Computers & Industrial Engineering 149 (2020) 106777

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

SMT encodings for Resource-Constrained Project Scheduling Problems

Miquel Bofill, Jordi Coll *, Josep Suy, Mateu Villaret

Universitat de Girona, Spain

ARTICLE INFO

Keywords:
Scheduling
RCPSP
MRCPSP
RCPSP/t
SMT
Resource constraints

ABSTRACT

The Resource-Constrained Project Scheduling Problem (RCPSP) is a paradigmatic scheduling problem where the activities of a project have to be scheduled while respecting a combination of precedence and resource constraints. Precedence constraints are relations between two activities stating that one cannot start until the other has ended, and resource constraints bound the amount of resources used by activities running simultaneously. Many generalizations of the RCPSP have been proposed in the literature, including multiple execution modes for the activities (MRCPSP), or time varying resource availabilities and demands (RCPSP/t).

In this work we present Satisfiability Modulo Theories (SMT) formulations to solve the RCPSP, as well as its two variants MRCPSP and RCPSP/t. Although it is really natural to formulate resource constraints of RCPSP-like problems using the linear integer arithmetic (LIA) theory, we show how, by exploiting the information provided by the precedence relations, we can obtain compact and efficient encodings of resource constraints to Boolean Satisfiability (SAT) formulas. Using these SAT encodings instead of the LIA ones, turns to be crucial regarding efficiency. The method is adapted to encode resource constraints for the other two considered variants. In this adaptation, the method exploits not only precedences, but multiple execution modes or time varying resource availabilities and demands. Our experimental results show that the proposed encodings are more efficient than existing state-of-the-art exact solving techniques for the studied problems.

1. Introduction

The Resource Constrained Project Scheduling Problem (RCPSP) is a paradigmatic scheduling problem that can be used to model a large number of industrial applications, and for this reason has been extensively studied and is still nowadays. In the RCPSP, a set of non-preemptive activities have to be scheduled respecting some precedence and resource constraints. Precedence constraints relate pairs of activities by forbidding one to start before the other has been completed. Resource constraints forbid that the amount of resources used by activities running simultaneously exceed the availability of that resource. The typical objective is to minimize the duration of the project (makespan) (Artigues, Demassey, & Neron, 2013). There exist many variants of this problem: considering multiple execution modes of the activities with different durations and resource demands, as well as distinguishing between renewable and non-renewable resources (Multi-mode Resource-Constrained Scheduling Problem, MRCPSP); considering minimum and maximum time lags between start-times of pairs of activities (RCPSP/max), possibly with multiple execution modes (MRCPSP/max); considering time-varying resource availabilities and demands (Resource-Constrained Project Scheduling Problem with Time-Dependent Resource Capacities and Requests, RCPSP/t), or considering skills associated to resources and activities (Multi-skill Project Scheduling Problem, MSPSP). We refer the reader to Hartmann and Briskorn (2010) for a complete survey on different variations and extensions of the RCPSP.

The RCPSP (and so any of its generalizations) is NP-hard in the strong sense (Blazewicz, Lenstra, & Kan, 1983). Finding a schedule for the RCPSP with no upper bound on the makespan of the project can be done in polynomial time with a greedy algorithm (Kelley, 1963; Kolisch, 1996). However, this is not the case when there exist multiple execution modes and non-renewable resources (MRCPSP), since determining whether there is a combination of modes that satisfies non-renewable resource constraints is already NP-hard (Kolisch & Drexl, 1997).

Due to the hardness of most of the scheduling problems, the majority of efforts have been devoted to provide meta-heuristic methods. A recent and extensive survey on meta-heuristics for the RCPSP can be found in Pellerin, Perrier, and Berthaut (2019). Nevertheless, there have been successful exact approaches to solve reasonably sized instances. For many years, the leading exact approach to scheduling problems has been to solve them by translation to 0/1 Mixed Integer Linear Programming (MILP), i.e., to MILP formulations where

* Corresponding author.
E-mail address: jordi.coll@imae.udg.edu (J. Coll).

https://doi.org/10.1016/j.cie.2020.106777
Received 10 February 2020; Received in revised form 29 July 2020; Accepted 17 August 2020
Available online 22 August 2020

the variables can only take the values 0 or 1. We can find the first MILP formulations in the 1970s (Patterson & Huber, 1974; Stinson, Davis, & Khumawala, 1978; Talbot & Patterson, 1978), and still in the last decade there have been many publications on MILP models for RCPSP-based problems (Bellenguez-Morineau, 2008; Koné, Artigues, Lopez, & Mongeau, 2011; Toffolo, Santos, Carvalho, & Soares, 2016). However, alternative exact methods such as Satisfiability Modulo Theories (SMT) (Ansótegui, Bofill, Palahí, Suy, & Villaret, 2011; Bofill, Coll, Suy, & Villaret, 2016, 2017a, 2017b) or Constraint Programming (CP) (Schnell & Hartl, 2016; Schutt, Feydy, & Stuckey, 2013; Schutt, Feydy, Stuckey, & Wallace, 2013; Szeredi & Schutt, 2016; Vilím, 2011; Vilím, Laborie, & Shaw, 2015) have become state-of-the-art in solving RCPSP and variants. On the one side, CP-based approaches typically provide specialized propagators to deal with resource constraints while, on the other side, SMT approaches focus on finding efficient encodings. Regarding SMT, in Bofill et al. (2016) it was shown the convenience of solving resource constraints of the MRCPSP with Boolean Satisfiability (SAT) encodings of pseudo-Boolean (PB) constraints, and dealing with the precedence relations with Integer Difference Logic (IDL) expressions. Later on, in Bofill et al. (2017a) it was presented a method based on Multi-valued Decision Diagrams (MDD) to compactly encode PB constraints to SAT when there appear collateral at-most-one (AMO) constraints in the formulation at hand. This method was successfully applied to model resource constraints in the MRCPSP and the RCPSP/t.

This work collects, standardizes and extends the contributions on SMT formulations of scheduling problems from Bofill et al. (2016) and Bofill et al. (2017a). More precisely, we present SMT formulations for the RCPSP, the MRCPSP (Brucker, Drexl, Möhring, Neumann, & Pesch, 1999) and the RCPSP/t (Hartmann, 2013). For the two first problems we present a time-indexed formulation and a task-indexed formulation, while only a time-indexed formulation is presented for the RCPSP/t, since this problem is inherently related to time-indexed formulations. These formulations always express precedence constraints with the computationally easy SMT theory of IDL. On the other hand, resource constraints are modeled as PB constraints, that can be encoded to SMT in different ways. In particular we consider, and experimentally evaluate, three techniques to encode PB constraints. The two first are well known in the literature: using Linear Integer Arithmetic (LIA) expressions, and using SAT encodings. The third one is a recent encoding approach which takes into account implied AMO constraints in the formulation at hand to obtain compact and efficient SAT encodings of PB constraints (Bofill et al., 2017a; Bofill, Coll, Suy, & Villaret, 2019). When referring to this approach will talk about encoding PB constraints "modulo AMO relations" or, more succinctly, about encoding PB(AMO) constraints. A significant contribution of this paper is to provide a specific way to detect large AMO constraints in RCPSP-like problems, which is a crucial aspect in the encoding of PB(AMO) constraints. It is important to note that the AMO relations on the variables of the PB constraint can be either explicit (i.e., stated explicitly by some AMO constraints in the formulation) or implicit (i.e., implied by other constraints).

As stated before, this work collects some of the contributions of the previous works (Bofill et al., 2016, 2017a). The following are the new contributions of this work:

- A task-indexed SMT formulation, and experiments, for the RCPSP and the MRCPSP, and a discussion on the unsuitability of task-indexed formulations for the RCPSP/t.
- A performance evaluation of LIA encodings of resource constraints in the RCPSP and the RCPSP/t, compared to SAT encodings.
- Detailed explanations of the process of detecting implicit AMO constraints over Boolean variables in time-indexed formulations of RCPSP-like problems.
- A process to detect implicit AMO constraints over Boolean variables in task-indexed formulations of RCPSP-like problems.
- Experiments on additional datasets, namely all the experiments for the RCPSP, and also the experiments for the MMLIB100 set of the MRCPSP.
- An evaluation of the three new SAT encodings of PB(AMO) constraints presented in Bofill et al. (2019). Reporting all these results would mean a huge amount of data. Therefore the tables only detail results for MDD-based encodings, which are in most cases the best. There is a paragraph in the experimental section devoted to discuss the results with other PB(AMO) encodings.

The remainder of this paper is structured as follows:

- In Section 3 we summarize the technique of encoding PB(AMO) constraints, that is, encoding PB constraints taking into account collateral AMO constraints.
- In Section 4 we introduce the three scheduling problems studied in this paper (the RCPSP, the MRCPSP and the RCPSP/t) and review a series of common preprocessing techniques that we apply in our systems.
- In Section 5 we present time-indexed and task-indexed formulations for the decision versions of the RCPSP and the MRCPSP, and a time-indexed formulation for the decision version of the RCPSP/t.
- In Section 6 we introduce a technique to detect AMO constraints between the Boolean variables used in resource constraints, both in time-indexed and task-indexed formulations. We present a basic method for the RCPSP that could be easily adapted to most RCPSP extensions. We also adapt this method to the cases of the MRCPSP and the RCPSP/t.
- In Section 7 we explain the optimization algorithms implemented in our systems, which use an SMT solver as an oracle to solve encodings of decision versions of the scheduling problems.
- In Section 8 we evaluate the performance of our systems with extensive experiments with different settings. On the one hand, we compare the time-indexed and task-indexed formulations. On the other hand, we evaluate the following strategies for encoding resource constraints: LIA expressions, PB SAT encodings and PB(AMO) SAT encodings. Finally, we compare the performance of our systems with state-of-the-art solvers for the studied problems, and we show that we can solve more instances in many families.
- In Section 9 we present the conclusions of our work.

2. Related work

In this section we review the principal formulation viewpoints for the RCPSP and its extensions, as well as the state-of-the-art exact approaches to solve these problems.

2.1. Formulations

Among the different viewpoints that have been used to model the RCPSP, we distinguish four major kinds of formulations: *time-indexed*, *task-indexed*, *flow-based*, *event-based*.

Time-indexed. These formulations discretize the period of time in which the schedule will take place into unit intervals from 0 to a given upper bound on the makespan. Then, to ensure that the capacity of a resource is never exceeded, a resource constraint is added for each resource and discrete time instant. Also time-indexed variables are introduced for each activity and time instant, in order to check whether an activity is running at a certain time instant. For further details and precise examples, we refer the reader to Artigues (2013), Schutt, Feydy, Stuckey, and Wallace (2009) and Suy (2013).

Task-indexed. These formulations are based on the fact that the occupancy of a resource is only increased when an activity starts. Therefore, it is enough to guarantee that resource capacities are not exceeded when activities start. To this purpose, the formulation introduces variables representing whether an activity is running when another activity starts. For further details and precise examples, we refer the reader to Schutt et al. (2009) and Suy (2013).

Flow-based. These formulations are based on modeling resource availabilities as finite resources that activities transfer to forthcoming activities once they finish, creating this way a flow of resources. Therefore, the formulation of resource constraints enforces that the flow is valid (i.e., activities transfer to other activities the exact resource amounts that they have received), and that each activity receives the required quantity of each resource. For further details and precise examples, we refer the reader to Koné et al. (2011) and Suy (2013).

Event-based. These formulations define a sequence of events that occur during the schedule, map every activity to some events, and only check the resource consumptions at these events. For instance, in start/end event formulations, a set of variables represents a sequence of start times and a sequence of end times (events). Then, other variables represent which is the start event and the end event of each activity. Finally, constraints must ensure that the distance between events is coherent with activity durations and precedence relations, and that resources can supply the total demand at each event. For further details and precise examples, we refer the reader to Koné et al. (2011) and Suy (2013).

Some of these formulations admit continuous (real) variables instead of discrete (integer) variables to denote start/end times of activities or events. Continuous variables have frequently used in MILP (simplex-based) approaches. However, technologies based on search and propagation on finite-domains such as CP and SMT mostly use discrete variables in scheduling models and formulations (Ansótegui et al., 2011; Roselli, Bengtsson, & Åkesson, 2018; Schnell & Hartl, 2016; Vilím et al., 2015).

2.2. CP

Apart from the SMT works that we revisit in this paper, to the best of our knowledge, the technologies that have provided the best results in the last decade for exact solving of the RCPSP and the extensions that we consider are CP-based.

The system presented in Vilím et al. (2015) implements a search procedure for CP in the *IBM ILOG CPOptimizer* which is specially designed for scheduling problems. The authors name this technique *Failure Directed Search* (FDS), and it consists in prioritizing the exploration of the configurations that will lead to a conflict. In that paper there were considered many extensions of the RCPSP, more precisely MRCPSP, RCPSP/max and MRCPSP/max.

Lazy Clause Generation (LCG) (Feydy & Stuckey, 2009) has also shown to be very competitive in scheduling problems. LCG solvers are based on CP solving techniques, assisted by a SAT solver, and using specialized propagators for global constraints. The system presented in Schutt, Feydy, and Stuckey (2013) provides *time-table-edge-finding* propagator for cumulative constraints in LCG, and reports very good results in solving the RCPSP. Also, the system presented in Szeredi and Schutt (2016) consists of a CP model for the MRCPSP which uses the LCG solver *Chuffed*. The model handles resource constraints with the *cumulative* global constraint, and a specialized search heuristic is proposed.

2.3. SMT

The research on SAT solving has provided a series of algorithms and techniques that make modern SAT solvers a competitive tool for solving certain hard combinatorial problems. Among the most relevant we can highlight the introduction of clause learning during search (Marques-Silva & Sakallah, 1999), the use of conflict-based branching heuristics and efficient implementations of unit-propagation (Moskewicz, Madigan, Zhao, Zhang, & Malik, 2001), the literal block distance metric to remove unused learnt clauses (Audemard & Simon, 2009), and preprocessing and in-processing techniques of reformulation and clause vivification (Fazekas, Biere, & Scholl, 2019; Li, et al., 2020). SMT solvers benefit from all these techniques thanks to their core SAT solver, and provide specialized and efficient algorithms to deal with background theories such as integer arithmetic (Dutertre & De Moura, 2006; Nieuwenhuis & Oliveras, 2005).

Thanks to its good performance, we can find several successful applications of SMT to scheduling problems. For instance, in Cheng, Zhang, Tan, and Lim (2016) an SMT encoding is used to solve the problem of scheduling processes with deadlines in a multiprocessor system. Also in the field of parallel computation, in Malik, Walker, O'Sullivan, and Sinnen (2018) there was presented an SMT model for a different task scheduling problem. In Erkinger and Musliu (2017), a personnel scheduling problem is solved using the bit-vector theory of SMT. In Roselli et al. (2018), three SMT models are given for the job-shop scheduling problem, and the conducted experiments show that SMT is competitive with MILP. As far as we know, besides the papers that we extend here (Bofill et al., 2016, 2017a), the only works that tackle RCPSP using SMT are Ansótegui et al. (2011) and Suy (2013). These works present several SMT formulations which differ in how resource constraints are dealt with. The most successful viewpoints are time-indexed and task-indexed resource constraints using the LIA theory. Two more alternatives that had shown to be competitive in MILP formulations are also studied in these works, namely flow-based and event-based formulations, proving experimentally that they are extremely inefficient with SMT. Therefore, based on these results, in the current work we only consider time-indexed and task-indexed formulations, not only using LIA expressions to deal with resource constraints as done in Ansótegui et al. (2011) and Suy (2013), but also SAT encodings of PB constraints.

