# A System for Generation and Visualization of Resource-Constrained Projects

Miquel BOFILL [a,1], Jordi COLL [a], Josep SUY [a] and Mateu VILLARET [a]
[a] *Universitat de Girona, Spain*

**Abstract.** In this paper we present a system for solving the Resource-Constrained Project Scheduling Problem (RCPSP) by means of reformulation into SMT. We extend previous results to RCPSP/max, where maximum delays between tasks are considered in addition to minimum delays. We also present a tool for easy modeling and solving of RCPSP/max instances, which allows a graphical representation of the solution obtained. The system is modular in the sense that new solvers, not necessarily based on SMT, can be plugged-in as external executables.

**Keywords.** Scheduling, RCPSP, Reformulation, SMT, Tools

## Introduction

The Resource-Constrained Project Scheduling Problem (RCPSP) is one of the most general scheduling problems that has been extensively studied in the literature. It consists in scheduling a set of non-preemptive activities (or tasks) with predefined durations and demands on each of a set of renewable resources, subject to partial precedence constraints. Durations and precedence constraints between tasks imply minimal distances between tasks (minimum time lags). Normally, the goal is to minimize the makespan, i.e., the end time of the schedule.

Many generalizations exist: for example, in RCPSP/max, minimum and maximum time lags are considered. That is, a maximum delay between the start time of every two tasks can be specified, in addition to the minimum delays implied by precedences. RCPSP/max is harder than RCPSP, since finding a feasible solution already becomes NP-hard. For a survey on variants and extensions of the RCPSP see [5].

Exact methods for solving the RCPSP are based on Mixed Integer Linear Programming (MILP) [11], Boolean satisfiability(SAT) [6], Satisfiability Modulo Theories (SMT) [1] and Lazy Clause Generation (LCG) [16]. The latter approach is state of the art in RCPSP and RCPSP/max.

In this paper we extend the approach of [1] based on SMT to RCPSP/max, and report some experimental results showing that our approach is competitive with LCG.

[1]Corresponding Author: Miquel Bofill, Universitat de Girona, Campus de Montilivi, Edifici P-IV, E-17071 Girona, Spain; E-mail: miquel.bofill@udg.edu

Moreover, we present a tool, similar to PSPSolver [15], to graphically represent RCPSP instances and their solutions. Our system differs from PSPSolver in several aspects:

- The user can create and modify instances.
- The user can plug-in any solver by simply setting its execution parameters, provided that the solver accepts the `rcp` or `sch` formats proposed in [10].
- Solutions can automatically be retrieved and graphically represented if they are supplied in a simple format (described in Section 3).
- The system supports several variants of RCPSP, such as: RCPSP/max, multi-mode RCPSP and multi-mode RCPSP/max.

We specially remark the feature of creating/modifying instances since we believe that this feature can be very helpful to the researcher.

The rest of the paper is organized as follows. In Section 1 we briefly introduce the RCPSP. In Section 2 we extend the SMT encodings for the RCPSP presented in [1] to RCPSP/max, and provide some experiments to evaluate the approach. In Section 3 we present the graphical tool. In Section 4 we conclude and point out some future research directions.

## 1. The Resource-Constrained Project Scheduling Problem (RCPSP)

The RCPSP is defined by a tuple $(V, p, E, R, B, b)$ where:

- $V = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ is a set of activities. $A_0$ and $A_{n+1}$ are dummy activities representing, by convention, the starting and finishing activities, respectively. The set of non-dummy activities is defined as $A = \{A_1, \ldots, A_n\}$.
- $p \in \mathbb{N}^{n+2}$ is a vector of durations, where $p_i$ denotes the duration of activity $i$, with $p_0 = p_{n+1} = 0$ and $p_i > 0, \forall i \in \{1, \ldots, n\}$.
- $E$ is a set of pairs representing precedence relations, thus $(A_i, A_j) \in E$ means that the execution of activity $A_i$ must precede that of activity $A_j$, i.e., activity $A_j$ must start after activity $A_i$ has finished. We assume that we are given a precedence activity-on-node graph $G(V, E)$ that contains no cycles; otherwise the precedence relation is inconsistent. Since the precedence is a transitive binary relation, the existence of a path in $G$ from node $i$ to node $j$ means that activity $i$ must precede activity $j$. We assume that $E$ is such that $A_0$ precedes all other activities and $A_{n+1}$ succeeds all other activities.
- $R = \{R_1, \ldots, R_m\}$ is a set of $m$ renewable resources.
- $B \in \mathbb{N}^m$ is a vector of renewable resource availabilities, where $B_k$ denotes the available amount of each resource $R_k$ per time unit.
- $b \in \mathbb{N}^{(n+2)\times m}$ is a matrix containing the resource demands of activities, where $b_{i,k}$ denotes the amount of resource $R_k$ required by activity $A_i$. Note that $b_{0,k} = 0$, $b_{n+1,k} = 0$ and $b_{i,k} \geq 0$, $\forall i \in \{1, \ldots, n\}$, $\forall k \in \{1, \ldots, m\}$.

