Solving the Multi-Mode Resource-Constrained Project Scheduling Problem with SMT

Miquel Bofill, Jordi Coll, Josep Suy and Mateu Villaret
Departament d'Informàtica, Matemàtica Aplicada i Estadística
Universitat de Girona, Spain
Email: {miquel.bofill,jordi.coll,josep.suy,mateu.villaret}@imae.udg.edu

***Abstract*—The Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP) is a generalization of the well known Resource-Constrained Project Scheduling Problem (RCPSP). The most common exact approaches for solving this problem are based on branch-and-bound algorithms, mixed-integer linear programming and Boolean satisfiability (SAT). In this paper, we present a new exact approach for solving this problem, using Satisfiability Modulo Theories (SMT). We provide two encodings into SMT and several reformulation and preprocessing techniques. The optimization algorithm that we propose uses an SMT solver as an oracle, and depending on its answer is able to update the encoding for the next optimization step. We report extensive performance experiments showing the utility of the proposed techniques and the good performance of our approach that allows us to close several open instances.**

I. Introduction

Solving methods which use learning, and conflict-driven search based heuristics, are becoming state-of-the-art for combinatorial problems and, in particular, are extremely well-suited for scheduling problems [1], [2], [3]. In this work we report our experience using Satisfiability Modulo Theories (SMT) for solving the Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP)[1].

MRCPSP is an extension of the Resource-Constrained Project Scheduling Problem (RCPSP). RCPSP consists on scheduling a set of non-preemptive activities which consume some renewable resources while minimizing the completion time. This schedule must satisfy some precedence requirements between activities and the resource capacities can never be exceed over time. In the MRCPSP every activity has a number, greater or equal to 1, of execution modes. Each activity mode is described with a pair, formed by the duration of the activity and a vector of resource demands in this mode. Also, in the MRCPSP one distinguishes between renewable resources and non-renewable resources: renewable resources are replenished at each time unit, while for non-renewable ones, resource usage is accumulated across the entire project. For example, a renewable resource could be man-hours per day, or the capacity of a machine, while a non-renewable resource could be a budget, or some kind of stock. The objective of the MRCPSP is to find a mode and a start time for each activity, such that the makespan is minimized and the schedule is feasible with respect to the precedence and (renewable and non-renewable) resource constraints.

The MRCPSP is NP-hard [6]. Several exact and heuristic approaches to solve the MRCPSP have been proposed in the last years. The most common exact approaches for solving this problem are based on branch-and-bound [7], [8] and mixed-integer linear programming (MILP) formulations [9], [10], [11]. There exists a non exact hybrid method that uses MILP and Boolean satisfiability (SAT) [12]. In [3] we found an exact hybrid system using large neighbourhood search and failure-directed search which was, as far as we know, the one closing more MRCPSP instances. However, there does not exist one using satisfiability modulo theories (SMT).

In a previous work [2], we presented an SMT based system to tackle the RCPSP, concluding that SMT is competitive with the state-of-the art solvers for this problem. It is worth noting that the MRCPSP can be naturally modeled using Boolean combinations of arithmetic constraints, such as implications (i.e., disjunctions) related to activity modes. Those models fit perfectly into the SMT language. Moreover, modern SMT solvers are specifically designed to deal efficiently with Boolean combinations of arithmetic predicates. Hence, we can use an off-the-shelf SMT solver without modifying it. For this reason, we propose a model-and-solve approach for solving the MRCPSP using SMT.

We provide two distinct encodings based on a well-known approach of the literature, also used in [13], [14]. Basically, this approach consists in defining a constraint over each renewable resource for every discrete time instant. In the first encoding we provide, we use linear integer arithmetic constraints to bound resource consumption, while in the other one, we bound resource consumption with Binary Decision Diagrams that we translate into plain SAT clauses. A number of preprocessing steps are performed to obtain lower and upper bounds, and compute time windows of activities. We also introduce a preprocessing step that allows us to reduce the demand of non-renewable resources for each activity and a simplified method for checking infeasibility.

The optimization algorithm we have developed, successively calls the decision procedure, bounding the cost function, until the optimum is found. This procedure also tries to reformulate the SMT formula by narrowing the time windows of activities after each iterative step as the search advances. The reformulation preserves the internal search state of the SMT solver

[1]This problem is denoted as $MPS|prec|C_{max}$ in [4] and as $m,1T|cpm,disc.mu|C_{max}$ in [5].

between successive calls, thus taking profit of the learning capabilities of the CDCL(T) SMT solvers.