3. Encoding pseudo-Boolean constraints

In this section we recall some standard nomenclature and show how to encode pseudo-Boolean constraints into SAT and SMT. We focus on encoding them into SAT by taking profit of collateral at-most-one constraints. In Section 6 we will show how to find such collateral constraints in formulations of RCPSP-like problems.

A *SAT formula* is usually represented as a CNF or conjunction of clauses. Clauses are disjunctions of literals and literals are Boolean variables or their negation. For instance, the CNF $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_4) \wedge (x_1 \vee x_2)$ has three clauses, four Boolean variables, the positive literals x_1, x_2, x_3 and x_4 and the negative literals $\overline{x_1}$ and $\overline{x_2}$. An *SMT formula*(for further details) is a SAT formula where literals can also be atoms from some background theory like, for instance, Linear Integer Arithmetic (LIA), e.g., $((a = 3) \vee \overline{x_2} \vee (-2a + 3b \leq c)) \wedge (\overline{a = 3} \vee x_4) \wedge ((a = 3) \vee x_2)$, where a, b and c are integer variables, and we are considering atoms from the LIA theory like $a = 3$ and $-2a+3b \leq c$. More precisely, the LIA theory supports atoms of the form $\sum_{i=1}^{n} q_i y_i \# K$ where K and all q_i are integer constants, all y_i are integer variables, and $\# \in \{<, \leq, =, \geq, >\}$ (a detailed description of SMT can be found in Barrett, Fontaine, & Tinelli, 2016; Barrett & Tinelli, 2018). An *assignment* is a mapping of Boolean variables to truth values and of variables of the theory (e.g. integer variables) to appropriately typed values (e.g. integers). Given an assignment, atoms are evaluated according to the theory (e.g., if the assignment maps a to 2, the atom $a = 3$ is evaluated to

false). A positive literal evaluates to true if it is a theory atom evaluated to true or if it is a Boolean variable mapped to true in the assignment (conversely for negated literals). A clause evaluates to true if some of its literals is evaluated to true and, finally, a CNF evaluates to true if all its clauses are evaluated to true. By $A \models F$ we denote that the assignment A makes the formula F evaluate to true. In such a case, we also say that A *satisfies* the formula F. Given two formulas F and G, we say that G is a *logical consequence* of F (or that it is *implied* by F), written $F \models G$, iff every assignment satisfying F also satisfies G. We refer the reader to Biere, Heule, and van Maaren (2009) for further details on SAT and SMT.

A *pseudo-Boolean* (PB) constraint is a Boolean function of the form $\sum_{i=1}^{n} q_i x_i \# K$ where K and all q_i are integer constants, all x_i are 0/1 (false/true) variables, and $\# \in \{<, \leq, =, \geq, >\}$. It is usually assumed that $\#$ is $\leq$, and that K and all q_i are non-negative, since the other cases can be easily reduced to this one (Eén & Sorensson, 2006). A particular case of PB constraints are *at-most-one* (AMO) constraints, which are Boolean functions of the form $\sum_{i=1}^{n} x_i \leq 1$, where all x_i are 0/1 variables.

There are many ways to encode PB constraints into SMT. An almost direct one is introducing a unary clause with a LIA atom of the form: $\sum_{i=1}^{n} q_i y_i \# K$ where y_i are integer variables, and introducing also the clauses $x_i \rightarrow (y_i = 1)$ and $\overline{x_i} \rightarrow (y_i = 0)$ (we recall that a formula $A \rightarrow B$ is equivalent to $\overline{A} \vee B$). Another way is to encode the PB constraint into plain SAT. There exist several possible procedures to encode PB constraints to SAT, based on Binary Decision Diagrams (BDD) (Abío, Nieuwenhuis, Oliveras, Rodríguez-Carbonell, & Mayer-Eichberger, 2012; Eén & Sorensson, 2006), on Generalized Totalizers (Joshi, Martins, & Manquinho, 2015), and many others (Bailleux, Boufkhad, & Roussel, 2009; Hölldobler, Manthey, & Steinke, 2012; Manthey, Philipp, & Steinke, 2014). All these procedures introduce a number of auxiliary Boolean variables and encode the constraint with a set of clauses. The amount of auxiliary variables as well as the size and propagation properties of the set of clauses usually have an impact over the efficiency of the encoding, i.e., on the solving time.

The previous encoding procedures can be improved by considering collateral AMO constraints (Bofill et al., 2017a, 2019). These works report huge decreases in the number of clauses and auxiliary variables in the encodings of PB constraints, as well as decreases in the solving times w.r.t. other encodings that do not take AMO constraints into account. Fortunately, in our scheduling formulations there appear many explicit and implicit AMO constraints. The idea of how to take profit of these AMO constraints to improve the encoding of the PB constraints is the following: the SAT encodings of the PB constraints forbids all possible assignments of Boolean variables that would falsify the PB constraint; it is likely that collateral AMO constraints already forbid some of these assignments that would falsify the PB constraint; therefore, when encoding the PB constraint there is no need to care again about these falsifying assignments and one can save auxiliary variables and clauses. When we encode a PB constraint taking into account collateral AMO constraints, we say that we are encoding a *PB(AMO) constraint*. A PB(AMO) constraint is characterized by a PB constraint and a partition of the variables of the PB constraint. The encoding procedure of a PB(AMO) constraint will assume that, in the problem formulation at hand, there exist other constraints explicitly or implicitly forbidding that more than one variable of a same set of the partition are assigned to true, and hence such assignments can be safely ignored. Example 1 illustrates the benefits of PB(AMO) encodings.

Example 1. Let us assume that we have a formulation with these three constraints

$$P \quad \wedge \quad C_1 \quad \wedge \quad C_2$$

where P is the PB constraint $2x_1 + 3x_2 + 3x_3 + 4x_4 \leq 6$ and constraint $C_1 \models x_1 + x_2 \leq 1$, and constraint $C_2 \models x_3 + x_4 \leq 1$.

We could encode the PB constraint P and obtain the following equisatisfiable formulation:

$$(\overline{x_2} \vee \overline{x_4}) \wedge (\overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \quad \wedge \quad C_1 \quad \wedge \quad C_2$$

Notice however that any assignment not satisfying $x_1 + x_2 \leq 1 \wedge x_3 + x_4 \leq 1$, will neither satisfy $C_1 \wedge C_2$ and, therefore, will not satisfy the conjunction of the three constraints. Therefore, instead of encoding the PB constraint P, we can focus only on the assignments satisfying $x_1 + x_2 \leq 1 \wedge x_3 + x_4 \leq 1$ and, we can safely encode the PB(AMO) constraint characterized by P with partition $\{\{x_1, x_2\}, \{x_3, x_4\}\}$. In this case, the only assignment satisfying $x_1 + x_2 \leq 1 \wedge x_3 + x_4 \leq 1$ but not satisfying P is the one mapping x_1 to false, x_2 to true, x_3 to false and x_4 to true. Therefore, the following formulation would be equisatisfiable to the original conjunction of the three constraints:

$$(\overline{x_2} \vee \overline{x_4}) \quad \wedge \quad C_1 \quad \wedge \quad C_2$$

4. Problem definitions

In this section we define the RCPSP, and its extensions MRCPSP and RCPSP/t. Tables 1–3 summarize the nomenclature used in this paper. After that, we list some well-known preprocessing techniques that we use in our system.

4.1. RCPSP

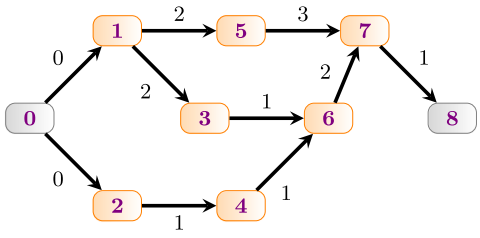
The goal of the *Resource-Constrained Project Scheduling Problem* (RCPSP) is to find a start time for each one of the activities of a project such that the makespan is minimized (Artigues et al., 2013). The makespan is defined as the duration of the project. The activities have a predefined duration and are non-preemptive, i.e., they have to be executed without interruptions. Also, there is a set of requested end-start precedences between pairs of activities that have to be respected, meaning that the successor cannot start until the predecessor has ended. Finally, there is a set of renewable resources with finite capacity, and every activity has a certain demand on each resource while it is running. The total demand on a resource at any particular time cannot be greater than the given resource capacity.

The RCPSP can be formally defined by a tuple (V, p, E, R, B, b) where:

- $V = \{0, 1, \ldots, n, n+1\}$ is a set of activities. Activities 0 and $n+1$ are dummy activities introduced by convention, which represent the start and the end of the schedule, respectively. The set of non-dummy activities is defined by $A = \{1, \ldots, n\}$.
- $p \in \mathbb{N}^{n+2}$ is a vector of naturals, where p_i is the duration of activity i. For the dummy activities we have $p_0 = p_{n+1} = 0$, and $p_i > 0 \ \forall i \in A$.
- E is a set of pairs of activities, representing end-start precedence relations. Specifically, $(i, j) \in E$ iff the execution of activity i must precede that of activity j, i.e., activity i must finish before activity j starts. We will assume that we are given an activity-on-node precedence graph $G = (V, E)$ that contains no cycles, since otherwise the precedence relation would be inconsistent. By convention, there is a path from activity 0 to each other activity, and also a path from each activity to activity $n+1$.
- $R = \{1, \ldots, v\}$ is a set of renewable resources.
- $B \in \mathbb{N}^v$ is a vector of naturals, where B_k is the available amount of each resource k.
- $b \in \mathbb{N}^{(n+2)\times v}$ is a matrix of naturals corresponding to the resource demands of activities, where $b_{i,k}$ represents the amount of resource k that activity i demands per time step during its execution, with $b_{0,k} = 0$, $b_{n+1,k} = 0$ and $b_{i,k} \geq 0, \forall k \in 1..v, \forall i \in 1..n$.

A schedule is a vector of naturals $S = (S_0, S_1, \ldots, S_n, S_{n+1})$ where S_i denotes the start time of activity i. We require w.l.o.g. that $S_0 = 0$. A solution of the RCPSP is a schedule S of minimal makespan S_{n+1} subject to:

Precedence graph:



Activity durations and demands (left), resource capacities (right):

	0	1	2	3	4	5	6	7	8
p_i	0	2	1	1	1	3	2	1	0
$b_{i,1}$	0	3	3	2	1	2	0	3	0
$b_{i,2}$	0	2	2	4	3	1	2	2	0

B_1	B_2
3	4

Fig. 1. Example of an RCPSP instance with 7 (non-dummy) activities and 2 resources.

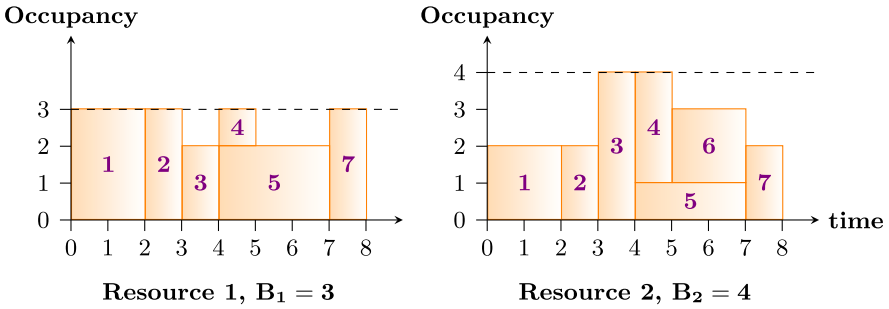


Fig. 2. An optimal solution for the instance of Fig. 1, with schedule $S = (0, 0, 2, 3, 4, 4, 5, 7, 8)$. The capacity of the resources is never exceeded, and the precedence relations are respected.

- Precedence constraints: every activity must start after all its predecessors have finished.
- Renewable resource constraints: the capacity of a resource cannot be surpassed at any time, by the sum of the demands of the activities running at that time.

Fig. 1 depicts an example of an RCPSP instance with 7 (non-dummy) activities and 2 resources, and Fig. 2 shows an optimal solution for it.

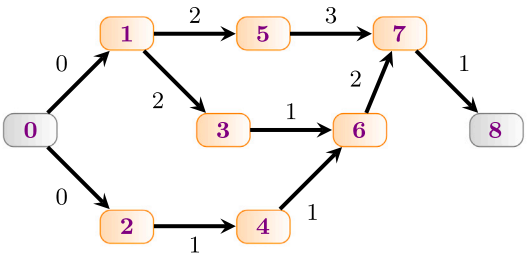
4.2. MRCPSP

The *Multi-mode Resource-Constrained Project Scheduling Problem* (MRCPSP) (Brucker et al., 1999) can be defined by a tuple (V, M, p, E, R, B, b). This problem generalizes the RCPSP by defining a number of execution modes M_i for each activity i. The duration and resource demands of the activities are dependent on the selected execution mode. Also, there can be a number of non-renewable resources (in addition to the renewable resources). Summing up, the added or modified elements w.r.t. the RCPSP are M, p, R, B and b:

- $M \in \mathbb{N}^{n+2}$ is a vector of naturals, where M_i is the number of execution modes of activity i, with $M_0 = M_{n+1} = 1$ and $M_i \geq 1\ \forall i \in A$.
- p is a vector of vectors of naturals, with $p_{i,o}$ being the duration of activity i using mode o. For the dummy activities, $p_{0,1} = p_{n+1,1} = 0$, and $p_{i,o} > 0, \forall i \in A, \forall o \in 1..M_i$.
- $R = \{1, \ldots, v-1, v, v+1, \ldots, q\}$ is a set of resources. The first v resources are renewable, included in the set $RR = \{1, \ldots, v\}$. The last $q - v$ resources are non-renewable, included in the set $NR = \{v+1, \ldots, q\}$.
- $B \in \mathbb{N}^q$ is a vector of naturals, with B_k being the available amount of each resource k. The first v resource availabilities correspond to the renewable resources, while the last $q - v$ correspond to the non-renewable resources.
- b is a three-dimensional matrix of naturals corresponding to the resource demands of activities per mode: $b_{i,k,o}$ represents the amount of resource k used during the execution of activity i in mode o. We have $b_{0,k,1} = 0$, $b_{n+1,k,1} = 0 \forall k \in R$, and we have $b_{i,k,o} \geq 0$, $\forall k \in R$, $\forall i \in A$, $\forall o \in 1..M_i$.

A schedule is a vector of naturals $S = (S_0, S_1, \ldots, S_n, S_{n+1})$ where S_i denotes the start time of activity i. Again, we require w.l.o.g. that $S_0 = 0$. A schedule of modes is a vector of naturals $SM = (SM_0, SM_1, \ldots, SM_n, SM_{n+1})$ where SM_i, with $1 \leq SM_i \leq M_i$, denotes the selected execution mode for each activity i. A solution of the MRCPSP is a schedule of modes SM and a schedule S of minimal makespan S_{n+1}, subject to precedence constraints, renewable resource constraints, and non-renewable resource constraints. Non-renewable resource capacities are not only demanded while an activity is running, but their capacity is permanently decreased by each demand. An example of a non-renewable resource is a budget. Non-renewable resource constraints require that the capacity of non-renewable resources must be enough to supply the demands of the activities, according to the selected execution modes.

