Satisfiability Modulo Theories: an Efficient Approach for the Resource-Constrained Project Scheduling Problem

Carlos Ansótegui
Departament d'Informàtica i Enginyeria Industrial,
Universitat de Lleida, Spain
carlos@diei.udl.cat

Miquel Bofill and **Miquel Palahí** and **Josep Suy** and **Mateu Villaret**
Departament d'Informàtica i Matemàtica Aplicada,
Universitat de Girona, Spain
{mbofill,mpalahi,suy,villaret}@ima.udg.edu

Abstract

The Resource-Constrained Project Scheduling Problem (RCPSP) and some of its extensions have been widely studied. Many approaches have been considered to solve this problem: constraint programming (CP), Boolean satisfiability (SAT), mixed integer linear programming (MILP), branch and bound algorithms (BB) and others. In this paper, we present a new approach for solving this problem: satisfiability modulo theories (SMT). Solvers for SMT generalize SAT solving by adding the ability to handle arithmetic and other theories. We provide several encodings of the RCPSP into SMT, and introduce `rcp2smt`, a tool for solving RCPSP instances using SMT solvers, which exhibits good performance.

Introduction

In this paper we consider the Resource Constrained Project Scheduling Problem (RCPSP)[1]. The RCPSP consists in scheduling a set of non-preemptive activities with predefined durations and demands on each of a set of renewable resources, subject to partial precedence constraints. Normally the goal is to minimize the makespan. This is one of the most general scheduling problems that has been extensively studied in the literature. Some surveys published in the last years include (Herroelen, Reyck, and Demeulemeester 1998; Brucker et al. 1999; Kolisch and Padman 2001; Herroelen and Leus 2005) and, more recently, (Hartmann and Briskorn 2010; Koné et al. 2011). The RCPSP is NP-hard in the strong sense (Bartusch, Mohring, and Radermacher 1988; Blazewicz, Lenstra, and Kan 1983). However, many small instances (with up to 50 activities) are usually tractable within a reasonable time.

Many approaches have been considered in order to solve the RCPSP: constraint programming (CP) (Liess and Michelon 2008; Baptiste 2009), Boolean satisfiability (SAT) (Horbach 2010), mixed integer linear programming (MILP) (Koné et al. 2011), branch and bound algorithms (BB) (Dorndorf, Pesch, and Phan-Huy 2000) and others (Debels and Vanhoucke 2007). A hybrid approach using SAT and CP technology (Schutt et al. 2009; 2010) has shown very good results.

Over the last decade there have been important advances in SAT solving techniques, to the point that SAT solvers have become a viable engine for solving constraint satisfaction problems (CSPs) (Walsh 2000; Cadoli, Mancini, and Patrizi 2006; Tamura et al. 2009). In the so-called "lazy clause generation" approach of (Schutt et al. 2009; 2010), a SAT model of a finite domain problem is lazily created as computation progresses, hence allowing the solver to get profit especially from SAT nogood learning.

On the other hand, the SAT solving techniques have been adapted for more expressive logics. For instance, in the case of SMT, the problem is to decide the satisfiability of a formula with respect to a background theory (or combinations of them) in first order logic with equality (Nieuwenhuis, Oliveras, and Tinelli 2006; Sebastiani 2007). Although most SMT solvers are restricted to decidable quantifier free fragments of their logics, this suffices for many applications. There are some promising results in the utilization of SMT technology for solving CSPs, even in the case of combinatorial optimization (Nieuwenhuis and Oliveras 2006; Bofill et al. 2009; Bofill, Suy, and Villaret 2010). Fundamental challenges on SMT for constraint programming and optimization are detailed in (Nieuwenhuis et al. 2007).

In this paper we show that state-of-the-art SMT solvers are a viable engine for solving the RCPSP. We propose some variations of well-known MILP formulations of the RCPSP, which are easily encoded using the SMT formalism with the theory of linear integer arithmetic. Since the RCPSP is an optimization problem there exist at least two natural approaches: on the one hand we can simply iteratively encode the RCPSP instance into SMT instances that bound the optimal makespan, and on the other hand we can encode the RCPSP into a Weighted Max-SMT instance where the soft clauses encode the objective function. We have built a tool, named `rcp2smt`, which solves RCPSP instances using the (Weighted Max-)SMT solver Yices (Dutertre and de Moura 2006). Apart from using the Yices default algo-

[1]This problem is denoted as $PS|temp|C_{max}$ in (Brucker et al. 1999) and $m, 1|gpr|C_{max}$ in (Herroelen and Leus 2005).