A schedule is a vector $S = (S_0, S_1, \ldots, S_n, S_{n+1})$, where $S_i$ denotes the start time of activity $A_i$. We assume that $S_0 = 0$. A solution to the RCPSP is a non-preemptive[2] sched-

[2] An activity cannot be interrupted once it is started.

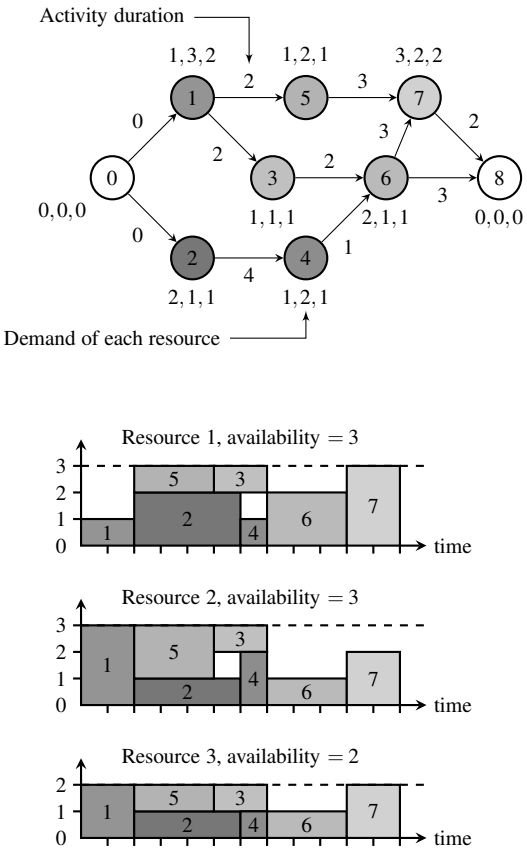


**Figure 1.** An example of RCPSP and one of its possible solutions [12].

ule $S$ of minimal makespan $S_{n+1}$, subject to the precedence and resource constraints. It can be stated as follows:

$$\text{minimize } S_{n+1} \tag{1}$$

subject to:

$$S_j - S_i \geq p_i \qquad \forall (A_i, A_j) \in E \tag{2}$$

$$\sum_{A_i \in \mathscr{A}_t} b_{i,k} \leq B_k \qquad \forall B_k \in B, \forall t \in H \tag{3}$$

where $\mathscr{A}_t = \{A_i \in A \mid S_i \leq t < S_i + p_i\}$ represents the set of non-dummy activities in process at time $t$, the set $H = \{0, \ldots, T\}$ is the scheduling horizon, with $T$ (the length of the scheduling horizon) being an upper bound of the makespan. A schedule $S$ is said to be feasible if it satisfies the precedence constraints (2) and the resource constraints (3).

In the example of Figure 1, three resources and seven (non-dummy) activities are considered. Each node is labeled with the number of the activity it represents. The durations of the activities are indicated in the on-going arcs, and the resource consumptions

are indicated in the labels next to the nodes. The upper part of the picture represents the instance to be solved, while the lower part gives a feasible solution using Gantt charts. For each resource, the horizontal axis represents the time and the vertical axis represents the consumption.