Precedence graph:



Activity durations and demands:

	0	1	2	3	4	5	6	7	8
p_i	0	2	1	1	1	3	2	1	0
$b_{i,1,0}$	-	4	1	2	2	2	1	2	-
$b_{i,1,1}$	-	3	-	-	-	3	3	-	-
$b_{i,1,2}$	-	-	-	-	-	1	-	-	-

Resource capacities:

$B_{1,0}$	$B_{1,1}$	$B_{1,2}$	$B_{1,3}$	$B_{1,4}$	$B_{1,5}$	$B_{1,6}$	$B_{1,7}$	$B_{1,8}$	$B_{1,9}$
5	4	3	3	3	4	2	4	5	5

Fig. 3. Example of an RCPSP/t instance with 7 (non-dummy) activities, 1 resource and $UB = 10$.

4.3. RCPSP/t

The *Resource-Constrained Project Scheduling Problem with Time-Dependent Resource Capacities and Requests* (RCPSP/t) (Hartmann, 2013) can be defined by a tuple (V, UB, p, E, R, B, b). This problem generalizes the RCPSP by defining varying resource demands over the execution of the activities, and varying resource capacities over the execution of the schedule. The modified elements w.r.t. the RCPSP are B and b:

- $B \in \mathbb{N}^{v \times UB}$ is a matrix of naturals, with $B_{k,t}$ being the available amount of resource k at time t. UB is an upper bound of the makespan, that is part of the definition of the instance. The resources can have a different availability for each time instant in $0..UB-1$, and they have an undefined availability in any other time outside this scheduling horizon.
- b is a three-dimensional matrix of naturals corresponding to the resource demands of activities over their execution: $b_{i,k,e}$ represents the amount of resource k that activity i will demand during the eth time instant relative to its start time. Therefore, an activity i has a vector of demands for resource k defined by $b_{i,k,0}, \ldots, b_{i,k,p_i-1}$. As usual, all demands are non-negative and dummy activities do not have resource demands.

Similarly to the RCPSP, a solution for the RCPSP/t is a schedule $S = (S_0, S_1, \ldots, S_n, S_{n+1})$ that satisfies the precedences and the (time-dependent) renewable resource constraints.

Fig. 3 illustrates an example instance of the RCPSP/t with 7 non-dummy activities and 1 renewable resource, and Fig. 4 contains an optimal solution for this instance.

4.4. Preprocessing

We summarize some of the functions that we will compute before generating the encoding of each particular problem. The information

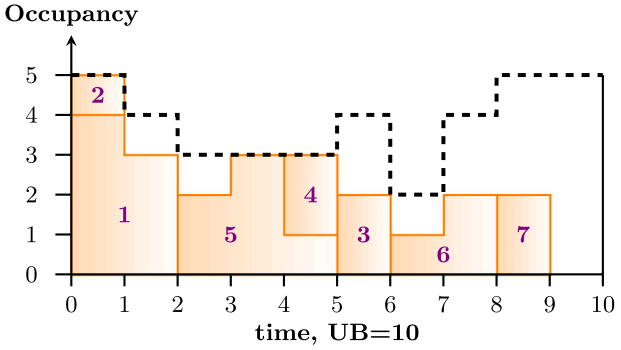


Fig. 4. An optimal solution for the instance of Fig. 3, with schedule $S = (0, 0, 0, 5, 4, 2, 6, 8, 9)$. The capacity of the resource is never exceeded, and the precedence relations are respected.

obtained from these functions will allow us to obtain a more compact encoding saving auxiliary variables and clauses. All these preprocesses are well known in the scheduling literature, see for instance (Brucker & Knust, 2012), although some had been adapted for the considered problems.

4.4.1. Extended precedence relations

It is very common when solving scheduling problems to compute the transitive closure of the given activity precedence graph, in order to obtain an extended precedence graph. We will denote by $G^* = (V, E^*)$ the extended precedence graph, where E^* is a set of weighted edges. There will be an edge $(i, j, l_{i,j}) \in E^*$ for every pair of activities (i, j) in V which are connected by a path starting at activity i and ending at activity j. The time lag $l_{i,j}$ will be a lower bound on the difference between the start times of activities i and j. This time lag $l_{i,j}$ will

correspond to the weight of the critical path from activity i to activity j in G, i.e., the minimum weight (sum of durations) of any path going from activity i to activity j in G. Time lags can be easily computed with the Floyd–Warshall algorithm. In the case of the MRCPSP, where each activity has different possible durations, we consider the minimum durations when computing critical paths.

4.4.2. Energetic reasoning on precedences

The demands on renewable resources in the RCPSP let us go one step further when computing lower bounds on the minimum time lags between activities. Note that all activities a such that $(i, a, l_{i,a}) \in E^* \wedge (a, j, l_{a,j}) \in E^*$ will be completely executed inside the time interval $[S_i + p_i, S_j - 1]$. Hence, this interval must be wide enough to run all such activities a without exceeding the availability of any renewable resource. In other words, for each resource $k \in RR$, a lower bound on the distance between the end time of activity i and the start time of activity j is:

$$RLB_{i,j,k} = \left\lceil \frac{1}{B_k} \cdot \sum_{\substack{a \in A \ s.t. \\ (i,a,l_{i,a}) \in E^* \\ (a,j,l_{a,j}) \in E^*}} (p_a \cdot b_{a,k}) \right\rceil \quad (1)$$

Therefore, it is sound to update the extended precedences with the following new time lags:

$$l'_{i,j} = \max(l_{i,j},\ p_i + \max_{k \in RR}(RLB_{i,j,k})) \quad \forall (i, j, l_{i,j}) \in E^* \quad (2)$$

Note that an increase on a single time lag can be propagated to other time lags in E^*. We can achieve this propagation with only one new execution of the Floyd–Warshall algorithm.

This preprocess resembles the energy based reasoning used by some constraint propagators (see, e.g., Artigues et al., 2013). It can be easily adapted to the multi-mode case by considering the minimum possible consumptions $p_{i,o} \cdot b_{i,k,o}$ over modes in Eq. (1), and minimum durations $p_{i,o}$ in Eq. (2). It can also be adapted to the RCPSP/t by considering the sum of each demand $b_{i,k,t}$ instead of the product $p_i \cdot b_{i,k}$ and the highest possible capacity $B_{k,t}$ over time in Eq. (1).

4.4.3. Time windows

In the RCPSP, as well as in the RCPCP/t, given the extended precedence relations, it is possible to precompute a lower bound on the start time and the end time of each activity, that is, the *earliest start time* and the *earliest close time*:

$$ES(i) = l_{0,i}$$
$$EC(i) = l_{0,i} + p_i$$

Similarly, we can precompute an upper bound on the start and close times if we are given an upper bound for the makespan UB (*latest start time* and *latest close time*):

$$LS(i) = UB - l_{i,n+1}$$
$$LC(i) = UB - l_{i,n+i} + p_i$$

As particular cases for the dummy activities, we have that $l_{0,0} = l_{n+1,n+1} = 0$, and therefore $ES(0) = EC(0) = 0$, and $LS(n+1) = LC(n+1) = UB$.

In the multi-mode case, the minimum possible duration of the activities must be considered in the bounds of the close times:

$$EC(i) = l_{0,i} + \min_{o \in 1..M_i} p_{i,o}$$
$$LC(i) = UB - l_{i,n+i} + \min_{o \in 1..M_i} p_{i,o}$$

Table 1
Summary of nomenclature.

Parameters	
$V = \{0, \ldots, n+1\}$	Set of activities
$A = \{1, \ldots, n\}$	Set of non-dummy activities
p_i	Duration of activity i
$G = (V, E)$	Precedence graph
$(i, j) \in E$	End-start precedence
$R = \{1, \ldots, v\}$	Set of resources
B_k	Capacity of resource k
$b_{i,k}$	Demand of activity i for resource k
UB	Upper bound on the makespan
$H = \{0, \ldots, UB-1\}$	Scheduling horizon
Pre-computed sets and functions	
$G^* = (V, E^*)$	Extended precedence graph
$(i, j, l_{i,j}) \in E^*$	Extended precedence with lag $l_{i,j}$
$ES(i)$ and $LS(i)$	Earliest and latest start time of activity i
$EC(i)$ and $LC(i)$	Earliest and latest close time of activity i
$RTW(i)$	Running time window of activity i
$STW(i)$	Start time window of activity i
$OA(j)$	Activities that can run at the start of activity j
$\mathcal{P} = \{P_1, \ldots, P_c\}$	Path cover
$\mathcal{P}(t) = \{P_1, \ldots, P_c\}$	Path cover over activities that can run at time t
$\mathcal{P}(j) = \{P_1, \ldots, P_c\}$	Path cover over activities that can run when activity j starts
Variables used in formulations	
S_i	Start time of activity i
$x_{i,t}$	Activity i is running at time t
$z_{i,j}$	Activity i is running when activity j starts

4.4.4. Non-renewable resource demands

An additional preprocess can be applied to the MRCPSP. The idea is to reduce the demand and availability of non-renewable resources in a sound way, so that some demands become zero. This will have a positive impact when encoding non-renewable resource constraints, since monomials of the form $0 \cdot x$ can be removed from PB constraints, and hence smaller SAT encodings for them can be generated. Let us illustrate it with an example.

Example 2. Consider an activity with three execution modes, with demands 2, 3 and 5 (for modes 1, 2 and 3, respectively) over a non-renewable resource with capacity 10. We know for sure that this activity will require at least 2 units of the resource, and therefore we can rewrite the instance by subtracting 2 units in the following way: the demands become 0, 1 and 3, respectively, and the capacity becomes 8.

In general, we can replace the capacity B_k of each resource k and the demands $b_{i,k}$ of each activity i with new capacities B'_k and new demands $b'_{i,k,o}$ in the following way:

$$B'_k = B_k - \sum_{i \in A} \min_{o \in 1..M_i} b_{i,k,o} \qquad \forall k \in NR \quad (3)$$

$$b'_{i,k,o} = b_{i,k,o} - \min_{o' \in 1..M_i} b_{i,k,o'} \qquad \forall i \in A,\ \forall o \in 1..M_i,\ \forall k \in NR \quad (4)$$

We apply this preprocessing step always, so that we use the values of B'_k and $b'_{i,k,o}$ in replacement of B_k and $B_{i,k,o}$.

5. SMT formulations

We provide SMT formulations for the decision version of the RCPSP, the MRCPSP and the RCPSP/t. Therefore, we assume that, apart from a particular problem instance, we receive as input an upper bound UB of the makespan, and the problem consists in determining the existence

Table 2
Specific nomenclature of MRCPSP.

Parameters	
$p_{i,o}$	Duration of activity i in mode o
$R=\{1,\ldots,q\}$	Set of resources
$RR=\{1,\ldots,v\}$	Set of renewable resources
$NR=\{v+1,\ldots,q\}$	Set of non-renewable resources
$b_{i,k,o}$	Demand of activity i for resource k in mode o
M_i	Number of execution modes of activity i
Variables used in formulations	
$x_{i,t,o}$	Activity i is running at time t in mode o
$z_{i,j,o}$	Activity i is running in mode o when activity j starts
$sm_{i,o}$	Activity i is runs in mode o

Table 3
Specific nomenclature of RCPSP/t.

Parameters	
$B_{k,t}$	Capacity of resource k at time t
$b_{i,k,e}$	Demand of activity i for resource k at its eth execution instant
Variables used in formulations	
$y_{i,t}$	Activity i starts at time t

of a schedule with a makespan smaller or equal than UB. Minimization of the makespan will be achieved as we explain later in Section 7.

The SMT formulas that our system provides to the SMT solver are all in CNF. Nevertheless, in order to improve readability of the formulations and make clearer the semantics of each constraint, some precedence and channeling constraints are written as implications or double implications, that are trivially translated to a set of clauses in the obvious way. On the other hand, resource constraints are formulated as PB constraints over Boolean variables. As we have explained in Section 3, PB constraints can be handled in different ways. In particular, in the experimental section we evaluate three alternatives: translating the PB constraints to LIA expressions, using a SAT encoding of PB constraints, and using a SAT encoding of PB(AMO) constraints.

All the formulations that we present have in common that they use the theory of Integer Difference Logic (IDL) to encode precedence constraints. This is a sub-theory of LIA that is restricted to inequalities of the form $x - y < c$ (or basic rewritings of that), where x and y are integer variables, and c is an integer constant (see Barrett et al., 2016). IDL is easier than LIA: a conjunction of IDL expressions can be solved in polynomial time, while solving a conjunction of LIA expressions is NP-complete.

Regarding resource constraints, there have been proposed many ways of formulating them in the literature. In this paper we use the two most recurrent ones (Ansótegui et al., 2011; Schutt et al., 2009): time-indexed formulations in Section 5.1 and task-indexed formulations in Section 5.2. The main difference between these two kinds of formulations is that time-indexed ones discretize the time in unit intervals or time instants, and introduce a resource constraint for each time instant and resource, while task-indexed formulations do not discretize the time in unit intervals but introduce a resource constraint for each activity start and resource.

5.1. Time-indexed formulations

The time-indexed approach to formulate resource constraints (Pritsker, Watters, & Wolfe, 1996) consists in discretizing the *scheduling horizon* H —the period of time in which the schedule will take place— into unit intervals from 0 to the given UB, i.e., $H = \{0, 1, \ldots, UB - 1\}$. Then, to ensure that the capacity of a resource is never exceeded, a resource constraint is added for each resource and time instant in H. To express these constraints, an auxiliary 0/1 variable for each activity and for each time instant is typically introduced. There are different semantics that can be given to these auxiliary time-indexed variables, leading to different kinds of constraints (Artigues, 2013). In our formulations we will be using variables with (extensions of) the following semantics:

$x_{i,t}$: 0/1 variable denoting whether activity i is running at time t.

$y_{i,t}$: 0/1 variable denoting whether activity i starts at time t.

Instead of introducing an auxiliary variable for each activity and time instant in H, we avoid introducing variables for time instants at which we know beforehand that the activity cannot start or be running. This will let us reduce the number of auxiliary variables, and consequently the size of resource constraints. For this purpose we define two intervals of integers for each activity i: the *start time window* $STW(i)$, denoting the set of time instants at which activity i can possibly start, and the *run time window* $RTW(i)$, denoting the set of time instants at which activity i can be running. These are computed as:

$$STW(i) = [ES(i), LS(i)]$$
$$RTW(i) = [ES(i), LC(i) - 1]$$

5.1.1. RCPSP

In the formulation for the RCPSP we have two sets of variables:

S_i: Integer variables denoting the start time of activity i, $\forall i \in V$.

$x_{i,t}$: Boolean variables denoting whether activity i is running at time t, $\forall i \in A$, $\forall t \in RTW(i)$.