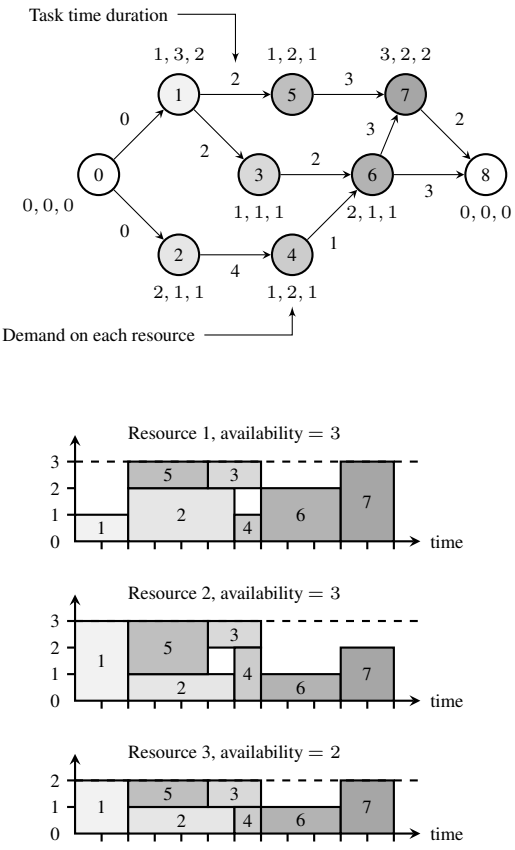

Figure 1: An example of RCPSP (Liess and Michelon 2008)

rithm for solving the Weighted Max-SMT instances we have implemented, through the Yices API, an algorithm based on unsatisfiable cores (Ansótegui, Bonet, and Levy 2009; Manquinho, Silva, and Planes 2009).

The rest of this paper is organized as follows. In the next two sections we briefly introduce the RCPSP and SMT. Then we describe `rcp2smt` with special emphasis in the preprocessing and optimization phases. Next we describe our encodings for the RCPSP using the SMT formalism, and provide performance comparisons between ours and others approaches. We conclude pointing out some future work.

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. Activity A_0 represents by convention the start of the schedule and activity A_{n+1} represents the end of the schedule. The set of non-dummy activities is defined by $A = \{A_1, \ldots, A_n\}$.
- $p \in \mathbb{N}^{n+2}$ is a vector of durations. p_i denotes the duration of activity i, with $p_0 = p_{n+1} = 0$ and $p_i > 0, \forall A_i \in A$.
- 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 the node i to node j means that activity i must precede activity j. We assume that E is such that A_0 is a predecessor of all other activities and A_{n+1} is a successor of 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 resource availabilities. B_k denotes the availability amount of each resource R_k
- $b \in \mathbb{N}^{(n+2)\times m}$ is a matrix of the demands of the activities for resources. $b_{i,k}$ represents the amount of resource R_k used during the execution of A_i. Note that $b_{0,k} = 0$ and $b_{n+1,k} = 0, \forall k \in \{1, \ldots, m\}$.

A schedule is a vector $S = (S_0, S_1, \ldots, S_n, S_{n+1})$ where S_i denotes the start time of each activity $A_i \in V$. We assume that $S_0 = 0$. A solution of the RCPSP problem is a non-preemptive (an activity cannot be interrupted once it is started) schedule S of minimal makespan S_{n+1} subject to the precedence and resource constraints:

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

subject to:

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

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

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

In the example of Figure 1, three resources and seven 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 resources consumptions are indicated in the labels next to the nodes. The upper part of the picture represents therefore the instance to be solved, while the bottom part gives a feasible solution using Gantt charts. For each resource, the horizontal axis represents the time and the vertical axis represents the consumption.

Satisfiability Modulo Theories (SMT)

In the last decades SAT solvers have spectacularly progressed in performance thanks to better implementation techniques and conceptual enhancements such as, for example, non-chronological backtracking and conflict-driven lemma learning, which in many instances of real problems are able to reduce the size of the search space significantly. Thanks to those advances, nowadays best SAT solvers can tackle problems with hundreds of thousands of variables and millions of clauses.

An SMT instance is a generalization of a Boolean formula in which some propositional variables have been replaced by predicates with predefined interpretations from background theories such as, e.g., linear integer arithmetic. For example, a formula can contain clauses like, e.g., $p \vee q \vee (x+2 \leq y) \vee (x > y + z)$, where p and q are Boolean variables and x, y and z are integer variables. Predicates over non-Boolean variables, such as linear integer inequalities, are evaluated according to the rules of a background theory. A Weighted Max-SMT instance is an SMT instance where clauses may have an associated weight of falsification. Such clauses are called soft clauses, whilst the ones without weight are called hard clauses. A solution for these kind of instances must satisfy all hard clauses and minimize the sum of the weights of the falsified clauses.

Leveraging the advances made in SAT solvers in the last decade, SMT solvers have proved to be competitive with classical decision methods in many areas. Most modern SMT solvers integrate a SAT solver with specialized solvers for a set of literals belonging to each theory. It is worthy to notice that most state-of-the-art SMT solvers use the simplex method for dealing with linear integer arithmetic predicates. This way, we can hopefully get the best of both worlds: in particular, the efficiency of the SAT solver for the Boolean reasoning and the efficiency of special-purpose algorithms for the theory reasoning.

Therefore, our approach for RCPSP solving is a *model and solve* approach, that is, instead of looking for ad hoc algorithms, we seek for a modeling that can profit from the efficiency of the SMT solvers. We also look for efficient optimization algorithms on top of the decision procedures, e.g., by using unsatisfiable cores.