The work in [1] introduced the `rcpsp2smt`[3] system for solving RCPSP instances in the `rcp` and `sch` formats[4] by means of reformulation into SMT, using several SMT encodings, named *Time*, *Task* and *Flow*. We briefly recall the main ideas of the two former, which are the ones exhibiting best performance:

- The *Time* encoding is very similar to the MILP encoding proposed in [14]. The idea is to check, for every time step $t$ and resource $R_k$, that the sum of demands of all activities for resource $R_k$ at time $t$ does not exceed the availability of resource $R_k$. 0/1 variables are used to represent if an activity is being processed at each time step.
- The *Task* encoding is inspired by the encoding proposed in [13]. In this encoding, variables are indexed by activity number instead of by time. The key idea is that it suffices to check that there is no overload at the beginning (or end) of each activity. Contrarily to the *Time* encoding, here the number of variables does not depend on the time horizon.

## 2. The Resource-Constrained Project Scheduling Problem with Minimum and Maximum Time Lags (RCPSP/max)

RCPSP/max is a generalization of the RCPSP where the precedence graph $G(V,E)$ becomes $G(V,E,g)$, being $g$ an edge labeling function, valuating each edge $(A_i,A_j) \in E$ with an integer time lag $g_{i,j}$. Non-negative lags $g_{i,j} \geq 0$ correspond to a minimum delay in the start of activity $A_j$ with respect to the start of activity $A_i$. The case $g_{i,j} < 0$ corresponds to a maximum delay of $-g_{i,j}$ units in the start of activity $A_i$ with respect to the start of activity $A_j$ (see Figure 2). Note that the standard RCPSP can be seen as the particular case of RCPSP/max where only minimum time lags are considered, taking $g_{i,j} = p_i \; \forall (A_i,A_j) \in E$.

Regardless of minimizing the makespan, deciding if there exits a resource-feasible schedule that respects the minimum and maximum lags is NP-complete [2], and the optimization problem is NP-hard in the general case.

The *Time* and *Task* encodings proposed for RCPSP in [1] can be straightforwardly adapted to RCPSP/max as follows. We simply need to replace the constraints (2) by the following, where general time lags are considered instead of only durations:

$$S_j - S_i \geq g_{i,j} \qquad \forall (A_i,A_j) \in E \tag{4}$$

Although the constraints are almost the same as before, the preprocessing steps of [1] need to be adapted to this more general setting, as we describe in the next subsection.

[3] `http://imae.udg.edu/recerca/lap/rcpsp2smt/`
[4] RCPSP formats of the PSPLib [10].

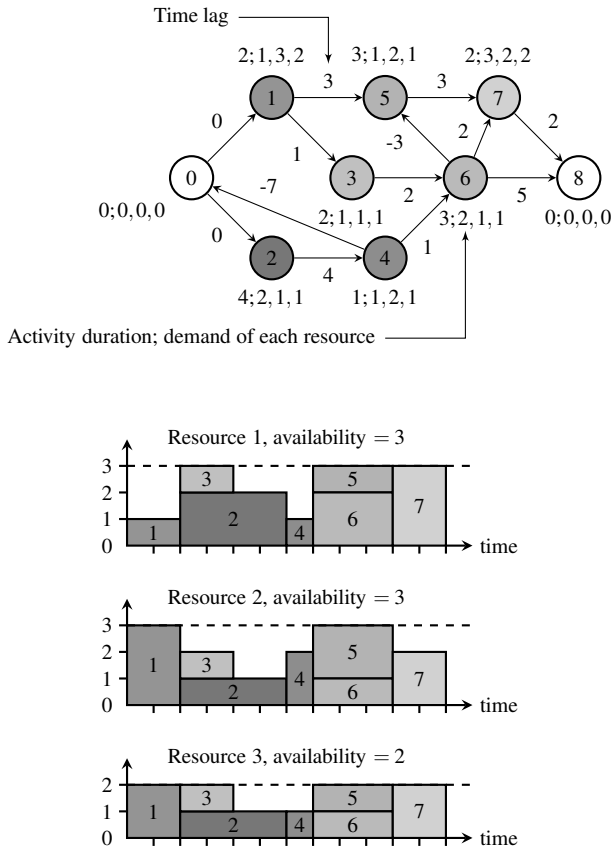


**Figure 2.** An example of RCPSP/max and one of its possible solutions.

### *2.1. Preprocessing*

In [1], several preprocessing computations are proposed to derive implied constraints that can help improve propagation during the solving process. Those include the computation of an extended precedence set, a lower and an upper bound for the makespan, time windows of activities and a matrix of incompatibilities between activities. By extended precedences the authors refer to minimum time lags between all pairs of activities. They are computed in polynomial time $\mathcal{O}(n^3)$, where $n$ is the number of activities, by running the Floyd-Warshall algorithm on the graph defined by the precedence relation $E$. For RCPSP/max, only positive edges must be considered.