The constraints are the following:

$$S_0 = 0 \tag{5}$$
$$S_i \geq ES(i) \qquad \forall i \in V \backslash \{0\} \tag{6}$$
$$S_i \leq LS(i) \qquad \forall i \in V \backslash \{0\} \tag{7}$$
$$S_j - S_i \geq l_{i,j} \qquad \forall (i,j,l_{i,j}) \in E^* \tag{8}$$
$$x_{i,t} \leftrightarrow S_i \leq t \wedge t < S_i + p_i \qquad \forall i \in A,\ \forall t \in RTW(i) \tag{9}$$
$$\sum_{\substack{i \in A,\ s.t. \\ t \in RTW(i)}} b_{i,k} \cdot x_{i,t} \leq B_k \qquad \forall k \in R,\ \forall t \in H \tag{10}$$

Constraint (5) ensures that the initial dummy activity starts at time 0. Constraints (6) and (7) bound the start time variables using the time windows. In particular, Constraint (7) is the one that sets the upper bound on the makespan, since $LS(n+1) = UB$. Constraints (8) enforce the extended precedences. Constraints (9) enforce that variable $x_{i,t}$ is true iff activity i is running at time t. Finally, (10) defines the renewable resource constraints.

5.1.2. MRCPSP

In the formulation for the MRCPSP we have three sets of variables:

S_i: Integer variables denoting the start time of activity i, $\forall i \in V$.

$sm_{i,o}$: Boolean variables denoting whether activity i runs in mode o, $\forall i \in V$, $\forall o \in 1..M_i$.

$x_{i,t,o}$: Boolean variables denoting whether activity i is running at time t in mode o, $\forall i \in A$, $\forall t \in RTW(i)$, $\forall o \in 1..M_i$.

The constraints are the following:

$$S_0 = 0 \tag{11}$$
$$S_i \geq ES(i) \qquad \forall i \in V \backslash \{0\} \tag{12}$$
$$S_i \leq LS(i) \qquad \forall i \in V \backslash \{0\} \tag{13}$$
$$sm_{i,o} \rightarrow S_j - S_i \geq p_{i,o} \qquad \forall (i,j) \in E,\ \forall o \in 1..M_i \tag{14}$$
$$S_j - S_i \geq l_{i,j} \qquad \forall (i,j,l_{i,j}) \in E^* \tag{15}$$

$$\bigvee_{o\in 1..M_i} sm_{i,o} \qquad \forall i \in V \tag{16}$$

$$\overline{sm_{i,o}} \vee \overline{sm_{i,o'}} \qquad \forall i \in V,\ \forall o \in 1..M_i-1,\ \forall o' \in o+1..M_i \tag{17}$$

$$x_{i,t,o} \leftrightarrow S_i \leq t \wedge t < S_i + p_{i,o} \wedge sm_{i,o} \qquad \forall i \in A,\ \forall o \in 1..M_i,\ \forall t \in RTW(i) \tag{18}$$

$$\sum_{\substack{i\in A,\ s.t.\\ t\in RTW(i)}} \sum_{o\in 1..M_i} b_{i,k,o} \cdot x_{i,t,o} \leq B_k \qquad \forall k \in RR,\ \forall t \in H \tag{19}$$

$$\sum_{i\in A} \sum_{o\in 1..M_i} b_{i,k,o} \cdot sm_{i,o} \leq B_k \qquad \forall k \in NR \tag{20}$$

Constraints (11)–(13) and (15) are independent on the execution modes, and therefore have the same semantics as in the RCPSP. Constraints (14) define the precedence relations, relative to which is the execution mode of the predecessor. Constraints (16) ensure that each activity runs at least in one mode, and (17) ensure that each activity runs in at most in one mode. Constraints (18) enforce that variable $x_{i,t,o}$ is true iff activity i is running in mode o at time t. Finally, (19) and (20) define the renewable and non-renewable resource constraints, respectively.

5.1.3. RCPSP/t

The formulation for the RCPSP/t is very similar to the formulation for the RCPSP. However, when formulating renewable resource constraints, we need to know not only if an activity is running at a particular time, but also for how long it has been running at that time, since the demands depend on this. For this purpose, we do not use variables $x_{i,t}$ but another kind of time-indexed variables with different semantics, that we denote by $y_{i,t}$. Therefore, the variables of this formulation are:

S_i: Integer variables denoting the start time of activity i, $\forall i \in V$.

$y_{i,t}$: Boolean variables denoting whether activity i starts at time t.

Then the RCPSP/t is formulated as follows:

$$S_0 = 0 \tag{21}$$

$$S_i \geq ES(i) \qquad \forall i \in V \setminus \{0\} \tag{22}$$

$$S_i \leq LS(i) \qquad \forall i \in V \setminus \{0\} \tag{23}$$

$$S_j - S_i \geq l_{i,j} \qquad \forall (i,j,l_{i,j}) \in E^* \tag{24}$$

$$y_{i,t} \leftrightarrow S_i = t \qquad \forall i \in A,\ \forall t \in STW(i) \tag{25}$$

$$\sum_{\substack{i\in A,\ \text{s.t.}\\ t\in RTW(i)}} \sum_{\substack{e\in 0..p_i-1\ \text{s.t.}\\ t-e\in STW(i)}} b_{i,k,e} \cdot y_{i,t-e} \leq B_{k,t} \qquad \forall k \in R,\ \forall t \in H \tag{26}$$

Constraints (25) enforce that variable $y_{i,t}$ is true iff activity i starts at time t. Constraints (26) define the renewable resource constraints. For all time instant t and resource k, this constraint considers all activities i that can be running at time instant t according to their running time window. For each activity i, the constraint considers all possible start time instants $t-e$ of activity i which imply that it is running at time t. Notice that when activity i starts at time $t-e$ (i.e. $y_{i,t-e}$ is true), its demand for resource k at time t is $b_{i,k,e}$. Recall also that, in the RCPSP/t, resource capacities can be different at each time instant t, but this characteristic naturally fits in the time-indexed Constraint (26).

5.2. Task-indexed formulations

The task-indexed approach to formulate resource constraints (Schutt et al., 2009) is based on the fact that the occupancy of a resource is only increased when an activity starts. Therefore, in order to guarantee that resource capacities are never exceeded, it is enough to guarantee this when activities start. For this purpose, for each pair of activities i, j, a Boolean variable states if an activity i is running when another activity j starts. Therefore, in our formulations we will be using variables with (extensions of) the following semantics:

$z_{i,j}$: Boolean variables denoting whether activity i is running when activity j starts, $\forall i,j \in A$.

Again, in order to reduce the number of auxiliary variables and the size of resource constraints, we avoid introducing variables for pairs of activities that we know beforehand that cannot overlap due to precedences. For this purpose we define the auxiliary set *Overlapping Activities* $OA(j)$ for every activity j, defined as the set of all non-dummy activities different from activity j that are not connected with activity j by a path in the precedence graph:

$$OA(j) = \{i \mid i \in A,\ i \neq j,\ (i,j,l_{i,j}) \notin E^* \text{ and } (j,i,l_{j,i}) \notin E^* \text{ for any } l_{i,j} \text{ and } l_{j,i}\}$$

5.2.1. RCPSP

In the formulation for the RCPSP we have two sets of variables:

S_i: Integer variables denoting the start time of activity i, $\forall i \in V$.

$z_{i,j}$: Boolean variables denoting whether activity i is running when activity j starts, $\forall j \in A$, $\forall i \in OA(j)$.

All constraints that are not related with variables $z_{i,j}$ remain the same as in the time-indexed formulation, that is, the task-indexed formulation includes constraints (5)–(8). Constraints (9) and (10) are replaced by the following two constraints:

$$z_{i,j} \leftrightarrow S_i \leq S_j \wedge S_j < S_i + p_i \qquad \forall j \in A,\ \forall i \in OA(j) \tag{27}$$

$$b_{j,k} + \sum_{i\in OA(j)} b_{i,k} \cdot z_{i,j} \leq B_k \qquad \forall k \in R,\ \forall j \in A \tag{28}$$

5.2.2. MRCPSP

In the formulation for the MRCPSP we have three sets of variables:

S_i: Integer variables denoting the start time of activity i, $\forall i \in V$.

$sm_{i,o}$: Boolean variables denoting whether activity i runs in mode o, $\forall i \in V$, $\forall o \in 1..M_i$.

$z_{i,j,o}$: Boolean variables denoting whether activity i is running in mode o when activity j starts, $\forall j \in A$, $\forall i \in OA(j)$, $\forall o \in 1..M_i$.

Again, we reuse the constraints that are independent of time-indexed variables. Therefore, this formulation contains constraints (11)–(17) and (20). Constraints (18) and (19) are replaced by the following two constraints:

$$z_{i,j,o} \leftrightarrow S_i \leq S_j \wedge S_j < S_i + p_{i,o} \wedge sm_{i,o} \qquad \forall j \in A,\ \forall i \in OA(j),\ \forall o \in 1..M_i \tag{29}$$

$$\sum_{o\in 1..M_j} b_{j,k,o} \cdot sm_{j,o} + \sum_{i\in OA(j)} \sum_{o\in 1..M_i} b_{i,k,o} \cdot z_{i,j,o} \leq B_k \qquad \forall k \in RR,\ \forall j \in A \tag{30}$$

5.2.3. RCPSP/t

The RCPSP/t is inherently related to time-indexed formulations, since the problem definition itself discretizes a scheduling horizon in order to define resource availabilities, as well as execution periods of activities in order to define resource demands. Therefore, we only provide a time-indexed formulation for the RCPSP/t.

One could try to write a task-indexed formulation for the RCPSP/t by inserting a number of constraints for each activity to ensure that, at each time instant of its execution period, the resource constraints are satisfied. Apparently, the number of constraints of this model would be independent of any scheduling horizon. Instead, this would be directly proportional to the sum of durations of the different activities. However, unless we consider unrealistic instances with huge periods of inactivity (with null resource capacity), a schedule will have a number of activities running nearly at every time instant from 0 to the makespan. Therefore, in practice there would be many constraints for each resource and time instant, probably much more than in a time-indexed formulation.

6. PB(AMO) for resource constraints

The formulations described in Section 5 contain a number of PB formulas expressing resource constraints. These are in particular Constraints (10), (19) and (26) for time-indexed formulations, Constraints (28) and (30) for task-indexed formulations, and Constraints (20) in both formulations of the MRCPSP. In these formulations we also have information about some relations hold by the Boolean variables of the PB constraints. In particular, a solution of a RCPSP problem will satisfy a set of AMO constraints, that may not be explicit but implied by other constraints in the problem. In this section we show how to detect these AMO constraints, and therefore, instead of encoding the PB constraints, we will be able to encode PB(AMO) constraints and obtain more compact SAT encodings, as explained in Section 3. We remark that we do not add to our formulations any of the AMO constraints that we identify in this section, but they are already implied by our formulations.

Let us introduce the idea with an example. Considering the time-indexed formulation for the RCPSP, a constraint (10), states that the added demands over renewable resource k of all activities running at any time cannot be greater than its capacity B_k. On the other hand, precedence constraints (8) enforce that two activities i, j with an extended precedence $(i, j, l_{i,j}) \in E^*$ cannot run together at any time. Therefore, there is an implied AMO constraint over the variables $x_{i,t}$ and $x_{j,t}$ for each time instant t considered, implied by Constraints (8) and (9) altogether. Overall, we have a conjunction of (i) a PB constraint over variables $x_{i,t}$ and (ii) a set of (implied) AMO constraints over the same variables. Therefore, we have a PB(AMO) constraint. Intuitively, the SAT encodings of PB(AMO) constraints that we will consider will encode a relaxation of the resource constraints (10). Such relaxed resource constraint will allow some pairs of activities (i, j) overlap at time t (i.e., having both $x_{i,t}$ and $x_{j,t}$ assigned true) even if the capacity of resource k is exceeded because of this overlap. This will be allowed, only if this assignment is already inconsistent with a given AMO constraint between $x_{i,t}$ and $x_{j,t}$ (this given AMO constraint may have been inferred from a precedence relation between activities i and j). Hence, while smaller encodings can be obtained (thanks to ignoring certain assignments locally), the soundness of the overall encoding is not compromised.

The variables of the PB constraints for renewable resources in the RCPSP, both in time-indexed and task-indexed formulations, can have many implied AMO constraints, introduced by precedence relations. Now we introduce a technique to identify such AMO constraints from the precedence graph. Then we show how this method can be used for formulating resource constraints as PB(AMO) constraints in the RCPSP and some of its extensions, such as the MRCPSP and the RCPSP/t.

As stated before, a precedence $(i, j) \in E$ introduces an incompatibility between activities i and j, i.e., they can never be running at the same time. This is also the case for any pair of activities connected by a path in the precedence graph G, i.e., for any pair of activities with an edge in G^*. In other words, the precedence relations ensure that, at any time instant, at most one of the activities of a path in G^* is running. From this information we can infer implicit AMO constraints on the variables $x_{i,t}$ and $z_{i,t}$. To this end, we will use path covers.

Definition 6.1. A *path cover* of a graph $G = (V, E)$ is a partition of the vertices in V into a set of c sequences of vertices $\{P_1, \ldots, P_c\}$, such that each sequence P_π $\forall \pi \in 1..c$ is a path of G. Note that P_π can contain a single vertex. We restrict to vertex-disjoint path covers, i.e., each vertex belongs exactly to one path.

We will compute a path cover $\mathcal{P} = \{P_1, \ldots, P_c\}$ from G^* and then we will extract an AMO constraint from each P_π. From these AMO constraints we will extract the partition of the PB variables characterizing the PB(AMO). The size of all existing PB(AMO) encodings is proportional to the size of the given partition. Therefore, in order to obtain small encodings, a good heuristic is to obtain large AMO constraints and hence partitions with few parts. For that reason, we are interested in computing minimum path covers.

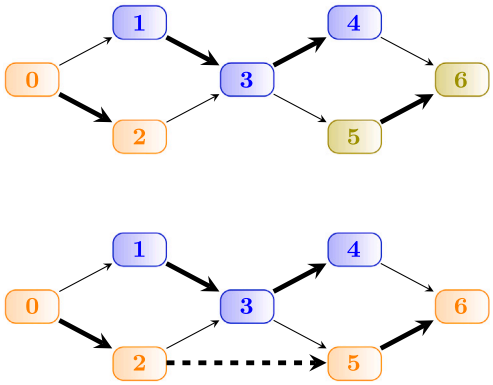


Fig. 5. Minimum path cover of the precedence graph (top) and the extended precedence graph (bottom) of a project.

Definition 6.2. A *minimum path cover* of a graph $G = (V, E)$ is a path cover $\{P_1, \ldots, P_c\}$ of G such that there does not exist any path cover $\{P'_1, \ldots, P'_{c'}\}$ of G with $c' < c$.

Notice the importance of computing a minimum path cover on G^* instead of G. Fig. 5 depicts a minimum path cover of the precedence graph of a project with seven activities, and a minimum path cover of the extended precedence graph of the same project. For the sake of simplicity, the extended precedence graph does not show all the additional extended precedences but only $(2, 5) \in E^*$. Thanks to this extended edge, we can find $\mathcal{P} = \{(1, 3, 4), (0, 2, 5, 6)\}$ which only has two paths. Otherwise, using the non-extended precedence graph, a minimum path cover would contain three paths.

Computing a minimum path cover is an NP-hard problem, since the Hamiltonian path problem is NP-complete. However, in the particular case of directed acyclic graphs (DAG), like the precedence graph, it can be done in polynomial time. We reduce the problem of computing the minimum path cover to the problem of finding a maximum cardinality matching of a bipartite graph (Ntafos & Hakimi, 1979). Then we solve the matching problem with the Hopcroft–Karp algorithm (Hopcroft & Karp, 1973), which runs in $O(|E|\sqrt{|V|})$ time given a bipartite graph $G = (V, E)$.