RCPSP to SMT

The input to `rcp2smt` is an RCPSP instance in rcp or sch format[2], which is preprocessed in order to build more suitable instances for our solving method (e.g., obtaining better bounds, extending the set of precedences, etc.). After the preprocessing phase, `rcp2smt` searches for an optimal solution as described in the Optimization section.

Preprocessing

The preprocessing phase computes the extended precedence set, a lower and an upper bound for the makespan, time windows for each activity and a matrix of incompatibilities between activities.

Extended precedence set Since a precedence is a transitive relation we can compute the minimum precedence between each pair of activities in E. For this calculation we use the Floyd-Warshall (FW) algorithm $O(n^3)$ on the graph defined by the precedence relation E labeling each arc (A_i, A_j) with the duration p_i. This extended precedence set is named E^* and contains, for all pair of activities A_i and A_j such that A_i precedes A_j, a tuple of the form $(A_i, A_j, l_{i,j})$ where $l_{i,j}$ is the length of the longest path from A_i to A_j. Notice also that, if $(A_i, A_i, l_{i,i}) \in E^*$ for some A_i and $l_{i,i} > 0$, then there is a cycle in the precedence relation and therefore the problem is inconsistent and has no solution.

[2]RCPSP formats from PSPLib (Kolisch and Sprecher 1997).

Lower Bound The lower bound LB is the minimum time we can ensure that the last activity A_{n+1} begins. There are different methods for computing the lower bounds, see (Klein and Scholl 1999; Mingozzi et al. 1998). We have only implemented two of them:

- $LB1$*: Critical path bound.* This is the most obvious lower bound. To compute this, we ignore the capacity restrictions. Then, the minimal project duration is the length of a critical path in the project network. The critical path is the longest path of the graph of precedences between the initial activity A_0 and the final activityA_{n+1}. For instance, in Figure 1 the critical path is $[A_0, A_2, A_4, A_6, A_7, A_8]$ and has length 10. Notice that we can easily know the length of this path if we have already computed the precedence set since we only need to obtain $l_{0,n+1}$ from $(A_0, A_{n+1}, l_{0,n+1}) \in E^*$.
- $LB2$*: Capacity bound.* To compute this bound, we ignore the precedence restrictions. This bound value is the maximum of the total requirement for each resource divided by the capacity of the resource and rounded up to the next integer. The additional cost to compute this lower bound is $O(nm)$ (being n the number of activities and m the number of resources). For instance, in the example of Figure 1 the capacity bound of resource 3 is 11.

$$LB2 = \max\{\lceil (\sum_{A_i \in A} b_{i,k} * p_i)/B_k \rceil \mid B_k \in B\}$$

Finally, we set our computed lower bound to $LB = \max\{LB1, LB2\}$.

Upper Bound The upper bound UB is the maximum time we can ensure that the last activity A_{n+1} begins. We have considered the trivial upper bound for the RCPSP problem:

$$UB = \sum_{A_i \in A} p_i$$

For instance, in the example of Figure 1 the upper bound is 22.

Time windows We can reduce the domain of each variable S_i (start time of activity A_i), that initially is $\{0 \,..\, UB - p_i\}$, by computing its time window. The time window of activity A_i is $[ES_i, LS_i]$, being ES_i the earliest start time and LS_i the latest start time. To compute the time window we use the lower and upper bound and the extended precedence set as follows:

For activities $A_i, 0 \leq i \leq n$,

$$\begin{array}{lcll} ES_i & = & l_{0,i} & (A_0, A_i, l_{0,i}) \in E^* \\ LS_i & = & UB - l_{i,n+1} & (A_i, A_{n+1}, l_{i,n+1}) \in E^* \end{array}$$

and for activity A_{n+1},

$$ES_{n+1} = LB \qquad LS_{n+1} = UB$$

For instance, in the example of Figure 1, activity A_4 has time window $[4, 16]$.

Notice that, if E^* has been successfully computed, then $l_{0,i} \geq 0$ for all $(A_0, A_i, l_{0,i}) \in E^*$ and $l_{i,n+1} \geq p_i$ for all $(A_i, A_{n+1}, l_{i,n+1}) \in E^*$.

Incompatibility We compute a matrix of Booleans I, where each element $I[i,j]$ (noted as $I_{i,j}$) is true if the activity A_i and the activity A_j cannot overlap in time. The incompatibility can occur for two reasons:

- *Precedence.* There exists a precedence constraint between A_i and A_j, i.e., $(A_i, A_j, l_{i,j}) \in E^*$ or $(A_j, A_i, l_{j,i}) \in E^*$.
- *Resources.* For some resource R_k, the sum of the demands of the two activities A_i and A_j, is greater than the resource capacity B_k: $\exists R_k \in R$ s.t. $b_{i,k} + b_{j,k} > B_k$.

For instance, in the example of Figure 1, activity A_4 is incompatible with activities A_0, A_2, A_6, A_7 and A_8 due to the precedences, and with activities A_1, A_5 and A_7 due to resources demands.