The extended precedence set can be used to easily detect some unsatisfiable instances, given that an RCPSP/max instance is satisfiable only if the following two conditions hold:

1. There is no cycle of positive length, i.e., the minimum time lag between each activity and itself is zero.
2. There is no contradiction between the maximum and the minimum time lags between activities, i.e., for every negative edge $g_{j,i}$ indicating a maximum delay of $|g_{j,i}|$ time units in the start of activity $A_j$ after the start of activity $A_i$, we must

have that the minimum time lag between $A_i$ and $A_j$ in the extended precedence set is not greater than $|g_{j,i}|$.

Although finding a solution (with minimal makespan) for the RCPSP is NP-hard, a feasible schedule can be trivially found by concatenating all activities, i.e., without running any two activities in parallel. This determines a trivial upper bound. Moreover, well known heuristics, such as the *parallel scheduling generation scheme* [7,8] can be used to improve this bound. However, this is not the case for the RCPSP/max, due to the presence of maximum time lags. In fact, as commented before, in this case finding a feasible solution is already NP-complete. For this reason, the upper bound that we consider here is the following trivial one:

$$UB = \sum_{A_i \in A} \max(p_i, \max_{(A_i,A_j) \in E}(g_{i,j}))$$

The lower bound, time windows and incompatibilities between activities are calculated exactly as in [1].

### *2.2. Experiments*

We have carried out experiments on the following RCPSP/max instances accessible from the PSPLib [10]:

- Test sets J10, J20 and J30, each consisting of 270 instances with 5 resources and 10, 20 and 30 activities, respectively (cf. [9]).
- Test sets UBO10, UBO20, and UBO50, each consisting of 90 instances with 5 resources and 10, 20 and 50 activities, respectively (cf. [4]).

The experiments were run on an Intel® Xeon™CPU@3.1GHz, with 8GB of RAM, using Yices [3] version 1.0.40 as core SMT solver, with a cutoff of 600 seconds per instance.

All instances in the J10, J20, UBO10 and UBO20 test sets were successfully solved within the allowed time. For the J30 test set, only 7 instances could not be solved with the *Time* encoding, and 13 with the *Task* encoding. For the UBO50 test set, only 3 instances could not be solved with the *Time* encoding, and also 3 with the *Task* encoding.

These results show that our approach is competitive with the state of the art approach of LCG, which is an hybrid of a finite domain solver and a SAT solver: the results reported in [16] show between 7 and 9 unsolved instances in the J30 test set (depending on the strategy used) and 3 unsolved instances in the UBO50 test set, with a runtime limit of 600 seconds.

We ignore which was the memory limit used in the experiments of [16]. Unfortunately, our system raised an out of memory error for bigger instances like the ones in the UBO100 test set, consisting of 100 activities.

## 3. Description of the tool

VIEWPROJECT[5] is a desktop tool for visualizing and editing scheduling problems' instances. It represents in a graphic environment the resources, activities, and restrictions

[5] `http://imae.udg.edu/recerca/lap/viewproject/`

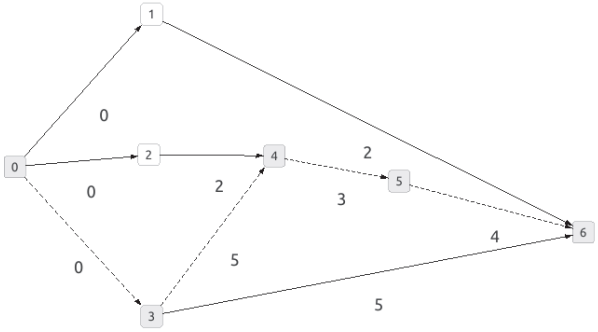


**Figure 3.** Graph with a leftmost distribution. The filled nodes are member of the critical path between the starting and finishing activities.

on time and resource usages in a project, and lets the user to modify them interactively. The user can create instances or modify existing ones by introducing or removing activities and updating their durations, resource usages and precedences. The user can also introduce or remove resources and modify their capacities. Finally, the system is also able to run external solvers to find schedules for the project and to plot them in Gantt charts, with the only restriction of getting solver's output in a specific format.