In order to show the efficiency of the presented solution, we report very good results of the experiments conducted on the most challenging MRCPSP instances from the PSPLib [15] and on some instances from MMLIB [16]. Namely, in Section VI we report the results on the j30 set, because that is the only one with open instances in PSPLib and that our system is unable to completely close, and on the MMLIB50 set from MMLIB, which contains harder instances and many of them remain open. We compare our results with [3] which, to the best of our knowledge, is currently state-of-the-art. Our system and the detailed experiments can be found at http://ima.udg.edu/Recerca/lap/mrcpsp/mrcpsp.html.

The rest of this paper is organized as follows. In Section II we formally define the MRCPSP. In Section III we describe different preprocessing steps. In Section IV we describe our SMT formulations for the MRCPSP. In Section V we describe our optimization algorithm. In Section VI we provide the results of our experiments, and conclude in Section VII by pointing out some future work.

II. The Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP)

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

- $V = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ is a set of activities. Activities A_0 and A_{n+1} are dummy activities representing, by convention, the start and the end of the schedule, respectively. The set of non-dummy activities is defined by $A = \{A_1, \ldots, A_n\}$.
- $M \in \mathbb{N}^{n+2}$ is a vector of naturals, being M_i the number of modes that activity i can execute, with $M_0 = M_{n+1} = 1$ and $M_i \geq 1, \forall A_i \in A$.
- p is a vector of vectors of naturals, being $p_{i,o}$ the duration of activity i using mode o, with $1 \leq o \leq M_i$. For the dummy activities, $p_{0,1} = p_{n+1,1} = 0$, and $p_{i,o} > 0, \forall A_i \in A, 1 \leq o \leq M_i$.
- E is a set of pairs of activities representing precedence relations. Concretely, $(A_i, A_j) \in E$ iff the execution of activity A_i must precede that of activity A_j, i.e., activity A_j must start after activity A_i has finished.
  We assume that we are given a precedence activity-on-node graph $G(V, E)$ that contains no cycles, since otherwise the precedence relation is inconsistent. We assume that E is such that A_0 is a predecessor of all other activities and A_{n+1} is a successor of all other activities.
- $R = \{R_1, \ldots, R_{v-1}, R_v, R_{v+1}, \ldots, R_q\}$ is a set of resources. The first v resources are renewable, and the last $q - v$ resources are non-renewable.
- $B \in \mathbb{N}^q$ is a vector of naturals, being B_k the available amount of each resource R_k. The first v resource availabilities correspond to the renewable resources, while the last $q-v$ ones correspond to the non-renewable resources.
- b is a matrix of naturals corresponding to the resource demands of activities per mode. $b_{i,k,o}$ represents the amount of resource R_k used during the execution of activity A_i in mode o. Note that $b_{0,k,1} = 0$ and $b_{n+1,k,1} = 0, \forall k \in \{1, \ldots, q\}$.

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 A_i. We assume 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, satisfying $1 \leq SM_i \leq M_i$, denotes the mode of each activity A_i. A solution of the MRCPSP problem is a schedule of modes SM and a feasible schedule S of minimal makespan S_{n+1}. The MRCPSP can hence be formulated as

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

subject to the following precedence and resource constraints:

$$\begin{gathered} S_0 = 0 \\ (SM_i = o) \rightarrow (S_j - S_i \geq p_{i,o}) \\ \forall (A_i, A_j) \in E, \forall o \in \{1, \ldots, M_i\} \end{gathered} \tag{2}$$

$$\left(\sum_{A_i \in A} \sum_{o \in \{1,\ldots,M_i\}} ite(SM_i = o;\, b_{i,k,o};\, 0) \right) \leq B_k \qquad \forall R_k \in \{R_{v+1}, \ldots, R_q\} \tag{3}$$

$$\left(\sum_{A_i \in A} \sum_{o \in \{1,\ldots,M_i\}} ite\big((SM_i = o) \wedge (S_i \leq t) \wedge (t < S_i + p_{i,o});\, b_{i,k,o};\, 0\big) \right) \leq B_k \qquad \forall R_k \in \{R_1, \ldots, R_v\}, \forall t \in H \tag{4}$$

where $ite(c;\, e_1;\, e_2)$ is an *if-then-else* expression denoting e_1 if c is true and e_2 otherwise, $H = \{0, \ldots, T\}$ is the scheduling horizon, and T (the length of the scheduling horizon) is an upper bound for the makespan.