This matrix of incompatibilities is symmetric, and the additional cost for its computation is $O(n^2 m)$.

Notice that we could compute incompatibilities due to resource demands between subsets of activities in general (i.e., not restricted to two activities). We have explored this approximation for subsets of size bigger than 2 but the gain of efficiency has been almost zero or even negative, in all encodings and solving methods.

Optimization

SMT has its roots in the field of hardware and software verification, typically dealing with decision problems. For this reason, optimization is not yet supported by most SMT solvers.

There are two approaches we follow to solve RCPSP through SMT solvers. Both approaches share the constraints derived from the preprocessing steps, and the ones from (2) and (3) modelled as the Encodings section describes. The difference consists on how we treat the objective function. On the one hand, we implement an ad hoc search procedure which calls the SMT solver successively constraining the domain of the variable S_{n+1}, by adding a constraint $S_{n+1} \leq bound$ (where $bound$ is a constant and $LB \leq bound \leq UB$) and perform a dichotomic search strategy. Notice that when we get a satisfiable answer, we can eventually refine our upper bound by checking the value of S_{n+1} in the satisfying assignment. We will refer to this method as dico.

On the other hand, some SMT solvers, like Yices, can solve Weighted Max-SMT instances. Weighted Max-SMT allows us to represent optimization problems. We just need to transform the objective function into a set of soft constraints, and keep the rest of the constraints as hard. In order to translate the objective function, we can simply enumerate all its possible values. For example, taking into account the RCPSP in figure 1, where $LB = 11$ and $UB = 22$, we add the following set of soft constraints: $\{(11 \leq S_{n+1}, 1), (12 \leq S_{n+1}, 1), \ldots, (22 \leq S_{n+1}, 1)\}$. Each soft constraint is represented by the pair (C, w) where C is a constraint and w the weight (or cost) of falsifying C. We can obtain a more compact encoding by considering the binary representation of S_{n+1}. Following our example, we would add a new hard constraint, $1{\cdot}b_1{+}2{\cdot}b_2{+}4{\cdot}b_3{+}8{\cdot}b_4{+}LB = S_{n+1}$, where b_i are Boolean variables, and the following set of soft constraints: $\{(\neg b_1, 1), (\neg b_2, 2), (\neg b_3, 4), (\neg b_4, 8)\}$. The resulting instance can be solved by any SMT solver supporting Weighted Max-SMT, like Yices. In our experiments, we refer to the method which uses the non-binary encoding of the objective function as yices.

Since this is yet an immature research topic in SMT, we have extended the Yices framework by incorporating Weighted Max-SAT algorithms. In particular, we have implemented the algorithm WPM1 and WBO from (Ansótegui, Bonet, and Levy 2009; Manquinho, Silva, and Planes 2009), which is based on the detection of unsatisfiable cores. The performance of these algorithms heavily depends on the quality of the cores. Therefore, we have extended them with a heuristic that prioritizes those cores which involve constraints with higher weights. In our experiments, we refer to the method which uses the binary encoding of the objective function and this algorithm as core.

Encodings

SAT modulo linear integer arithmetic allows us to directly express all the constraints of the following encodings, since we can logically combine arithmetic predicates. It is worth noting that in the resulting formulas both Boolean variables and integer variables can occur together. The three encodings we propose are inspired by existing ones, and conveniently adapted to SMT. Moreover, some refinements are introduced considering time windows, incompatibilities, extended precedences. We also add redundant constraints for better propagation.

Since a schedule is a vector $S = (S_0, S_1, \ldots, S_n, S_{n+1})$ where S_i denotes the start time of each activity $A_i \in V$, in all encodings we use a set $\{S_0, S_1, \ldots, S_n, S_{n+1}\}$ of integer variables. By S' we denote the set $\{S_1, \ldots, S_n\}$.

Also, in all encodings the objective function is (1), and we add the following constraints:

$$S_0 = 0 \tag{4}$$
$$S_i \geq ES_i \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{5}$$
$$S_i \leq LS_i \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{6}$$
$$S_j - S_i \geq l_{i,j} \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{7}$$

where (5) and (6) are simple encodings of the time windows for each activity, and (7) encodes the extended precedences.

We also introduce additional constraints for the incompatibilities between activities due to resource capacity constraints:

$$\begin{gathered} S_i + p_i \leq S_j \vee S_j + p_j \leq S_i \\ \forall I_{i,j} \in I \text{ s.t. } I_{i,j} = true, \\ (A_i, A_j, l_{i,j}) \notin E^* \text{ and } (A_j, A_i, l_{j,i}) \notin E^* \end{gathered} \tag{8}$$

Notice that the incompatibilities between activities due to precedence constraints are already encoded in (7).