Problem instances are loaded and saved from and to plain text files. The supported encodings are `.rcp` files for RCPSP and `.sch` files for RCPSP/Max (see [10]).

The set of activities and their relations are displayed in an interactive directed and acyclic graph (see Figure 3). Each activity has its corresponding node labeled with its identifier number. The edges between pairs of activity nodes represent time constraints (e.g., precedences), and they are labeled with the associated time lags. The application offers a default set of node distributions giving visual information of precedences, but it is possible to move the nodes everywhere on the display area. The default distributions are the following:

- Leftmost distribution: All nodes are placed in columns and each node is in the leftmost possible column according to the precedence relation (i.e., being always at the right of all its predecessors).
- Rightmost distribution: All nodes are placed in columns and each node is in the rightmost possible column according to the precedence relation.
- Centered distribution: All nodes are placed in columns and as centered as possible according to the precedence relation.
- Circular distribution: The nodes are placed equidistantly on the edge of an imaginary circle. This distribution ensures that no two edges overlap completely. All nodes will be at the right or at the same vertical position of all their predecessors.

The rest of the project properties, such as the durations of the activities, resource usages and the capacity of the resources are displayed in text panels (see Figure 4).

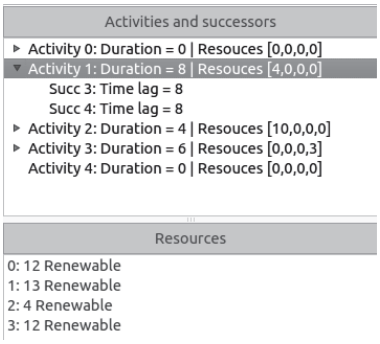


**Figure 4.** Editor panel of project properties.

Another of the utilities of the tool is to draw critical (longest necessary) paths between pairs of nodes in the graph.[6] In order to find the critical paths, the longest path between each pair of nodes is calculated with the all-pairs shortest (longest) path Floyd-Warshall algorithm.

It is possible to run external solvers to solve problem instances. For that purpose, the path to an executable file must be given to the tool, as well as the needed command line arguments. No other information than the arguments is passed to the solvers, so it is the solver's responsibility to load the problem instance from the source file. The expected output is a plain text in the standard output channel, with only the following tokens:

1. A sequence $\texttt{S\_0} = S_0$; $\texttt{S\_1} = S_1$; $\texttt{S\_2} = S_2$; ... where $S_i$ is the start time of activity $A_i$ in the schedule. It is mandatory to finish the sequence with an end of line character "$\backslash n$".
2. Optionally, it can be stated that the schedule is optimum with a final line:
   `s OPTIMUM FOUND`

If the output doesn't match with this template, it will be displayed as plain text.

Once the the solution is loaded, it is represented in $m$ Gantt charts, one for each resource $R_i$ in the project (see Figure 5). The horizontal edge represents the timeline, and the vertical edge the capacity of the resource. In the chart for a resource $R_i$, each activity $A_j$ is represented as a rectangle placed in the horizontal position corresponding to its start time $S_j$. The width of the rectangle is equal to the duration $p_j$ of activity $A_j$. The height of the rectangle is equal to the usage $b_{j,i}$ of resource $R_i$ by activity $A_j$. All the start times $S_0, \ldots, S_{n+1}$ are also displayed in a text panel.

## 4. Conclusion

We have presented an extension of the results in [1] for RCPSP to RCPSP/max, and showed that reformulation into SMT is a competitive approach for these problems in front of other methods. With preprocessing, an iterative process of minimization, and

[6] Note that the critical path between two nodes determines the minimum time lag between the two corresponding activities.

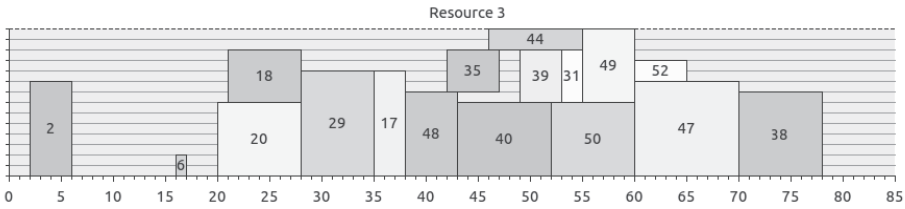


**Figure 5.** Gantt chart for a resource in a schedule.

using an SMT solver as core solver, we have obtained similar performance than those of state of the art systems.