We also have to force the execution mode to be correct:

$$1 \leq SM_i \leq M_i \qquad \forall A_i \in A \tag{5}$$

A schedule S is feasible if it satisfies the precedence constraints (2), the non-renewable resource constraints (3), the renewable resource constraints (4) and the execution mode correctness constraints (5). An example is shown in Figure 1.

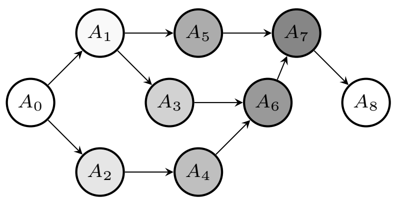
A problem instance has no feasible schedules if and only if some of the following conditions holds:

- There exists a cycle in the precedences graph (i.e., an activity is forced to start after itself finishes).
- There exists some activity whose demand over a resource in all execution modes is greater than its capacity.
- There is no schedule of modes such that all the non-renewable resource constraints (3) are satisfied.

The two first conditions are verifiable in polynomial time. These are typically satisfied in the benchmark instances available in the literature. Hence, having checked them, we propose in Subsection III-C to focus in the the third condition to detect the infeasibility of an instance.

Instance:

A_i	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
$p_{i,1}$	0	2	4	3	1	3	3	2	0
$b_{i,1,1}$	0	3	1	1	2	2	1	2	0
$b_{i,1,2}$	-	1	1	1	1	1	1	2	-
$p_{i,2}$	-	4	1	1	1	1	2	1	-
$b_{i,2,1}$	0	1	3	2	1	0	0	3	0
$b_{i,2,2}$	-	3	0	1	2	4	2	0	-



Solution:

A_i	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
S_i	0	0	2	3	4	4	5	7	8
SM_i	1	1	2	2	2	1	2	2	1

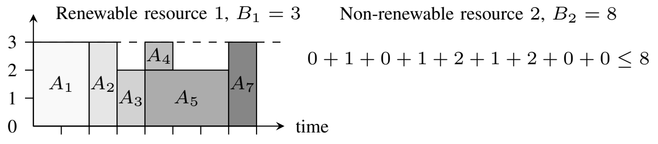


Fig. 1. An MRCPSP instance and its solution. The first table contains the durations and the resource (one renewable and one non-renewable) demands for each activity mode. The graph represents the precedences. The second table contains an optimum schedule $makespan = 8$. Finally, it is represented the schedule and the validity of the resources usage.

III. Preprocessing

We perform standard preprocessing steps to compute the extended precedence set, a lower and an upper bound for the makespan, and time windows for each activity. We introduce a new preprocessing step called *non-renewable resource demand reduction*, which allows us to reduce the size of the constraints related to non-renewable resources. All these preprocesses are implemented in order to extract some features that will allow us to reduce the size of the encodings and the search space.

A. Extended precedence set

Since a precedence is a transitive relation, we can compute a lower bound on the time between each pair of activities in E. For this calculation we use the Floyd-Warshall algorithm on the graph defined by the precedence relation E, where each arc (A_i, A_j) is labeled with the duration $\min_{o\in\{1,\ldots,M_i\}}(p_{i,o})$. This extended precedence set is named E^* and contains, for each pair of activities A_i and A_j such that A_i precedes A_j, a tuple of the form $(A_i, A_j, l_{i,j})$ where $l_{i,j}$ is the length of the longest path from A_i to A_j. Note that this longest path length is minimal with respect to the different activity modes. Note also that, if $(A_i, A_i, l_{i,i}) \in E^*$ for some A_i and $l_{i,i} > 0$, then there is a cycle in the precedence relation and therefore the problem instance is inconsistent.