Time formulation

The more natural encoding for the RCPSP in SMT is the Time formulation, very similar to the MILP encoding proposed by (Pritsker and Wolfe 1996) and referred in (Koné et al. 2011) as Basic discrete-time formulation and in (Schutt et

al. 2009; 2010) as Time-resource decomposition. The idea is that, for every time t and resource R_k, the sum of all resource requirements for the activities must be less than or equal to the resource availabilities.

$$\begin{aligned} ite((S_i \leq t) \wedge \neg(S_i \leq t - p_i); x_{i,t} = 1; x_{i,t} = 0) \\ \forall S_i \in S', \forall t \in \{ES_i, \ldots, LS_i + p_i - 1\} \end{aligned} \tag{9}$$

$$\sum_{A_i \in A} ite(t \in \{ES_i, \ldots, LS_i + p_i - 1\}; b_{i,r} * x_{i,t}; 0) \leq B_r$$

$$\forall B_r \in B, \forall t \in H \tag{10}$$

where $ite(c, e_1, e_2)$ is an *if-then-else* expression denoting e_1 if c is true and e_2 otherwise. Notice that the condition $t \in \{ES_i, \ldots, LS_i + p_i - 1\}$ can be easily encoded into $ES_i \leq t \wedge t \leq LS_i + p_i - 1$.

We remark that (9) imposes that $x_{i,t} = 1$ if the activity A_i is active at time t and $x_{i,t} = 0$ otherwise. We restrict the possible times by using the time windows. Constraints (10) encode the resource constraints (3) using the $x_{i,t}$ variables.

Constraints (9) can be replaced by the equivalent ones:

$$\begin{aligned} y_{i,t} \leftrightarrow (S_i \leq t) \wedge \neg(S_i \leq t - p_i) \\ \forall S_i \in S', \forall t \in \{ES_i, \ldots, LS_i + p_i - 1\} \end{aligned} \tag{11}$$

$$\begin{aligned} ite(y_{i,t}; x_{i,t} = 1; x_{i,t} = 0) \\ \forall S_i \in S', \forall t \in \{ES_i, \ldots, LS_i + p_i - 1\} \end{aligned} \tag{12}$$

We have observed that for small problem instances this formulation gives better performance results than the previous. However, for big instances (with more than 50 activities) the addition of the new variables $y_{i,t}$ usually makes the problem intractable by state-of-the-art SMT solvers.

In order to improve propagation we have considered the following constraints:

$$\begin{aligned} (S_i = t) \rightarrow y_{i,t'} \\ \forall S_i \in S', \forall t \in \{ES_i, \ldots, LS_i + p_i - 1\}, \\ \forall t' \in \{t, \ldots, t + p_i - 1\} \end{aligned} \tag{13}$$

$$\begin{aligned} (S_i = t) \rightarrow \neg y_{i,t'} \\ \forall S_i \in S', \forall t \in \{ES_i, \ldots, LS_i + p_i - 1\}, \\ \forall t' \in \{ES_i, \ldots, LS_i + p_i - 1\} \setminus \{t, \ldots, t + p_i - 1\} \end{aligned} \tag{14}$$

In our experiments, these constraints have shown to provide execution speedup only for small instances.

The following redundant constraints help improving the search time in all instances. This is probably due to the special treatment of those kind of constraints in SMT solvers:

$$x_{i,t} \geq 0 \quad \forall S_i \in S', \forall t \in \{ES_i, \ldots, LS_i + p_i - 1\} \tag{15}$$

$$x_{i,t} \leq 1 \quad \forall S_i \in S', \forall t \in \{ES_i, \ldots, LS_i + p_i - 1\} \tag{16}$$

Task formulation

In this formulation we use variables indexed by activities instead of by time. The key idea is that checking only that there is no overload at the beginning (end) of each activity is sufficient to ensure that there is no overload at every time (for the non-preemptive case). In this formulation, the number of variables and constraints is independent of the length of the scheduling horizon T. This formulation is similar to the Time-resource decomposition of (Schutt et al. 2009) and it is inspired by the encoding proposed in (O. El-Kholy 1999) for temporal and resource reasoning in planning.

$$z^1_{i,j} = true \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{17}$$

$$z^1_{j,i} = false \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{18}$$

$$\begin{aligned} z^1_{i,j} = S_i \leq S_j \\ \forall A_i, A_j \in A, \\ (A_i, A_j, l_{i,j}) \notin E^*, (A_j, A_i, l_{j,i}) \notin E^*, i \neq j \end{aligned} \tag{19}$$

$$z^2_{i,j} = false \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{20}$$

$$z^2_{j,i} = true \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{21}$$

$$\begin{aligned} z^2_{i,j} = S_j < S_i + p_i \\ \forall A_i, A_j \in A, \\ (A_i, A_j, l_{i,j}) \notin E^*, (A_j, A_i, l_{j,i}) \notin E^*, i \neq j \end{aligned} \tag{22}$$

$$z_{i,j} = 0 \qquad \forall I_{i,j} \in I \text{ s.t. } I_{i,j} = true \tag{23}$$

$$\begin{aligned} ite(z^1_{i,j} \wedge z^2_{i,j}; z_{i,j} = 1; z_{i,j} = 0) \\ \forall I_{i,j} \in I \text{ s.t. } I_{i,j} = false, i \neq j \end{aligned} \tag{24}$$

$$\sum_{A_i \in A \setminus \{A_j\}} b_{i,k} * z_{i,j} \leq B_k - b_{j,k} \quad \forall A_j \in A, \forall B_k \in B \tag{25}$$