6.1. AMO constraints in time-indexed formulations

The minimum path cover yields a way of extracting AMO constraints between variables in time-indexed formulations of the RCPSP and its extensions since, given a time instant t, all variables $x_{i,t}$ for activities in the same path hold an AMO relation. If time windows are taken into account, we can focus only on the subgraph containing activities that can be running at a particular time instant t, and probably get smaller covers on G^*. The procedure is the following. For a time instant t:

1. Compute $G^*(t)$ the subgraph from G^* that has as nodes the activities i such that $t \in RTW(i)$, i.e., $G^*(t) = (V(t), E^*(t))$, where:

   $V(t) = \{i \mid i \in V,\ t \in RTW(i)\}$

   $E^*(t) = \{(i, j, l_{i,j}) \mid (i, j, l_{i,j}) \in E^*,\ i \in V(t),\ j \in V(t)\}$

2. Compute a minimum path cover $\mathcal{P}(t) = \{P_1, \ldots, P_c\}$ of $G^*(t)$.
3. Compute an AMO constraint derived from each path $P_\pi \in \mathcal{P}(t)$.

The AMO constraints obtained in the third step depend on the problem at hand. Now we describe which are the AMO constraints obtained for each one of the three considered problems.

6.1.1. RCPSP

Given a time instant t, we know that the following AMO constraints are logically implied by Constraints (9) in the RCPSP time formulation:

$$\sum_{i \in P_\pi} x_{i,t} \leq 1 \qquad \forall P_\pi \in \mathcal{P}(t) \tag{31}$$

These constraints state that two activities belonging to the same path in $\mathcal{P}(t)$ cannot run at the same time. The AMO constraints (31) allow us to define the partition of the variables of the PB constraint (10) for a particular time t and any resource k:

$$\{\ \{x_{i,t} \mid i \in P_\pi\} \mid P_\pi \in \mathcal{P}(t)\ \} \tag{32}$$

The PB constraints (10) and the partitions (32) characterize the PB(AMO)s that will be encoded to SAT.

6.1.2. MRCPSP

Similarly as it happens with the RCPSP, in the MRCPSP we can find AMO relations between the variables of the renewable resource constraints (19), based on precedences. Moreover, in the MRCPSP we have a third subindex in the variables $x_{i,t,o}$ which is the selected execution mode. Notice however, that we can only choose one execution mode per activity, and this introduces additional AMOs. In particular, there are the following implicit constraints in the formulation for a particular time t:

$$\sum_{i \in P_\pi} \sum_{o \in 1..M_i} x_{i,t,o} \leq 1 \qquad \forall P_\pi \in \mathcal{P}(t) \tag{33}$$

Intuitively, the implicit AMO Constraints (33) state that over all activities connected by a precedence path, at most one of them will be running at a particular time, and in at most one execution mode. These AMO constraints are logically implied by Constraints (14), (15), (17) and (18). Similarly as done in RCPSP, from the AMO constraints (33), we can easily extract a partition of the variables of the PB constraints (19) for a particular time t and any renewable resource k. From these partitions and PB constraints we can make up a set of corresponding PB(AMO) constraints.

On the other hand, non-renewable resource Constraints (20) have variables $sm_{i,o}$ for the running modes of the activities. Constraints (17) enforce that each activity is running in at most one mode, i.e., they encode the following constraint:

$$\sum_{o \in 1..M_i} sm_{i,o} \leq 1 \qquad \forall i \in V \tag{34}$$

Therefore, for each non-renewable resource we can also obtain PB(AMO) constraints from PB constraints (20) and the partitions of their variables extracted from the AMO constraints in (34).

6.1.3. RCPSP/t

Analogously to what happens with the variables $x_{i,t}$ in the RCPSP, in the RCPSP/t the precedence relations introduce implicit AMO constraints over the variables $y_{i,t}$. Intuitively, if an activity i starts at time t, an activity j connected with activity i by a path in the precedence graph cannot start at time t, nor start at any other time that, according to the durations of the activities, implies an overlapping. Moreover, we know that an activity will only start at one particular time instant, and therefore there is an AMO constraint over all variables $y_{i,t}$ of the same activity i. This last AMO is implicitly enforced by the channeling between $y_{i,t}$ and the integer start variables S_i, in Eq. (25). Therefore, for a given time t, we have the following implicit constraints, which are logically implied by Constraints (24) and (25):

$$\sum_{i \in P_\pi} \sum_{\substack{e \in 0..p_i-1, \text{s.t.} \\ t-e \in STW(i)}} y_{i,t-e} \leq 1 \forall P_\pi \in \mathcal{P}(t) \tag{35}$$

The implicit Constraints (35) impose that over all activities connected by a path, at most one of them will be running at a particular time. This is achieved by imposing that, among all the time instants $t-e$ at which this activity i could have started, at most one of $y_{i,t-e}$ can be true.

Similarly as done in RCPSP, from the AMO constraints (35), we can easily extract a partition of the variables of the PB constraints (26) for a particular time t and any resource k. From these partitions and PB constraints we can make up a set of corresponding PB(AMO) constraints.

6.2. AMO constraints in task-indexed formulations

Although task-indexed resource constraints are not related to a particular time instant know beforehand, they are also related to a particular point in the schedule, which is the time instant at which activity j starts. Therefore, path covers also yield AMO constraints over the variables $z_{i,j}$ of resource constraints (28), i.e., over all the variables stating whether an activity i can possibly be running when activity j starts. For instance, if two activities i and a are connected by a path, they cannot be running in parallel when an activity j starts, and therefore at most one of $z_{i,j}$ and $z_{a,j}$ can be true. In this case we will compute a path cover for each activity j and extract associated AMO constraints to build the PB(AMO)s:

1. Compute $G^*(j)$ the subgraph from G^* which has as nodes the activities $i \in OA(j)$, i.e., $G^*(j) = (OA(j), E^*(j))$, where:

   $$E^*(j) = \{(i, a, l_{i,a}) \mid (i, a, l_{i,a}) \in E^*,\ i \in OA(j),\ a \in OA(j)\}$$

2. Compute a minimum path cover $\mathcal{P}(j) = \{P_1, \ldots, P_c\}$ of $G^*(j)$.
3. Compute an AMO constraint derived from each $P_\pi \in \mathcal{P}(j)$.

The AMO constraints obtained in the third step depend on the problem at hand. Now we describe which are the AMO constraints for each considered problem.

6.2.1. RCPSP

Given an activity j, we know that the following AMO constraints are logically implied by Constraints (8) and (27) in the RCPSP task formulation:

$$\sum_{i \in P_\pi} z_{i,j} \leq 1 \qquad \forall P_\pi \in \mathcal{P}(j) \tag{36}$$

Again, the AMO constraints (36) allow us to define the partition of the variables of the PB constraint (28) for a particular activity j and any resource k. From these partitions and PB constraints we obtain the corresponding PB(AMO)s.

6.2.2. MRCPSP

Since non-renewable resource constraints are the same in task-indexed and time-indexed formulations, we will also use AMO constraints (34) to make up PB(AMO) constraints from (20).

On the other hand, renewable resource constraints (30) are formulated over variables $z_{i,j,o}$ and $sm_{j,o}$. Then, for a particular activity j and any renewable resource k we can use the following AMO constraints to obtain a partition of the variables and make up the corresponding PB(AMO)s:

$$\sum_{i \in P_\pi} \sum_{o \in 1..M_i} z_{i,j,o} \leq 1 \qquad \forall P_\pi \in \mathcal{P}(j) \tag{37}$$

$$\sum_{o \in 1..M_j} sm_{j,o} \leq 1 \tag{38}$$

These AMO constraints are logically implied by Constraints (14), (15), (17) and (29).

7. Minimizing the makespan

The formulations given in Section 5 are designed for the decision version of the considered scheduling problems. Therefore, they require as input an upper bound on the makespan. Optimality can be obtained by iteratively encoding decision problems to SMT with decreasing upper bounds, until we find a value UB for which the instance has no solution, and therefore the optimal solution is the one obtained with $UB+1$. It may also be the case that the algorithm finds a solution with a makespan equal to LB. In such a case, the optimality of the solution is certified without the need of finding an unsatisfiable upper bound. This fact highlights the importance of having good preprocesses to find tight lower and upper bounds.

The optimization procedure corresponds to Algorithm 1. The specification of the functions used in the algorithm is the following.

$compute\_Upper\_Bound(I)$**:** given an instance I of the problem at hand, it computes an upper bound UB of the optimum makespan. The implementation of this method for each particular problem is described in the following subsections.

$compute\_Lower\_Bound(I)$: given an instance I of the problem at hand, it returns a lower bound for the makespan. We will be using the earliest start time of the last dummy activity, i.e., $ES(n+1)$.

$smt\_encode(I, UB)$**:** given an instance I of the problem at hand and an upper bound UB, it returns an SMT formula that encodes the feasibility of a schedule of instance I with a makespan smaller or equal than UB.

$smt\_check(ENC)$**:** checks using an SMT solver whether ENC is satisfiable. Returns $(true, MODEL)$ if $MODEL$ is a model of ENC, and $(false, \{\})$ otherwise.

$get\_makeS(ENC, MODEL)$**:** retrieves the value of the makespan from the model $MODEL$ of the encoding ENC.

$get\_solution(ENC, MODEL)$**:** retrieves a solution for the original problem from the model $MODEL$ of the encoding ENC.

Algorithm 1 Minimize makespan

Require: Problem instance I
Ensure: Return optimal solution and makespan if I is satisfiable. Otherwise, return *unsat*.

```
UB ← compute_Upper_Bound(I)
LB ← compute_Lower_Bound(I)
ENC ← smt_encode(I, UB)
(SAT, MODEL) ← smt_check(ENC)
if SAT then
  MAKESPAN ← get_makeS(ENC, MODEL)
  SOLUTION ← get_solution(ENC, MODEL)
  UB ← MAKESPAN − 1
else
  return unsat
end if
while SAT and UB ≥ LB do
  ENC ← ENC ∪ {S_{n+1} ≤ UB}
  (SAT, MODEL) ← smt_check(ENC)
  if SAT then
    MAKESPAN ← get_makeS(ENC, MODEL)
    SOLUTION ← get_solution(ENC, MODEL)
    UB ← MAKESPAN − 1
  end if
end while
return (SOLUTION, MAKESPAN)
```

It is worth mentioning that currently there exist SMT solvers that support Optimization Modulo Theories (OMT) (Sebastiani & Tomasi, 2012), an extension of SMT which allows to directly specify an objective function that the solver will optimize. For instance, in RCPSP-like problems, optimization can be achieved by simply specifying the objective function:

$$minimize \quad S_{n+1}$$

Nevertheless, we have conducted some experiments to evaluate the performance of several SMT solvers in our system, including solvers which offer optimization capabilities, such as OptiMathSAT (Sebastiani & Trentin, 2015) and Z3 (de Moura & Bjørner, 2008). Yices (Dutertre & de Moura, 2006) has shown the best performance, even though it requires to implement an ad-hoc optimization procedure like the one we have described. We did not observe a big reduction on the number of timeouts, but the solving times were generally smaller. There are several possible reasons why in RCPSP-like problems implementing an ad-hoc optimization algorithm does not penalize the solving time. First of all, we are optimizing a simple (regarding expressiveness) objective function, which can be bounded with just one unit clause of the form $S_{n+1} \leq UB$. Moreover, we have checked that adding an upper bound on the makespan enables theory propagation to set new upper bounds on the values of the start time variables, due to the transitivity of the precedence constraints. Also, the SAT engine within the SMT solver can unit-propagate to false the corresponding time-indexed or task-indexed Boolean variables. It is also relevant to note that we are integrating the back-end SMT solver in our system using its API, which allows the solver to keep the learned lemmas and decision heuristic metrics between the different calls.

One of the key points to solve the studied problems efficiently is being able to determine a good initial upper bound UB, since the smaller the upper bound, the less SMT satisfiability checks are required. Moreover, the size of time-indexed formulations is directly proportional to the value of UB, and a very high value could lead to practically unsolvable SMT formulas. In the RCPSP there is no constraint on how late an activity can start if the schedule is not limited by a maximum makespan. Therefore, a trivial upper bound can be obtained from a schedule in which only one activity runs at a time and there are no periods of inactivity. In any such schedule, the makespan is the sum of the durations:

$$\sum_{i \in A} p_i \tag{39}$$

This is a valid value for UB, but it will generally be very far from the optimal makespan. To overcome this issue, we implement a fast greedy heuristic to get a first schedule from which we can obtain a first upper bound. This first upper bound will be much smaller than the trivial upper bound (39) in most cases. This heuristic is the *Parallel Scheduling Generation Scheme* (PSGS) proposed in Kelley (1963) and described in Kolisch (1996). Given a project of n activities, this method requires at most n stages to find a schedule, and at each stage a subset of the activities are scheduled. Each stage s has associated a schedule time t_s (where $t_{s'} \leq t_s$, for $s' \leq s$). There are four activity sets that are updated as the algorithm runs:

- Complete set $\mathcal{C}$: activities already scheduled and completed up to the schedule time t_s.
- Active set $\mathcal{A}$: activities already scheduled, but still active at the schedule time t_s.
- Remaining set $\mathcal{R}$: activities not yet scheduled at the schedule time t_s.
- Decision set $\mathcal{D}$: activities not yet scheduled which are available for scheduling with start time t_s, w.r.t. precedence and resource constraints.

Each stage consists of two steps:

1. Defining the new t_s as the earliest completion time of activities in the active set A —the first stage starts at $t_0 = 0$. The activities with a finish time equal to the new t_s are removed from A and put into C. This may place new activities into D.
2. One activity from D is selected by some priority rule — we will be using input order, and scheduled to start at t_s, being removed from D and added to A. Then the set D is recomputed. This step is repeated until D becomes empty.

The method terminates when all activities are scheduled. We provide an algorithmic description of PSGS in Algorithm 2. We use the function *valid_activities_at_time* that, given the instance parameters I, a (partial) schedule S, the set of activities pending to be scheduled $\mathcal{R}$, and a time instant t, it computes which activities could start at time t considering precedence and resource constraints.

Algorithm 2 PSGS

Require: Problem instance $I = (V, p, E, R, B, b)$
Ensure: Return solution $S = (S_0, \ldots, S_{n+1})$ of I
  $\mathcal{R} \leftarrow V \setminus \{0\}$
  $\mathcal{A} \leftarrow \{0\}$
  $C \leftarrow \emptyset$
  $D \leftarrow \emptyset$
  $S \leftarrow (0, 0, \ldots, 0)$
  while $\mathcal{R} \neq \emptyset$ **do**
    $t \leftarrow \min_{i \in \mathcal{A}}(S_i + p_i)$
    //Compute activities finishing at time t
    $Ct \leftarrow \{i \mid i \in \mathcal{A}, S_i + p_i = t\}$
    $\mathcal{A} \leftarrow \mathcal{A} \setminus Ct$
    $C \leftarrow C \cup Ct$
    $D \leftarrow valid\_activities\_at\_time(I, S, \mathcal{R}, t)$
    while $D \neq \emptyset$ **do**
      $i \leftarrow pick\_any(D)$
      $S_i \leftarrow t$
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{i\}$
      $\mathcal{R} \leftarrow \mathcal{R} \setminus \{i\}$
      $D \leftarrow valid\_activities\_at\_time(I, S, \mathcal{R}, t)$
    end while
  end while
  return S

7.1. UB computation in the MRCPSP

Similarly as in RCPSP, a trivial UB can be computed as:

$$\sum_{i \in A} \max_{o \in 1..M_i} p_{i,o} \tag{40}$$

Again, this UB will probably be far from the optimal makespan, and therefore we propose a procedure to find a tighter UB.