In fact the demands on renewable resources let us go one step further. Note that any activity A_k such that $(A_i, A_k, l_{i,k}) \in E^* \wedge (A_k, A_j, l_{k,j}) \in E^*$ will be completely executed in the time interval $[S_i + \min_{o\in\{1,\ldots,M_i\}}(p_{i,o}), S_j]$. Hence, this interval must be wide enough to run all such A_k without exceeding the availability of any renewable resource. This time interval gives, for every resource $r \in \{1,\ldots,v\}$, a lower bound $(RLB_{i,j,r})$ of the time difference between the end of A_i and the start of A_j:

$$RLB_{i,j,r} = \lceil \frac{1}{B_r} * \sum_{\substack{A_k\in A\\(A_i,A_k,l_{i,k})\in E^*\\(A_k,A_j,l_{k,j})\in E^*}} \min_{o\in\{1,\ldots,M_k\}}(p_{k,o} * b_{k,r,o})\rceil$$

So we can update the extended precedence set as:

$$l'_{i,j} = \max(l_{i,j}, \min_{o\in\{1,\ldots,M_i\}}(p_{i,o}) + \max_{r\in\{1,\ldots,v\}}(RLB_{i,j,r})) \qquad \forall(A_i, A_j, l_{i,j}) \in E^*$$

where $l_{i,j}$ is the value obtained by transitivity on the precedence set, and $l'_{i,j}$ is the updated value. Note that an increase of a single extended precedence can be propagated to other precedences in E^*. We achieve this propagation with just a new execution of the Floyd-Warshall algorithm. This preprocessing resembles the energy based reasoning used by some constraint propagators (see [17]).

B. Lower Bound

A lower bound LB for the makespan is a lower bound for the start time of activity A_{n+1}. The critical path (i.e. the maximum length path) between the initial activity A_0 and the final activity A_{n+1} in the precedence graph is a lower bound for the makespan. Note that we can easily know the length of this path if we have already computed the extended precedence set, since it corresponds to the value $l_{0,n+1}$ in the tuple $(A_0, A_{n+1}, l_{0,n+1}) \in E^*$.

For instance, in Figure 1 the critical path is $[A_0, A_1, A_3, A_6, A_7, A_8]$ with modes $1, 1, 2, 2, 2, 1$, respectively, and its length is 6. Hence, we have $LB = 6$.

Moreover, there exists the possibility that $l'_{0,n+1}$ has been increased due to the renewable resource demands, thus obtaining a better lower bound.

C. Upper Bound

An upper bound UB for the makespan is an upper bound for the start time of activity A_{n+1}.

We compute an upper bound in two steps. First of all we find a feasible schedule of modes for the instance, by means of a satisfiability check of constraints (3) alone on the SMT solver. As pointed out in Section II, it is a necessary condition for an instance to have a feasible solution that we can find such a schedule of modes. If we succeed in this first check, and having tested the other trivial feasibility conditions exposed in Section II, we can conclude that the instance has feasible solutions. In the second step, given a feasible schedule of modes, we run an algorithm based on a fast heuristic method to find a (presumably non-optimal) solution, and use its makespan as the upper bound. We use the parallel scheduling generation scheme (parallel SGS) proposed in [18] and described in [19].

D. Time Windows

We can reduce the domain of each variable S_i (start time of activity A_i), that initially is $\{0 \mathinner{..} UB - \min_{o\in\{1,\ldots,M_i\}}(p_{i,o})\}$, by computing its *time window*. The time window of activity A_i will be $[ES_i, LS_i]$, being ES_i its earliest start time and LS_i its latest start time. To compute the time window we use the lower and upper bound and the extended precedence set, as follows. For activities $A_i, 1 \le i \le n$,

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

and, for activity A_{n+1},

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

For instance, in the example of Figure 1, there is a trivial UB for the makespan equal to 20, given by the sum of the maximum durations of all the activities. Considering this UB, and having $l_{4,8} = 4$, the activity A_4 has time window $[1, 16]$.

E. Non-Renewable Resource Demand Reduction

This preprocessing step consists in reducing the demand of non-renewable resources in a sound way. As we will see, this will allow us to save SMT literals in the constraints related to those kind of resources.

Let us introduce it through an example. In the example of Figure 1, the non-renewable resource R_2 has 8 units available and activity A_6 has two modes: mode 1 requires 1 unit of resource R_2, while mode 2 requires 2 units of the same resource. This problem can be transformed into an equivalent one, where the availability of resource R_2 is 7, and activity A_6 has a demand of 0 units of resource R_2 in mode 1, and of 1 unit in mode 2. Since in mode 1 the demand is of 0 units for this activity, it is not necessary to add any literal considering this mode in the constraints on non-renewable resources. Roughly, following the example, what could be done is to subtract from the availability of resource R_2, and from the different demands of activity A_6 for resource R_2 in each mode, the minimum amount of resource R_2 that activity A_6 needs. However, one could go one step further and, instead of subtracting the minimum demand value, subtract the demand value which most frequently occurs. This of course will lead to negative availabilities and demands. But, interestingly, it allows to reduce the size of the constraints even more (since more demands become zero), while keeping soundness. Details are given below.