The Boolean variables $z^1_{i,j}$ are true if activity A_i starts not after A_j and false otherwise. The Boolean variables $z^2_{i,j}$ are true if activity A_j starts before A_i ends and false otherwise. The integer variables $z_{i,j}$ are 1 if activities A_i and A_j overlap and 0 otherwise. The last constraints (25) state that, for every activity A_j and resource R_k, the sum of the resource demands $b_{i,k}$ for R_k from the activities A_i that overlap with A_j should not exceed the capacity B_k of R_k less the demand $b_{j,k}$ for R_k from A_j.

Moreover, in order to improve propagation, we have the following redundant constraints encoding anti-symmetry and transitivity of the precedence relation:

$$z^1_{i,j} \vee z^1_{j,i} \qquad \forall A_i, A_j \in A, i \neq j \tag{26}$$

$$(z^1_{i,j} \wedge z^1_{j,k}) \rightarrow z^1_{i,k} \quad \forall A_i, A_j, A_k \in A, i \neq j, j \neq k, i \neq k \tag{27}$$

The following constraints have also shown to improve propagation in this encoding:

$$z^1_{i,j} \vee z^2_{i,j} \qquad \forall A_i, A_j \in A, i \neq j \tag{28}$$

$$z^1_{i,j} \rightarrow z^2_{j,i} \qquad \forall A_i, A_j \in A, i \neq j \tag{29}$$

Flow formulation

This formulation is inspired by the formulations of (Koné et al. 2011; Artigues, Michelon, and Reusser 2003) named Flow-based continuous-time formulation.

$$y_{i,j} = true \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{30}$$

$$y_{j,i} = false \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{31}$$

$$y_{i,j} = S_j \geq S_i + p_i \quad \forall A_i, A_j \in V, \quad (A_i, A_j, l_{i,j}) \notin E^*, (A_j, A_i, l_{j,i}) \notin E^*, i \neq j \tag{32}$$

$$f_{i,j,k} \geq 0 \qquad \forall A_i, A_j \in V, R_k \in R, i \neq j \tag{33}$$

In the following constraints $b^e_{q,k}$ denotes $b_{q,k}$ for all $A_q \in A$ and denotes B_k for $q = 0$ and $q = n + 1$.

$$f_{i,j,k} \leq \min(b^e_{i,k}, b^e_{j,k}) \quad \forall A_i, A_j \in V, R_k \in R, i \neq j \tag{34}$$

$$y_{i,j} \vee f_{i,j,k} = 0 \qquad \forall A_i, A_j \in V, R_k \in R, i \neq j \tag{35}$$

$$\sum_{A_j \in A \cup \{A_{n+1}\}} f_{i,j,k} = b_{i,k} \quad \forall A_i \in A \cup \{A_0\}, \forall R_k \in R, i \neq j \tag{36}$$

$$\sum_{A_i \in A \cup \{A_0\}} f_{i,j,k} = b_{j,k} \quad \forall A_j \in A \cup \{A_{n+1}\}, \forall R_k \in R, i \neq j \tag{37}$$

The Boolean variables $y_{i,j}$ are true if activity A_j starts after A_i finishes and false otherwise. The integer variables $f_{i,j,k}$ denote the quantity of resource R_k that is transferred from A_i (when finished) to A_j (at the start of its processing). Constraints (33) and (34) fix their range of values. Constraints (35) denote that if A_j does not start after the completion of A_i then the flow from A_i to A_j should be 0 for all resource R_k. These constraints link the $y_{i,j}$ and $f_{i,j,k}$ variables. Constraints (36) and (37) state that, for each activity and resource, the sum of the flows transferred and the sum of the flows received by the activity must be equal to its demand for the resource. Notice no flow enters A_0 and no flow exits from A_{n+1}.

Similarly to the Task formulation, we have the following constraints for anti-symmetry and transitivity of the precedences:

$$\neg y_{i,j} \vee \neg y_{j,i} \qquad \forall A_i, A_j \in A, i \neq j \tag{38}$$

$$(y_{i,j} \wedge y_{j,k}) \rightarrow y_{i,k} \quad \forall A_i, A_j, A_k \in A, i \neq j, j \neq k, i \neq k \tag{39}$$

The constraints for the incompatibilities (8) are reformulated as follows:

$$y_{i,j} \vee y_{j,i} \quad \forall I_{i,j} \in I \text{ s.t. } I_{i,j} = true, \quad (A_i, A_j, l_{i,j}) \notin E^* \text{ and } (A_j, A_i, l_{j,i}) \notin E^* \tag{40}$$

Experiments and conclusions

We have run our experiments on machines with the following specs; operating system: Rocks Cluster 5.2 Linux 2.6.18, processor: AMD Opteron 242 Processor 1.5 GHz, memory: 450 MB, cache: 1024 KB, and compiler: GCC 4.1.2.