Due to the non-renewable resource constraints in the MRCPSP, the PSGS algorithm cannot be adapted to the multi-mode case by simply greedily selecting execution modes for the activities as they are scheduled. Therefore, for the MRCPSP we implement a two-step procedure to find a first solution that gives an upper bound of the makespan.

The first step consists in finding a feasible schedule of modes for the instance, i.e., a schedule of modes that satisfies the non-renewable resource constraints. Therefore, we build an SMT formula containing only variables $sm_{i,o}$, and Constraints (16), (17) and (20):

$$\bigvee_{o \in 1..M_i} sm_{i,o} \qquad \forall i \in V \tag{16}$$

$$\overline{sm_{i,o}} \vee \overline{sm_{i,o'}} \qquad \forall i \in V,\ \forall o \in 1..M_i - 1,\ \forall o' \in o+1..M_i \tag{17}$$

$$\sum_{i \in A} \sum_{o \in 1..M_i} b_{i,k,o} \cdot sm_{i,o} \leq B_k \qquad \forall k \in NR \tag{20}$$

we solve this formula with an SMT-solver. Notice that if the formula is unsatisfiable, the whole MRCPSP instance is infeasible.

In the second step, given the obtained schedule of modes, we fix the durations and resource demands of the activities according to the schedule of modes. Notice that by fixing the durations and resource demands of the activities, we obtain an RCPSP instance instead of an MRCPSP instance. Then we just need to use the PSGS algorithm to find a solution of this RCPSP instance. The makespan of the obtained solution is a valid upper bound of the whole MRCPSP instance. Using this upper bound, we encode the whole MRCPSP instance, either with the time-indexed or the task-indexed formulation, and start the optimization loop.

7.2. UB computation in the RCPSP/t

Every instance of the RCPSP/t already contains an upper bound in its definition, which is the maximum time instant at which resource capacities are defined. We consider that any instance that does not admit a schedule within its predefined upper bound is infeasible. This predefined value of UB is a valid candidate to start the optimization loop, but it is possibly much higher than the optimum makespan. Therefore, we also run the PSGS heuristic before solving RCPSP/t instances.

The high-level PSGS algorithm for RCPSP/t is essentially the same than the one given for RCPSP in Algorithm 2. The only difference is that, since RCPSP/t has a pre-defined UB, the greedily constructed schedule could exceed this upper bound. In such case we give up with the greedy search of a first schedule, and start the optimization loop from the predefined upper bound of the instance.

At a lower level, the check of resource constraints in function *valid_activities_at_time* of Algorithm 2, must now take into account the variability of demands and capacities over time.

8. Experiments

In this section we evaluate, on the one hand, the performance of the given SMT formulations with different encodings for PB resource constraints (Section 8.1), and the scalability of our system in Section 8.2. On the other hand, in Section 8.3 we compare the performance of our system with the best exact state-of-the art solvers for the considered problems. We perform these experiments by solving the instances of the following representative hard benchmarks for the three studied problems:

RCPSP datasets. We consider the *j30*, *j60*, *j90* and *j120* datasets from Kolisch and Sprecher (1997), *Pack* from Carlier and Néron (2003), and *Pack_d* from Koné et al. (2011).

MRCPSP datasets. We consider the *j30* dataset from Kolisch and Sprecher (1997), and *MMLIB50* (in short *M50*) and *MMLIB100* (in short *M100*) from Van Peteghem and Vanhoucke (2014).

RCPSP/t datasets. We consider the *j30*, *j120* datasets, generated in Hartmann (2013) and available in Kolisch and Sprecher (1997).

Tables 4–6 list the characteristics of the datasets for the RCPSP, the MRCPSP and the RCPSP/t, respectively. The information contained in the tables is: number of instances in the dataset (#ins); number of feasible instances (#f); number of non-dummy activities in the instances of the dataset (#act); only in the MRCPSP, number of modes of non-dummy activities (#m); number of renewable resources (#res), and only in the MRCPSP also non-renewable resources.

The experiments have been run on a 8 GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz. We used Yices 2.6.1 (Dutertre & de Moura, 2006) as core SMT solver in our system, communicating with it through its API, using the `multi-check` mode. This mode enables to perform consecutive satisfiability queries while adding new variables and

Table 4
Summary of RCPSP datasets.

set	#ins	#f	#act	#res
j30	480	480	30	4
j60	480	480	60	4
j90	480	480	90	4
j120	600	600	120	4
Pack	55	55	15–33	2–5
Pack_d	55	55	15–33	2–5

Table 5
Summary of MRCPSP datasets.

set	#ins	#f	#act	#m	#res
j30	640	552	30	3	2,2
M50	540	540	50	3	2,2
M100	540	540	100	3	2,2

Table 6
Summary of RCPSP/t datasets.

set	#ins	#f	#act	#res
j30	2880	2826	30	4
j120	3600	3600	120	4

clauses (such as tighter makespan bounds) between calls. The internal state of the solver, including learned clauses, is maintained between consecutive calls.[1]

In our experiments we use different exact solving systems for the different studied problems and with different configurations, as explained in the following subsections. For each instance and setting, we run the considered system with a cutoff of 600 s. There are four different possible outcomes:

1. An optimal solution is found (with certified optimality) in less than 600 s.
2. Unfeasibility of the instance is certified in less than 600 s.
3. Some solutions are found, but the optimality is not certified in less than 600 s.
4. No solution is found in less than 600 s and unfeasibility is not certified either.

When the outcome is 1, 2 or 3 we count the instance as *solved* and in addition, when the outcome is 1 or 2 we also count the instance as *certified*. Then, for each dataset and setting, we report the following measures related with the solving times and the quality of the solutions:[2]

- (**med**, **Q3**, **avg**) The median, third quartile and average of the solving time spent in each instance. For the instances that timed out (i.e. outcomes 3 and 4) we count the solving time as 600 s.
- (**#s**) The number of solved instances, i.e. outcomes 1, 2 and 3.
- (**#c**) The number of certified instances, i.e. outcomes 1 and 2.
- ($\boldsymbol{\Delta_{LB}}$) The average, among all instances I, of the normalized distances between the best found makespan ($Best\_MS(I)$) and the best known lower bound ($Best\_LB(I)$). Namely, this distance is computed as:

$$\frac{Best\_MS(I) - Best\_LB(I)}{Best\_LB(I)} \cdot 100$$

Unfeasible instances are left off this calculus. As particular cases: when the optimality is certified, $Best\_LB(I)$ is considered to be the optimal solution, and therefore the distance is 0; when no solution is found, $Best\_MS(I)$ is considered to be the trivial UB of the instance, i.e. Eq. (39) for RCPSP, Eq. (40) for MRCPSP, and the UB defined in the instance for RCPSP/t.

Table 7
Encoding sizes for the instances of the RCPSP.

	Time-indexed			Task-indexed		
	#Bv	#Iv	#cl	#Bv	#Iv	#cl
	j30					
LIA	821	736	6 424	510	486	4 231
PB	9 940	32	21 846	6 614	32	14 620
PBA	5 123	32	16 022	2 878	32	9 334
	j60					
LIA	2 155	2 037	17 388	2 338	2 310	19 324
PB	52 827	62	110 831	72 369	62	150 394
PBA	28 008	62	86 935	32 058	62	106 977
	j90					
LIA	3 831	3 686	31 259	5 630	5 600	46 278
PB	136 679	92	282 576	266 371	92	545 728
PBA	73 653	92	227 151	117 093	92	398 121
	j120					
LIA	9 110	8 986	74 256	10 457	10 369	85 682
PB	344 695	122	709 968	389 513	122	802 802
PBA	177 729	122	600 471	189 893	122	661 492
	Pack					
LIA	1 426	1 405	11 378	518	518	4 295
PB	16 533	25	36 073	4 966	25	11 219
PBA	12 879	25	31 644	3811	25	9 806
	Pack_d					
LIA	39 673	38 939	317 076	518	518	4 295
PB	467 736	25	1 017 542	4 966	25	11 219
PBA	364 709	25	892 274	3 811	25	9 806

8.1. Comparison of SMT encodings

In Section 5 we have provided time-indexed and task-indexed formulations for the RCPSP, the MRCPSP, and a time-indexed formulation for the RCPSP/t. All these formulations express resource constraints as PB constraints. These constraints must be encoded to SMT, and we consider three approaches explained in Sections 3 and 6:

LIA LIA expressions over 0/1 integer variables.

PB SAT encodings of PB constraints.

PBA SAT encodings of PB(AMO) constraints.

We have tried different SAT encodings of PB and PB(AMO) constraints. Regarding PB constraints, we considered encodings based on Binary Decision Diagrams (BDD) (Abío et al., 2012), Generalized Totalizers (GT) (Joshi et al., 2015), Sequential Weight Counters (SWC) (Höllddobler et al., 2012) and Global Polynomial Watchdog (GPW) (Bailleux et al., 2009). Regarding PB(AMO) constraints, we considered the generalizations of the former encodings from Bofill et al. (2017a) and Bofill et al. (2019), which are referred to as MDD, GGT, GSWC and GGPW in Bofill et al. (2019). Although there were slight performance differences in some datasets, the encodings that gave best results in general were the ones based on decision diagrams. Therefore we only provide detailed results for encodings based in decision diagrams in the tables below, and summarize the results with the other encodings in Section 8.1.3.

Tables 7–9 contain the encoding sizes of the instances for the RCPSP, the MRCPSP and the RCPSP/t, respectively. In these tables we provide, for each setting and dataset: the average number of Boolean variables (**#Bv**); the average number of Integer variables (**#Iv**); the average number of clauses (**#cl**). These average sizes correspond to the first SMT formulas generated in the optimization processes. Notice that

[1] Our system is available at http://imae.udg.edu/recerca/lai/.

[2] All the detailed results for every single instance are available at http://imae.udg.edu/recerca/lai/.

Table 8
Encoding sizes for the instances of the MRCPSP.

	Time-indexed			Task-indexed		
	#Bv	#Iv	#cl	#Bv	#Iv	#cl
	j30					
LIA	5 112	5 018	40 360	1 513	1 530	12 292
PB	123 350	32	256 888	33 028	32	69 327
PBA	19 512	32	89 386	6 152	32	24 955
	MMLIB50					
LIA	7 413	7 343	59 439	3 830	3 859	31 440
PB	278 933	52	573 412	159 836	52	328 222
PBA	44 110	52	200 742	32 005	52	127 910
	MMLIB100					
LIA	17 890	17 813	145 518	15 122	15 179	124 314
PB	698 656	102	1 436 064	745 952	102	1 525 726
PBA	108 652	102	441 467	140 467	102	534 722

Table 9
Encoding sizes for the instances of the RCPSP/t.

	#Bv	#Iv	#cl
	j30		
LIA	1 706	1 606	12 261
PB	198 126	32	398 801
PBA	15 605	32	60 406
	j120		
LIA	17 382	16 840	124 586
PB	1 747 948	122	3 507 249
PBA	331 425	122	1 325 129

the PB and PB(AMO) settings have the same small number of integer variables. This corresponds to the number of activities of the instances in the dataset, i.e., there is one integer start time variable for each activity.

Tables 10–12 contain the measures related with the solving times and the quality of the solutions for the RCPSP, the MRCPSP and the RCPSP/t, respectively.

8.1.1. Time-indexed vs. Task-indexed formulations

Generally, the time-indexed formulations behave better than the task-indexed ones in all datasets and with all PB encodings, even in the datasets where time-indexed encodings are bigger (namely j30 and Pack datasets in the RCPSP, and j30 and M50 in the MRCPSP). This better performance can be observed in the quality of the solutions (Δ_{LB}) and the number of certified instances with the given cutoff. Also, if we look at the percentiles of the solving times, we can observe that the time needed to certify half or three quarters of the instances is much smaller. On the other side, whereas the number of certified instances is much better in time-indexed formulations in MRCPSP, the quality of the solutions of task-indexed formulations is competitive.

The predominance of time-indexed encodings is not that clear in the Pack_d dataset of the RCPSP, where the sizes of time-indexed formulations is much bigger than the ones of task-indexed formulations. This is because in this dataset, the activities have very long durations, and therefore the scheduling horizon is also very large. For this reason, the time-indexed formulations require much more constraints and auxiliary variables than task-indexed formulations.

8.1.2. LIA vs. PB vs. PB(AMO) encodings

We can observe that in the RCPSP, SAT encodings are generally better than LIA encodings, and in particular the PB(AMO) encoding is the best. This is true for both time-indexed and task-indexed formulations. The predominance of PB/PB(AMO) encodings is dramatic in the MRCPSP and the RCPSP/t. Although the number of Boolean variables and clauses of LIA is much smaller than those of PB/PB(AMO) SAT encodings, performance differences cannot be just directly related to the size difference. Notice that the clauses of the SAT encoding approaches only contain either Boolean variables or IDL expressions, while in the LIA approach, even if there are fewer clauses, these clauses contain LIA expressions, and therefore the theory solver is required to solve a harder set of constraints.

Among the SAT encoding approaches, we can see that PB(AMO) is slightly better than PB in the RCPSP and dramatically better in the MRCPSP and the RCPSP/t. The advantage of PB(AMO) over PB is also observed in terms of Boolean variables and clauses of the encodings, and again the decrease in size is much more evident in the MRCPSP and the RCPSP/t. It is worth to highlight that in RCPSP/t, the PB encoding does not find any solution for two instances, whilst the others do. This is because in those instances the PSGS algorithm does not find a solution within the given UB, and therefore the instance-defined UB is used to encode the instance. Since LIA expresses a resource constraint with only one expression and without auxiliary variables, the resulting formula is still small. On the contrary, if we encode the PB constraints to SAT, the resulting formula has a prohibitive size. However, if we use a PB(AMO) encoding instead, we are able to obtain compact enough formulas and find solutions even for these two instances.

The hardness of finding a (not necessarily optimal) solution for MRCPSP is also observable in our results, and especially in set MMBLIB100. Here, the best choice is again using PB(AMO) encodings: only 9 instances remain unsolved using PB(AMO), whereas PB and LIA do not solve 46 and 79 instances respectively. This fact is also translated to a dramatic penalization over Δ_{LB} for the LIA and PB encodings.

Finally, we remark that these 9 MMLIB100 instances are the only ones considering all problems and datasets that are not solved with PB(AMO) within the given timeout of 600 s. In other words, with PB(AMO) encoding and time-indexed formulation, our exact solving system is able to provide solution for 99.9% of the instances. Moreover, it is able to certify the unfeasibility or provide and certify an optimal solution of 79.6% of the instances.

8.1.3. Other SAT encodings of PB and PB(AMO) constraints

As mentioned, the former results are obtained using the BDD for the PB encodings and MDD for the PB(AMO) encodings, but we have tried other configurations. Overall, PB(AMO) encodings are always better than PB encodings regardless of which PB encoding technique is used (BDD, GT, SWC or GPW) and which PB(AMO) encoding technique is used (MDD, GGT, GSWC, GGPW). Also, MDD is better or equally good than others regarding number of timeouts in most problems and datasets, but there are a few exceptions: in the j60 set of RCPSP, with the task-indexed formulation, GGT and GSWC certify 2 more optimum makespans and GGPW 1 more; in the j30 set of MRCPSP, with the time-indexed formulations, GGPW certifies 2 more optimum makespans and GGT and GSWC 1 more; in the M100 set of MRCPSP, with the task-indexed formulations, GGT certifies 4 more optimums and GSWC 3 more; in the j120 set of RCPSP/t, GGT certifies 4 more optimums.