For this preprocess, we construct a new vector B' of resource availabilities and a new matrix b' of resource demands, where:

- $B'_k = B_k$ and $b'_{i,k,o} = b_{i,k,o}$, $\forall k \in \{1, \ldots, v\}$, $\forall A_i \in V, \forall o \in \{1, \ldots, M_i\}$.
- For each non-renewable resource R_k and activity A_i, let $max_{k,i}$ denote the demand value for resource R_k with more occurrences in the different modes of activity A_i (and, in case of a tie, the smallest one). Then we state:

$$\begin{array}{ll} b'_{i,k,o} = b_{i,k,o} - max_{k,i} & \forall A_i \in A, \forall o \in \{1, \ldots, M_i\} \\ B'_k = B_k - \displaystyle\sum_{A_i \in A} max_{k,i} & \forall R_k \in \{R_{v+1}, \ldots, R_q\} \end{array}$$

Note that, as said, vector B' and matrix b' range now over integers instead of over naturals, i.e., they can contain some negative values. The zero $b'_{i,k,o}$ values (whose number is maximal thanks to the fact that we subtract the demand value with most occurrences) allow us to simplify the constraints on non-renewable constraints (see Equation 16 below).

IV. Encodings

SAT modulo the theory of *linear integer arithmetic* (LIA) allows us to easily encode MRCPSP instances as logical combinations of arithmetic constraints over integer variables. Namely, with LIA we can deal with linear integer constraints of the form $\sum_{i=1}^{n} a_i x_i \bowtie c$ where a_is and c are integer constants, x_i integer variables and $\bowtie$ any relational operator of $\{<, >, \le, \ge, =\}$. Many SMT solver use specialized Simplex like algorithms [20] to deal with satisfiability checks of conjunctions of LIA atoms (for instance Yices 2 [21]).

We propose two different encodings for the MRCPSP into SAT modulo the theory of *linear integer arithmetic*: *ITE* and *BDD*. The difference between the two encodings is in the formulation of the constraints over resources: *ITE* contains summatories of *if-then-else* expressions, while *BDD* uses Pseudo-Boolean constraints represented as Binary Decision Diagrams which are encoded into plain Boolean clauses.

We also introduce some refinements on the encodings considering the preprocessing steps described in Section III (extended precedences, non-renewable resource demand reduction, etc.)

We use the set of integer variables $\{S_0, S_1, \ldots, S_n, S_{n+1}\}$ to denote the start time of each activity. By S' we refer to the set $\{S_1, \ldots, S_n\}$. We encode the schedule of modes with the set of Boolean variables $\{sm_{i,o} | 0 \le i \le n+1, 1 \le o \le M_i\}$, being $sm_{i,o}$ true if and only if activity A_i is executed in mode o.

The objective function is always (1), and we have the following constraints in both encodings:

$$\begin{array}{lll} S_0 = 0 & & (6) \\ S_i \ge ES_i & \forall A_i \in \{A_1, \ldots, A_{n+1}\} & (7) \\ S_i \le LS_i & \forall A_i \in \{A_1, \ldots, A_{n+1}\} & (8) \\ (sm_{i,o}) \rightarrow (S_j - S_i \ge p_{i,o}) & \forall (A_i, A_j) \in E, & \\ & \forall o \in \{1, \ldots, M_i\} & (9) \\ S_j - S_i \ge l_{i,j} & \forall (A_i, A_j, l_{i,j}) \in E^* & (10) \\ sm_{0,1} = true & & (11) \\ sm_{n+1,1} = true & & (12) \\ \displaystyle\bigvee_{1 \le o \le M_i} sm_{i,o} & \forall A_i \in A & (13) \end{array}$$

$$\neg sm_{i,o} \vee \neg sm_{i,o'} \qquad \forall A_i \in A, 1 \leq o < M_i, \; o < o' \leq M_i \tag{14}$$

where (7) and (8) encode the time windows, (9) encodes the precedences, (10) encodes the extended precedences and (11), (12), (13) and (14) ensure that each activity runs in exactly one mode.