The benchmarks we considered are those from the experimental investigation conducted in (Koné et al. 2011). We focus on 4 families of RCPSP instances: KSD30 (KSD instances from the PSPLib with 30 activities), Pack instances from (Carlier and Néron 2003), Pack_d and KS15_d (Koné et al. 2011). We are mainly interested on the two last benchmarks. As pointed out in (Koné et al. 2011), KSD and Pack instances, which can typically be found in the process industry, involve relatively short time horizons. Therefore, the authors of (Koné et al. 2011) generated the Pack_d and KS15_d sets, which involve longer time horizons.

At tables, the numbers in parenthesis indicate the percentage of optimal solutions found, while the others indicate the average execution time of solved instances. The timeout was set to 500 seconds as in (Koné et al. 2011). Notice that the processor used in (Koné et al. 2011) was also of 1.5Ghz.

Table 1 compares presents the three formulation techniques we presented in the Encodings section: Time, Task, and Flow. For each formulation, we applied three different solving approaches based on the Yices SMT solver, which are described in the Optimization section: dico, yices and core. As we can see, for both sets KSD30 and Pack, the Time encoding is the best one while, for those sets involving longer time horizons, KSD15_d and Pack_d, the Task encoding is superior to the rest. We conjecture that Time behaves better when time horizons are smaller, like in the KSD30 and Pack sets, because the number of variables grows significantly with the time horizon. Both Task and Flow formulations are independent of time horizons, however the Flow formulation involves more variables and constraints. Notice that the Task formulation has $O(n^2)$ variables, being n the number of activities, while the Flow formulation has $O(n^2 * m)$ variables, where m is the number of resources. The same complexities apply to the number of constraints.

Table 2 compares our best solving method with other existing approaches. In (Koné et al. 2011) the authors compare several MILP formulations solved with the ILOG Cplex solver against the method of (Laborie 2005) which is based on the detection and resolution of Minimal Critical Sets (MCS) at each node of the search. Among the MILP formulations, we highlight the best performing ones: Disaggregated Discrete-Time (DDT), Flow-based Continuous-Time (FCT) and On/Off Event-based (OOE). Moreover, we compare with the `lazy_fd` solver from (Schutt et al. 2009).

Overall, the SMT based approach is the most robust one. The approach by (Koné et al. 2011) is the best performing one for the Pack set, closely followed by the Time formulation with the dico method, but it is worse on the other sets, especially on the sets with larger horizons: on the KS15_d set, there is a difference of two orders of magnitude in time, and on the Pack_d set, it fails to solve most of the instances. MCS is comparable to the best SMT approach except for the Pack set. Also, the `lazy_fd` solver exhibits very good

		Time			Task			Flow		
set	#	core	dico	yices	core	dico	yices	core	dico	yices
KSD30	480	**4.6(100)**	4.1(99)	7.4(100)	9.7(95)	8.7(96)	8.1(95)	59.2(91)	59.5(92)	33.9(89)
Pack	55	38.9(63)	**59.3(65)**	71.8(65)	69.1(36)	72.4(40)	95.4(41)	143.0(3)	88.4(3)	74.5(5)
KSD15_d	479	8.1(100)	7.7(100)	9.1(100)	0.2(100)	**0.1(100)**	0.5(100)	1.1(99)	1.0(99)	3.7(99)
Pack_d	55	176.4(14)	117.7(12)	212.7(12)	**79.4(43)**	19.8(41)	105.6(40)	108.3(12)	75.7(12)	94.5(12)

Table 1: Comparison of the different proposed solving methods (% solved instances in parenthesis, mean time in seconds, cutoff 500 seconds).

set	#	MCS	(Koné et al. 2011)	lazy_fd	SMT
KSD30	480	7.39(97)	DDT–10.45(82)	**0.49(100)**	core(Time)–4.60(100)
Pack	55	115.88(25)	**DDT–63.39(76)**	94.40(20)	dico(Time)–59.26(65)
KSD15_d	479	0.07(100)	FCT–12.06(94)	**0.04(100)**	dico(Task)–0.13(100)
Pack_d	55	72.34(38)	OOE–75.58(18)	**51.24(61)**	core(Task)–79.37(43)

Table 2: Comparison with other solvers (% solved instances in parenthesis, mean time in seconds, cutoff 500 seconds).

set	#	lazy_fd	SMT
Pack	55	1842(34)	dico(Time)–1030(74)
Pack_d	55	874(69)	core(Task)–2310(61)

Table 3: Scalability comparison (% solved instances in parenthesis, mean time in seconds, cutoff 36000 seconds).

performance in all sets of instances except for the Pack set. Table 3 compares `lazy_fd` with the SMT approach on the Pack and Pack_d sets with a cutoff of 10 hours. As we can see, the SMT approach shows a better scaling behaviour, and globally solves a larger number of instances.

As future work we plan to develop new encodings based on events and study the channeling of different encodings to achieve greater consistency.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science and Innovation through the projects SuRoS (ref. TIN2008-04547), TIN2010-20967-C04-01/03 and TIN2009-14704-C03-01.