As future work, a first goal is to take profit from maximal time lags to improve the initial upper bound of the schedule. This improvement could imply a reduction on the number of variables in the *Time* encoding, and hence, mitigate the memory problems when dealing with big instances. Also, the search for encodings that require less memory is planned as further work. In a broader sense, future plans involve using SMT to solve more general problems such as the Multi-Mode Resource Constrained Project Scheduling Problem (MRCPSP), the Multi-Mode Resource Constrained Project Scheduling Problem with Minimal and Maximal Time Lags (MRCPSP/max), the Multi-Skill Project Scheduling Problem (MSPSP), and the Resource Investment Problem with Minimal and Maximal Time Lags (RIP/max). In all these generalizations, with more complex Boolean structure, we believe that SMT will excel further.

We have also presented a tool for easy definition, modification and graphical representation of instances, as well as solutions. Contrarily to the library oriented approach of PSPSolver [15], our tool communicates with the RCPSP solver through text files in standard formats, so that any solver, based on any technology, can be easily plugged-in.

## Acknowledgements

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness through the project HeLo (ref. TIN2012-33042).

## References

[1] C. Ansótegui, M. Bofill, M. Palahí, J. Suy, and M. Villaret. Satisfiability Modulo Theories: An Efficient Approach for the Resource-Constrained Project Scheduling Problem. In *Proceedings of the 9th Symposium on Abstraction, Reformulation, and Approximation (SARA)*, pages 2–9. AAAI, 2011.

[2] M. Bartusch, R. H. Mohring, and F. J. Radermacher. Scheduling Project Networks with Resource Constraints and Time Windows. *Annals of Operations Research*, 16:201–240, January 1988.

[3] B. Dutertre and L. de Moura. The Yices SMT solver. Tool paper available at `http://yices.csl.sri.com/tool-paper.pdf`, August 2006.

[4] B. Franck, K. Neumann, and C. Schwindt. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR-Spektrum*, 23(3):297–324, 2001.

[5] S. Hartmann and D. Briskorn. A Survey of Variants and Extensions of the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 207(1):1 – 14, 2010.

[6] A. Horbach. A Boolean Satisfiability Approach to the Resource-Constrained Project Scheduling Problem. *Annals of Operations Research*, 181:89–107, 2010.

[7] J. Kelley. The critical-path method: Resources planning and scheduling. *Industrial scheduling*, pages 347–365, 1963.

[8] R. Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333, 1996.

[9] R. Kolisch, C. Schwindt, and A. Sprecher. Benchmark instances for project scheduling problems. In J. Weglarz, editor, *Project Scheduling*, volume 14 of *International Series in Operations Research & Management Science*, pages 197–212. Springer US, 1999.

[10] R. Kolisch and A. Sprecher. PSPLIB - A Project Scheduling Problem Library. *European Journal of Operational Research*, 96(1):205–216, 1997.

[11] O. Koné, C. Artigues, P. Lopez, and M. Mongeau. Event-Based MILP Models for Resource-Constrained Project Scheduling Problems. *Computers & Operations Research*, 38:3–13, January 2011.

[12] O. Liess and P. Michelon. A Constraint Programming Approach for the Resource-Constrained Project Scheduling Problem. *Annals of Operations Research*, 157:25–36, 2008.

[13] A. O. El-Kholy. *Resource Feasibility in Planning*. PhD thesis, Imperial College, University of London, 1999.

[14] L. J. W. Pritsker, A. Alan B. and P. S. Wolfe. Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach. *Management Science*, 16:93–108, 1996.

[15] J. Roca, F. Bossuyt, and G. Libert. PSPSolver: An Open Source Library for the RCPSP. In *26th workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2007)*, pages 124–125, 2007.

[16] A. Schutt, T. Feydy, P. Stuckey, and M. Wallace. Solving RCPSP/max by lazy clause generation. *Journal of Scheduling*, 16(3):273–289, 2013.