci. 49(3), 330–350 (2003)
12. PSPLib: Project Scheduling Problem LIBrary. `http://129.187.106.231/psplib/` (last accessed 2011)
13. Schutt, A., Feydy, T., Stuckey, P., Wallace, M.: Explaining the cumulative propagator. Constraints pp. 1–33 (2010)
14. Vilím, P.: Max energy filtering algorithm for discrete cumulative resources. In: van Hoeve, W.J., Hooker, J. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, LNCS, vol. 5547, pp. 294–308. Springer Berlin / Heidelberg (2009)