References

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2009. Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *LNCS*, 427–440. Springer.

Artigues, C.; Michelon, P.; and Reusser, S. 2003. Insertion Techniques for Static and Dynamic Resource-Constrained Project Scheduling. *European Journal of Operational Research* 149(2):249–267.

Baptiste, P. 2009. Constraint-Based Schedulers, Do They Really Work? In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, volume 5732 of *LNCS*, 1–1. Springer.

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

Blazewicz, J.; Lenstra, J. K.; and Kan, A. H. G. R. 1983. Scheduling Subject to Resource Constraints: Classification and Complexity. *Discrete Applied Mathematics* 5(1):11 – 24.

Bofill, M.; Palahi, M.; Suy, J.; and Villaret, M. 2009. SIMPLY: a Compiler from a CSP Modeling Language to the SMT-LIB Format. In *Proceedings of the 8th International Workshop on Constraint Modelling and Reformulation*, 30–44.

Bofill, M.; Suy, J.; and Villaret, M. 2010. A System for Solving Constraint Satisfaction Problems with SMT. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing*, volume 6175 of *LNCS*, 300–305. Springer.

Brucker, P.; Drexl, A.; Mhring, R.; Neumann, K.; and Pesch, E. 1999. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operational Research* 112(1):3 – 41.

Cadoli, M.; Mancini, T.; and Patrizi, F. 2006. SAT as an Effective Solving Technology for Constraint Problems. In *Proceedings of the 16th International Symposium on Foundations of Intelligent Systems*, volume 4203 of *LNCS*, 540–549. Springer.

Carlier, J., and Néron, E. 2003. On Linear Lower Bounds for the Resource Constrained Project Scheduling Problem. *European Journal of Operational Research* 149(2):314–324.

Debels, D., and Vanhoucke, M. 2007. A Decomposition-Based Genetic Algorithm for the Resource-Constrained Project-Scheduling Problem. *Operations Research* 55(3):457–469.

Dorndorf, U.; Pesch, E.; and Phan-Huy, T. 2000. A Branch-and-Bound Algorithm for the Resource-Constrained Project Scheduling Problem. *Mathematical Methods of Operations Research* 52:413–439.

Dutertre, B., and de Moura, L. 2006. The Yices SMT Solver. Technical report, Computer Science Laboratory, SRI International. Available at `http://yices.csl.sri.com`.

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

Herroelen, W., and Leus, R. 2005. Project Scheduling under Uncertainty: Survey and Research Potentials. *European Journal of Operational Research* 165(2):289–306.

Herroelen, W.; Reyck, B. D.; and Demeulemeester, E. 1998. Resource-Constrained Project Scheduling: A Survey of Recent Developments. *Computers and Operations Research* 25(4):279 – 302.

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

Klein, R., and Scholl, A. 1999. Computing Lower Bounds by Destructive Improvement: An Application to Resource-Constrained Project Scheduling. *European Journal of Operational Research* 112(2):322–346.

Kolisch, R., and Padman, R. 2001. An Integrated Survey of Deterministic Project Scheduling. *Omega* 29(3):249–272.

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

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

Laborie, P. 2005. Complete MCS-Based Search: Application to Resource Constrained Project Scheduling. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 181–186. Professional Book Center.

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

Manquinho, V. M.; Silva, J. P. M.; and Planes, J. 2009. Algorithms for Weighted Boolean Optimization. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *LNCS*, 495–508. Springer.

Mingozzi, A.; Maniezzo, V.; Ricciardelli, S.; and Bianco, L. 1998. An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation. *Management Science* 44:714–729.

Nieuwenhuis, R., and Oliveras, A. 2006. On SAT Modulo Theories and Optimization Problems. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing*, volume 4121 of *LNCS*, 156–169. Springer.

Nieuwenhuis, R.; Oliveras, A.; Rodríguez-Carbonell, E.; and Rubio, A. 2007. Challenges in Satisfiability Modulo Theories. In *Proceedings of the 18th International Conference on Term Rewriting and Applications*, volume 4533 of *LNCS*, 2–18. Springer.

Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2006. Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T). *Journal of the ACM* 53(6):937–977.

O. El-Kholy, A. 1999. *Resource Feasibility in Planning*. Ph.D. Dissertation, Imperial College, University of London.

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

Schutt, A.; Feydy, T.; Stuckey, P. J.; and Wallace, M. 2009. Why Cumulative Decomposition Is Not as Bad as It Sounds. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, volume 5732 of *LNCS*, 746–761. Springer.

Schutt, A.; Feydy, T.; Stuckey, P.; and Wallace, M. 2010. Explaining the Cumulative Propagator. *Constraints* 1–33.

Sebastiani, R. 2007. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation* 3(3-4):141–224.

Tamura, N.; Taga, A.; Kitagawa, S.; and Banbara, M. 2009. Compiling Finite Linear CSP into SAT. *Constraints* 14(2):254–272.

Walsh, T. 2000. SAT v CSP. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming*, volume 1894 of *LNCS*, 441–456. Springer.