Table 10
Measures for the datasets of the RCPSP, considering time-indexed and task-indexed formulations, with the three PB encodings: LIA, PB and PB(AMO).

	Time-indexed						Task-indexed					
	med	Q3	avg	#s	#c	Δ_{LB}	med	Q3	avg	#s	#c	Δ_{LB}
	j30 (480 instances, all feasible)											
LIA	0.04	0.10	0.45	480	480	0.00	0.09	0.38	12.04	480	475	0.35
PB	0.03	0.09	0.27	480	480	0.00	0.08	0.34	11.23	480	475	0.35
PBA	0.03	0.07	0.22	480	480	0.00	0.07	0.31	11.27	480	475	0.38
	j60 (480 instances, all feasible)											
LIA	0.10	0.51	64.11	480	433	2.04	1.66	18.58	107.47	480	406	3.84
PB	0.13	0.70	63.78	480	434	1.94	1.46	12.00	103.35	480	408	3.65
PBA	0.12	0.52	61.90	480	435	1.88	1.25	10.41	102.32	480	407	3.80
	j90 (480 instances, all feasible)											
LIA	0.23	1.65	106.17	480	400	3.72	9.27	130.26	148.03	480	375	5.23
PB	0.37	2.11	105.41	480	399	3.35	7.97	76.53	140.18	480	379	5.08
PBA	0.32	1.62	104.59	480	400	3.24	6.75	56.71	137.57	480	379	4.95
	j120 (600 instances, all feasible)											
LIA	t.o.	t.o.	326.78	600	284	11.09	t.o.	t.o.	433.41	600	202	16.75
PB	t.o.	t.o.	322.30	600	284	9.59	t.o.	t.o.	416.82	600	222	15.20
PBA	t.o.	t.o.	320.50	600	288	9.55	t.o.	t.o.	412.77	600	225	15.01
	Pack (55 instances, all feasible)											
LIA	6.37	t.o.	175.78	55	40	3.89	127.77	t.o.	293.94	55	30	5.15
PB	4.34	t.o.	174.44	55	40	3.89	160.06	t.o.	302.32	55	29	5.25
PBA	5.82	t.o.	166.65	55	41	3.87	203.05	t.o.	296.26	55	30	5.14
	Pack_d (55 instances, all feasible)											
LIA	265.39	t.o.	322.75	55	31	6.00	144.39	t.o.	281.78	55	32	3.74
PB	207.28	t.o.	297.96	55	34	3.68	135.47	t.o.	278.93	55	32	3.74
PBA	166.35	t.o.	287.40	55	34	3.65	118.73	t.o.	276.65	55	32	3.74

Table 11
Measures for the datasets of the MRCPSP, considering time-indexed and task-indexed formulations, with the three PB encodings: LIA, PB and PB(AMO).

	Time-indexed						Task-indexed					
	med	Q3	avg	#s	#c	Δ_{LB}	med	Q3	avg	#s	#c	Δ_{LB}
	j30 (640 instances, 552 feasible)											
LIA	1.13	2.32	29.76	640	619	1.63	0.99	3.15	44.63	640	605	2.05
PB	1.75	4.01	23.42	640	625	1.24	0.82	1.63	38.21	640	609	1.90
PBA	0.52	0.75	15.97	640	630	0.71	0.36	0.79	36.47	640	613	1.69
	MMLIB50 (540 instances, all feasible)											
LIA	5.06	t.o.	210.07	537	366	26.62	14.55	t.o.	233.12	537	345	28.51
PB	8.48	133.62	145.36	540	432	8.03	10.26	248.52	158.43	540	424	8.25
PBA	2.01	15.90	106.53	540	452	6.04	4.08	41.07	126.94	540	437	6.71
	MMLIB100 (540 instances, all feasible)											
LIA	t.o.	t.o.	335.41	461	255	351.12	t.o.	t.o.	384.59	461	217	355.19
PB	130.77	t.o.	304.85	494	284	216.41	t.o.	t.o.	358.04	494	260	221.02
PBA	26.45	t.o.	275.68	531	305	61.16	273.35	t.o.	317.50	531	283	65.71

Table 12
Measures for the datasets of the RCPSP/t, considering the time-indexed formulation, with the three PB encodings: LIA, PB and PB(AMO).

	med	Q3	avg	#s	#c	Δ_{LB}
	j30 (2880 instances, 2826 feasible)					
LIA	0.72	3.32	12.26	2880	2856	0.49
PB	1.06	2.96	2.88	2880	2880	0.00
PBA	0.15	0.37	0.34	2880	2880	0.00
	j120 (3600 instances, all feasible)					
LIA	t.o.	t.o.	346.47	3600	1792	32.13
PB	323.12	t.o.	325.63	3598	1977	29.19
PBA	41.82	t.o.	240.01	3600	2292	12.14

8.2. System's scalability

Now we study what is the performance of our system if smaller cutoff times are given. We consider the time-indexed and task-indexed formulations for the PB(AMO) encoding, which have exhibited the best performance in the previous experiments. These results can be seen in Tables 13–15.

Regarding number of solved instances there is almost no difference if we decrease the cutoff, i.e., even with only 6 s our system is able to provide a solution for 99.5% of the instances. Here, the greedy

Table 13
Measures for the datasets of the RCPSP with increasing cutoffs of 6, 60 and 600 s.

	Time-indexed						Task-indexed					
	med	Q3	avg	#s	#c	Δ_{LB}	med	Q3	avg	#s	#c	Δ_{LB}
	j30 (480 instances, all feasible)											
6 s.	0.03	0.07	0.18	480	475	0.25	0.07	0.31	0.72	480	448	2.40
60 s.	0.03	0.07	0.22	480	480	0.00	0.07	0.31	2.81	480	466	0.93
600 s.	0.03	0.07	0.22	480	480	0.00	0.07	0.31	11.27	480	475	0.38
	j60 (480 instances, all feasible)											
6 s.	0.12	0.52	1.12	480	411	3.42	1.25	t.o.	2.39	480	349	8.29
60 s.	0.11	0.53	7.70	480	427	2.43	1.25	10.41	13.70	480	390	5.27
600 s.	0.12	0.52	61.90	480	435	1.88	1.25	10.41	102.32	480	407	3.80
	j90 (480 instances, all feasible)											
6 s.	0.34	1.61	1.58	480	383	4.78	t.o.	t.o.	3.70	480	240	10.78
60 s.	0.33	1.61	11.79	480	393	3.94	6.76	57.25	20.28	480	360	6.88
600 s.	0.32	1.62	104.59	480	400	3.24	6.75	56.71	137.57	480	379	4.95
	j120 (600 instances, all feasible)											
6 s.	t.o.	t.o.	4.09	600	252	12.96	t.o.	t.o.	5.97	600	9	26.10
60 s.	t.o.	t.o.	34.24	600	268	10.71	t.o.	t.o.	53.30	600	121	21.90
600 s.	t.o.	t.o.	320.50	600	288	9.55	t.o.	t.o.	412.77	600	225	15.01
	Pack (55 instances, all feasible)											
6 s.	5.82	t.o.	3.36	55	31	5.11	t.o.	t.o.	4.97	55	15	6.86
60 s.	5.81	t.o.	22.25	55	38	4.25	t.o.	t.o.	36.78	55	26	5.58
600 s.	5.82	t.o.	166.65	55	41	3.87	203.05	t.o.	296.26	55	30	5.14
	Pack_d (55 instances, all feasible)											
6 s.	t.o.	t.o.	5.70	55	3	19.92	t.o.	t.o.	4.37	55	18	7.87
60 s.	t.o.	t.o.	48.23	55	19	8.91	t.o.	t.o.	37.26	55	25	4.61
600 s.	166.35	t.o.	287.40	55	34	3.65	118.73	t.o.	276.65	55	32	3.74

Table 14
Measures for the datasets of the MRCPSP with increasing cutoffs of 6, 60 and 600 s.

	Time-indexed						Task-indexed					
	med	Q3	avg	#s	#c	Δ_{LB}	med	Q3	avg	#s	#c	Δ_{LB}
	j30 (640 instances, 552 feasible)											
6 s.	0.51	0.76	1.01	640	596	2.76	0.36	0.79	1.19	640	562	4.00
60 s.	0.50	0.74	3.91	640	613	1.93	0.36	0.79	6.16	640	590	2.63
600 s.	0.52	0.75	15.97	640	630	0.71	0.36	0.79	36.47	640	613	1.69
	MMLIB50 (540 instances, all feasible)											
6 s.	2.01	t.o.	3.13	540	362	15.69	4.02	t.o.	3.89	540	312	23.12
60 s.	2.02	16.04	15.13	540	437	7.55	4.10	42.08	19.21	540	417	9.12
600 s.	2.01	15.90	106.53	540	452	6.04	4.08	41.07	126.94	540	437	6.71
	MMLIB100 (540 instances, all feasible)											
6 s.	t.o.	t.o.	5.21	493	203	232.63	t.o.	t.o.	5.94	493	18	281.57
60 s.	26.90	t.o.	32.45	517	283	120.90	t.o.	t.o.	45.12	517	210	135.58
600 s.	26.45	t.o.	275.68	531	305	61.16	273.35	t.o.	317.50	531	283	65.71

Table 15
Measures for the datasets of the RCPSP/t with increasing cutoffs of 6, 60 and 600 s.

	med	Q3	avg	#s	#c	Δ_{LB}
	j30 (2880 instances, 2826 feasible)					
6 s.	0.15	0.38	0.33	2880	2876	0.05
60 s.	0.15	0.37	0.34	2880	2880	0.00
600 s.	0.15	0.37	0.34	2880	2880	0.00
	j120 (3600 instances, all feasible)					
6 s.	t.o.	t.o.	5.09	3595	1032	76.03
60 s.	42.22	t.o.	34.50	3600	1932	28.62
600 s.	41.82	t.o.	240.01	3600	2292	12.14

methods to compute a first solution explained in Section 7 play a very important role. The number of certified instances is more sensible to the cutoff, and reduces to 75.1% with 60 s and to 64% with 6 s. However, this reduction is not uniform but depends on the considered problem: in RCPSP we are still able to certify 72.3% of the instances with 6 s, and moreover the overall quality of the makespans is similar to that obtained with 600 s; in MRCPSP and RCPSP/t we certify 67.5% and 60.3% of the instances respectively, and obtain noticeably worse makespans in the hardest datasets.

The number of certified solutions and the quality of the makespans with smaller cutoffs is still generally better with time-indexed than with task-indexed formulations. However, in set Pack_d of RCPSP, the task-indexed formulation behaves dramatically better than the time-indexed one.

Table 16
Measures for the datasets of the RCPSP with LCG, FDS and SMT.

	med	Q3	avg	#s	#c	Δ_{LB}
	j30 (480 instances, all feasible)					
LCG	–	–	–	480	480	0.00
FDS	0.01	0.03	0.93	480	480	0.00
SMT	0.03	0.07	0.22	480	480	0.00
	j60 (480 instances, all feasible)					
LCG	–	–	–	480	432	2.17
FDS	0.02	0.18	67.44	480	430	1.91
SMT	0.12	0.52	61.90	480	435	1.88
	j90 (480 instances, all feasible)					
LCG	–	–	–	480	400	3.47
FDS	0.03	0.39	106.94	480	399	2.63
SMT	0.32	1.62	104.59	480	400	3.24
	j120 (600 instances, all feasible)					
LCG	–	–	–	600	283	9.76
FDS	t.o.	t.o.	322.52	600	285	7.02
SMT	t.o.	t.o.	320.50	600	288	9.55
	Pack (55 instances, all feasible)					
LCG	–	–	–	55	39	4.23
FDS	2.72	t.o.	177.07	55	40	4.00
SMT	5.82	t.o.	166.65	55	41	3.87
	Pack_d (55 instances, all feasible)					
LCG	–	–	–	55	37	3.25
FDS	18.20	t.o.	191.70	55	39	3.09
SMT	166.35	t.o.	287.40	55	34	3.65

Table 17
Measures for the datasets of the MRCPSP with LCG, FDS and SMT.

	med	Q3	avg	#s	#c	Δ_{LB}
	j30 (640 instances, 552 feasible)					
LCG	0.22	0.56	29.58	640	615	1.56
FDS	0.08	6.44	38.60	640	625	0.75
SMT	0.52	0.75	15.97	640	630	0.71
	MMLIB50 (540 instances, all feasible)					
LCG	6.59	338.40	162.39	540	425	8.79
FDS	1.33	t.o.	173.45	509	400	92.83
SMT	2.01	15.90	106.53	540	452	6.04
	MMLIB100 (540 instances, all feasible)					
LCG	446.61	t.o.	326.18	529	273	159.00
FDS	36.55	t.o.	280.66	462	301	340.64
SMT	26.45	t.o.	275.68	531	305	61.16

8.3. Comparison with state-of-the-art

To the best of our knowledge, as said in Section 2, the works where best results with exact solvers have been reported for the studied problems are the following:

- The system presented in Vilím et al. (2015), for RCPSP and MRCPSP, based on *Failure Directed Search* (FDS). We will refer to this system as **FDS**.
- The system presented in Schutt, Feydy, and Stuckey (2013), for RCPSP, based on Lazy Clause Generation (LCG). We contacted the authors and they kindly shared with us some tools and instructions for solvers for other problems different from the RCPSP. However, they were unable to give us the system used for the RCPSP in Schutt, Feydy, and Stuckey (2013), and therefore we have been unable to run their system in our machine for an accurate comparison. We only report in Table 16 the best numbers of solved and certified instances available on the website[3] as well as the makespan quality measure Δ_{LB}. We refer to this system in the results for RCPSP as **LCG**. They used a machine running GNU/Linux and an Intel® Core™ i7 CPU processor at 2.8 GHz, setting a cutoff of 600 s like us.
- The system presented in Szeredi and Schutt (2016), for MRCPSP, based on LCG. We refer to this system in the results for the MRCPSP as **LCG**.

In Tables 16 and 17 we report all the measures of the solving times and quality of makespan for the RCPSP and the MRCPSP respectively (with the exception of LCG for the RCPSP, as mentioned above). They have been run in the same machine as our solvers, and using the exact configuration that the authors described in each corresponding work. We also include in these tables the results of our system with time-indexed and PB(AMO) encodings (referred to as **SMT**), which are generally the best among all the considered SMT encodings. We do not provide any comparison with existing exact solvers for the RCPSP/t since, up to our knowledge, there is none.

As we can see, in general, the SMT approach is the best one with respect to the number of certified instances and average solving time. In particular, in RCPSP, with SMT we are able to certify 7 and 5 more instances than with LCG and FDS, respectively. Notice however that in Pack_d FDS is the best approach while SMT, which suffers from large makespans, is the worst one. The advantage of SMT is more evident in

[3] Their results are publicly available in https://people.eng.unimelb.edu.au/pstuckey/rcpsp/.

MRCPSP datasets. It solves 100 more instances than FDS, and overall it is able to certify 80.6% of the instances, whilst LCG and FDS certify 76.3% and 77.1% of the instances, respectively. The fact that SMT is much better in the MRCPSP may be probably related to the fact that resource constraints are more involved than in the RCPSP due to the modes of the activities. This complication is somehow absorbed by PB(AMO)s mitigating the increase in size of the encodings of those resource constraints. On the other hand, in two MRCPSP datasets, the reported median is slightly worse in SMT than in FDS. The time needed to build the SMT formula is surely penalizing the solving time of easy instances. Regarding Δ_{LB}, SMT is the best approach in MRCPSP with dramatic differences in some cases, whilst in RCPSP the comparison is more tight, being FDS the best in sets j90, j120 and Pack_d, and SMT the best in j60 and Pack.

Regarding RCPSP/t, to our knowledge no previous results of exact methods have been published. Nevertheless, the problem was introduced and tackled with heuristic methods in Hartmann (2013, 2015). From the 6480 existing instances, we have certified the optimality, or infeasibility for the time horizon specified, of 5172 instances, and improved the upper bound of the makespan of about 1800 instances with respect to those heuristic methods.

9. Conclusions

In this work we have presented different SMT formulations for the RCPSP, the MRCPSP and the RCPSP/t, that can be integrated in a solving process using any off-the-shelf SMT solver. In particular, we provide a time-indexed formulation for the three problems, and a task-indexed formulation for the RCPSP and the MRCPSP. The high expressiveness of the SMT language, together with the use of PB constraints that are later encoded to SAT/SMT, let us provide both concise and efficient formulations. The first thing to highlight is that the provided time-indexed formulations are clearly better than the task-indexed formulations in almost all the representative benchmark sets of the literature. This is true even though the size of time-indexed formulations depends on the given scheduling horizon, and this usually means that time-indexed encodings require more clauses and variables than task-indexed ones. In this regard, the use of preprocesses to quickly compute a first upper bound of the makespan is essential. Nevertheless it is also true that the common benchmarks, excepting Pack_d for RCPSP, contain activities with short durations, and hence we can find a short upper bound of the makespan. However this may be not the case in some industrial instances. Moreover, in set Pack_d, the task-indexed formulation behaves significantly better than the time-indexed one when we set small cutoffs. Therefore it is worthwhile to provide also efficient task-indexed formulations.

It is remarkable that our exact solving system is able to certify the optimality or unfeasibility of a large amount of instances. This is done with the relatively small computation time of 10 min, that can be afforded in many industrial applications where schedules are not computed in real time but beforehand. Moreover, we have shown that our system is also able to provide solutions with fairly good makespans even with little computation time.

A key point in the efficiency of our systems is the use of SAT encodings of PB(AMO) constraints. This technique lets us reduce considerably the sizes of the obtained encodings compared to using SAT encodings of PB constraints. As a result, the performance obtained with PB(AMO) encodings is significantly better than the performance using classic SAT encodings of PB constraints or LIA expressions. The use of PB(AMO) encodings is only possible thanks to being able to identify (usually implicit) AMO constraints over the variables of PB constraints. We have presented a quick method to compute partitions of the variables into implied AMO constraints, based on path covers of the precedence graph of the RCPSP. We have shown how to find even better implied AMO constraints for the particular variants MRCPSP and RCPSP/t. Moreover, the given template to build up AMO partitions from path covers could be easily adaptable to many other RCPSP variants. In the end, the systems that we present are situated on top of the state-of-the-art of exact methods for the studied problems.

Finally, we have shown that SMT is an efficient technique to exactly solve scheduling problems. Therefore we think it is worthwhile to study the formulation of other scheduling problems into SMT, such that problems with preemptive activities or different objective functions.

CRediT authorship contribution statement

Miquel Bofill: Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Supervision, Validation, Writing - review & editing. **Jordi Coll:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Writing - original draft, Writing - review & editing. **Josep Suy:** Conceptualization, Formal analysis, Investigation, Methodology, Software, Supervision, Validation, Writing - review & editing. **Mateu Villaret:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Supervision, Validation, Writing - original draft, Writing - review & editing.

Acknowledgments

This work was supported by *Ayudas para Contratos Predoctorales 2016*, funded by MINECO and co-funded by FSE (grant number BES-2016-076867), and by MICINN/FEDER, UE (grant number RTI2018-095609-B-I00).

References

Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., & Mayer-Eichberger, V. (2012). A new look at BDDs for pseudo-boolean constraints. *Journal of Artificial Intelligence Research*, *45*, 443–480.

Ansótegui, C., Bofill, M., Palahí, M., Suy, J., & Villaret, M. (2011). Satisfiability Modulo Theories: An Efficient Approach for the Resource-Constrained Project Scheduling Problem. In *Proceedings of the ninth symposium on abstraction, reformulation, and approximation*. (pp. 2–9).

Artigues, C. (2013). A note on time-indexed formulations for the resource-constrained project scheduling problem.

Artigues, C., Demassey, S., & Neron, E. (2013). *Resource-constrained project scheduling: Models, algorithms, extensions and applications*. John Wiley & Sons.

Audemard, G., & Simon, L. (2009). Glucose: a solver that predicts learnt clauses quality. *SAT Competition*, 7–8.

Bailleux, O., Boufkhad, Y., & Roussel, O. (2009). New encodings of pseudo-boolean constraints into CNF. In *LNCS: vol. 5584, Proceedings of the 12th international conference on theory and applications of satisfiability testing* (pp. 181–194). Springer.

Barrett, C., Fontaine, P., & Tinelli, C. (2016). The satisfiability modulo theories library (SMT-LIB). www.SMT-LIB.org.

Barrett, C., & Tinelli, C. (2018). *Handbook of model checking* (pp. 305–343). Springer International Publishing, Ch.

Bellenguez-Morineau, O. (2008). *Methods to solve multi-skill project scheduling problem* (Ph.D. thesis), Université François-Rabelais de Tours.

Biere, A., Heule, M., & van Maaren, H. (2009). *Handbook of satisfiability (vol. 185)*. IOS Press.

Blazewicz, J., Lenstra, J. K., & Kan, A. R. (1983). Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, *5*(1), 11–24.

Bofill, M., Coll, J., Suy, J., & Villaret, M. (2016). Solving the Multi-Mode Resource-Constrained Project Scheduling Problem with SMT. In *Proceedings of the 28th IEEE international conference on tools with artificial intelligence*. (pp. 239–246).

Bofill, M., Coll, J., Suy, J., & Villaret, M. (2017a). Compact MDDs for Pseudo-Boolean Constraints with At-Most-One Relations in Resource-Constrained Scheduling Problems. In *Proceedings of the 26th international joint conference on artificial intelligence*. (pp. 555–562).

Bofill, M., Coll, J., Suy, J., & Villaret, M. (2017b). An efficient SMT approach to solve MRCPSP/max Instances with tight constraints on resources. In *LNCS: vol. 10416, Proceedings of the 23rd international conference on principles and practice of constraint programming* (pp. 71–79). Springer.

Bofill, M., Coll, J., Suy, J., & Villaret, M. (2019). SAT Encodings of Pseudo-Boolean Constraints with At-Most-One Relations. In *Proceedings of the 16th International conference on the integration of constraint programming, artificial intelligence, and operations research*. (pp. 112–128).

Brucker, P., Drexl, A., Möhring, R., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, *112*(1), 3–41.

Brucker, P., & Knust, S. (2012). *Complex scheduling*. Springer Berlin Heidelberg, http://dx.doi.org/10.1007/978-3-642-23929-8_1.

Carlier, J., & Néron, E. (2003). On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research*, *149*(2), 314–324.

Cheng, Z., Zhang, H., Tan, Y., & Lim, Y. (2016). SMT-based scheduling for multiprocessor real-time systems. In *2016 IEEE/ACIS 15th international conference on computer and information science*. (pp. 1–7).

de Moura, L. M., & Bjørner, N. (2008). Z3: An efficient SMT solver. In *LNCS*: *vol. 4963*, *Proceedings of the 14th international conference on tools and algorithms for the construction and analysis of systems* (pp. 337–340). Springer.

Dutertre, B., & De Moura, L. (2006). A fast linear-arithmetic solver for DPLL(T). In *International conference on computer aided verification* (pp. 81–94). Springer.

Dutertre, B., & de Moura, L. (2006). *The yices SMT solver*: *Tech. rep*, Computer Science Laboratory, SRI International, Available at http://yices.csl.sri.com.

Eén, N., & Sorensson, N. (2006). Translating pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, *2*, 1–26.

Erkinger, C., & Musliu, N. (2017). Personnel scheduling as satisfiability modulo theories. In *Proceedings of the twenty-sixth international joint conference on artificial intelligence* (pp. 614–621). http://dx.doi.org/10.24963/ijcai.2017/86.

Fazekas, K., Biere, A., & Scholl, C. (2019). Incremental inprocessing in SAT solving. In *International conference on theory and applications of satisfiability testing* (pp. 136–154). Springer.

Feydy, T., & Stuckey, P. J. (2009). Lazy clause generation reengineered. In *Proceedings of the 15th international conference on principles and practice of constraint programming* (pp. 352–366). Springer.

Hartmann, S. (2013). Project scheduling with resource capacities and requests varying with time: a case study. *Flexible Services and Manufacturing Journal*, *25*(1–2), 74–93.

Hartmann, S. (2015). Time-varying resource requirements and Capacities. In *Handbook on project management and scheduling (vol. 1)* (pp. 163–176). Springer.

Hartmann, S., & 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.

Hölldobler, S., Manthey, N., & Steinke, P. (2012). A compact encoding of pseudo-boolean constraints into SAT. In *LNCS*: *vol. 7526*, *KI 2012: Advances in artificial intelligence - 35th annual german conference on artificial intelligence* (pp. 107–118). Springer.

Hopcroft, J. E., & Karp, R. M. (1973). An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, *2*(4), 225–231.

Joshi, S., Martins, R., & Manquinho, V. M. (2015). Generalized totalizer encoding for pseudo-boolean constraints. In *LNCS*: *vol. 9255*, *Proceedings of the 21st international conference on principles and practice of constraint programming* (pp. 200–209). Springer.

Kelley, J. E. (1963). The critical-path method: Resources planning and scheduling. *Industrial Scheduling*, *13*, 347–365.

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

Kolisch, R., & Drexl, A. (1997). Local search for nonpreemptive multi-mode resource-constrained project scheduling. *IIE Transactions*, *29*(11), 987–999.

Kolisch, R., & Sprecher, A. (1997). PSPLIB - A Project scheduling problem library. *European Journal of Operational Research*, *96*(1), 205–216.

Koné, O., Artigues, C., Lopez, P., & Mongeau, M. (2011). Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, *38*, 3–13.

Li, C.-M., Xiao, F., Luo, M., Manyà, F., Lü, Z., et al. (2020). Clause vivification by unit propagation in CDCL SAT solvers. *Artificial Intelligence*, *279*, Article 103197.

Malik, A., Walker, C., O'Sullivan, M., & Sinnen, O. (2018). Satisfiability modulo theory (SMT) formulation for optimal scheduling of task graphs with communication delay. *Computers & Operations Research*, *89*, 113–126. http://dx.doi.org/10.1016/j.cor.2017.08.012.

Manthey, N., Philipp, T., & Steinke, P. (2014). A more compact translation of pseudo-boolean constraints into CNF such that generalized arc consistency is maintained. In *LNCS*: *vol. 8736*, *KI 2014: Advances in artificial intelligence - 37th annual german conference on AI* (pp. 123–134).

Marques-Silva, J. P., & Sakallah, K. A. (1999). GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, *48*(5), 506–521.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual design automation conference* (pp. 530–535). ACM.

Nieuwenhuis, R., & Oliveras, A. (2005). DPLL(T) with exhaustive theory propagation and its application to difference logic. In *International conference on computer aided verification* (pp. 321–334). Springer.

Ntafos, S. C., & Hakimi, S. L. (1979). On path cover problems in digraphs and applications to program testing. *IEEE Transactions on Software Engineering*, *5*, 520–529.

Patterson, J. H., & Huber, W. D. (1974). A horizon-varying, zero-one approach to project scheduling. *Management Science*, *20*(6), 990–998.

Pellerin, R., Perrier, N., & Berthaut, F. (2019). A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*.

Pritsker, A. A. B., Watters, L. J., & Wolfe, P. S. (1996). Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, *16*, 93–108.

Roselli, S. F., Bengtsson, K., & Åkesson, K. (2018). SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation. In *2018 IEEE 14th international conference on automation science and engineering* (pp. 547–552). IEEE.

Schnell, A., & Hartl, R. F. (2016). On the efficient modeling and solution of the multi-mode resource-constrained project scheduling problem with generalized precedence relations. *OR Spectrum*, *38*(2), 283–303.

Schutt, A., Feydy, T., & Stuckey, P. J. (2013). Explaining time-table-edge-finding propagation for the cumulative resource constraint. In *Proceedings of the 10th international conference on AI and OR techniques in constraint programming for combinatorial optimization problems* (pp. 234–250). Springer.

Schutt, A., Feydy, T., Stuckey, P. J., & Wallace, M. (2009). Why cumulative decomposition is not as bad as it sounds. In *LNCS*: *vol. 5732*, *Proceedings of the 15th international conference on principles and practice of constraint programming* (pp. 746–761). Springer.

Schutt, A., Feydy, T., Stuckey, P. J., & Wallace, M. G. (2013). Solving RCPSP/max by Lazy Clause Generation. *Journal of Scheduling*, *16*(3), 273–289.

Sebastiani, R., & Tomasi, S. (2012). Optimization in SMT with LA(Q) cost functions. In *LNCS*: *vol. 7364*, *Proceedings of the 6th international joint conference on automated reasoning* (pp. 484–498). Springer.

Sebastiani, R., & Trentin, P. (2015). OptiMathSAT: A tool for optimization modulo theories. In *Proceedings of the 27th international conference on computer aided verification* (pp. 447–454). Springer.

Stinson, J. P., Davis, E. W., & Khumawala, B. M. (1978). Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions*, *10*(3), 252–259.

Suy, J. (2013). *A satisfiability modulo theories approach to constraint programming* (Ph.D. thesis), Universitat de Girona.

Szeredi, R., & Schutt, A. (2016). Modelling and solving multi-mode resource-constrained project scheduling. In *Proceedings of the 22nd international conference on principles and practice of constraint programming* (pp. 483–492). Springer.

Talbot, F. B., & Patterson, J. H. (1978). An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. *Management Science*, *24*(11), 1163–1174.

Toffolo, T. A., Santos, H. G., Carvalho, M. A., & Soares, J. A. (2016). An integer programming approach to the multimode resource-constrained multiproject scheduling problem. *Journal of Scheduling*, *19*(3), 295–307.

Van Peteghem, V., & Vanhoucke, M. (2014). An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *European Journal of Operational Research*, *235*(1), 62–72.

Vilím, P. (2011). Timetable edge finding filtering algorithm for discrete cumulative resources. In *Proceedings of the 18th international conference on AI and OR techniques in constraint programming for combinatorial optimization problems* (pp. 230–245). Springer.

Vilím, P., Laborie, P., & Shaw, P. (2015). Failure-directed search for constraint-based scheduling. In *Proceedings of the 12th integration of AI and OR techniques in constraint programming* (pp. 437–453). Springer.