Constraints (3) and (4) on resources are reformulated differently in each of the two encodings, as described below. However, in both cases, for the constraints over renewable resources, we introduce the Boolean variables $y_{i,t}$ denoting whether activity A_i is running at time t:

$$sm_{i,o} \rightarrow (y_{i,t} \leftrightarrow (S_i \leq t) \wedge (t < S_i + p_{i,o})) \quad \forall A_i \in A, \forall o \in \{1, \dots, M_i\}, \forall t \in \{ES_i, \dots, LS_i + p_{i,o}\} \tag{15}$$

A. ITE

This formulation makes uses of *if-then-else* expressions in the constraints over resources. The constraints over the non-renewable are the following:

$$\left(\sum_{A_i \in A} \sum_{\substack{o \in \{1,\dots,M_i\} \\ b'_{i,k,o} \neq 0}} ite\left(sm_{i,o}; b'_{i,k,o}; 0\right) \right) \leq B'_k \quad \forall R_k \in \{R_{v+1}, \dots, R_q\} \tag{16}$$

Notice that the *ite* expression is removed in the cases where $b'_{i,k,o} = 0$ (recall Section III-E). The demands on renewable resources are constrained as follows:

$$\left(\sum_{A_i \in A} \sum_{\substack{o \in \{1,\dots,M_i\} \\ ES_i \leq t \leq LS_i + p_{i,o} - 1 \\ b'_{i,k,o} \neq 0}} ite\left(sm_{i,o} \wedge y_{i,t}; b'_{i,k,o}; 0\right) \right) \leq B'_k \quad \forall R_k \in \{R_1, \dots, R_v\}, \forall t \in \{0, \dots, UB\} \tag{17}$$

B. BDD

This formulation expresses the constraints over resources using Pseudo-Boolean constraints. A Pseudo-Boolean constraint has the form $a_1x_1 + \cdots + a_nx_n \bowtie K$ where the a_i and K are integer coefficients, the x_i are Boolean variables, and the relation operator $\bowtie$ belongs to $\{<, >, \leq, \geq, =\}$. Formally the Boolean variable is interpreted as 1 when it is true, and as 0 when it is false, so that it can multiply the integer coefficient. Our definitions of the constraints over resources perfectly fit this template, and therefore can be modeled with Pseudo-Boolean constraints. We can define the constraints over non-renewable resources as:

$$\left(\sum_{A_i \in A} \sum_{\substack{o \in \{1,\dots,M_i\} \\ b'_{i,k,o} \neq 0}} b'_{i,k,o} \cdot sm_{i,o} \right) \leq B'_k \quad \forall R_k \in \{R_{v+1}, \dots, R_q\} \tag{18}$$

To encode the constraints over renewable resources, we are going to define Boolean variables $x_{i,o,t}$ which are true if and only if A_i runs in mode o at time t:

$$x_{i,o,t} \leftrightarrow (sm_{i,o} \wedge y_{i,t}) \tag{19}$$

The constraint over the renewable resources is:

$$\left(\sum_{A_i \in A} \sum_{\substack{o \in \{1,\dots,M_i\} \\ ES_i \leq t \leq LS_i + p_{i,o} - 1 \\ b'_{i,k,o} \neq 0}} b'_{i,k,o} \cdot x_{i,o,t}) \right) \leq B'_k \quad \forall R_k \in \{R_1, \dots, R_v\}, \forall t \in \{0, \dots, UB\} \tag{20}$$

There are some techniques to fully encode Pseudo-Boolean constraints with Boolean formulas. For this purpose we use Binary Decision Diagrams (BDDs) [22] for the Pseudo-Boolean constraints (18) and (20). Concretely we use the framework developed in [23] based on the ideas presented in [24]. Since this framework only deals with monotone functions, we can only use the reformulation suggested in Subsection III-E that removes the minimum amount resource demand instead of the resource demand occurring more times. If we were removing the resource demand occurring more times, we could get negative resource demands and the Pseudo-Boolean constraint would also contain negative coefficients, resulting in a non monotone function.

Notice also that with this encoding of the resources constraints, the only arithmetic predicates remaining in the *BDD* encoding are the ones expressing precedences, namely (9) and (10), or bounds for integer variables in (7), (8) and (15). Hence, the remaining arithmetic constraints are much simpler than the ones of the *ITE* encoding. In fact, in the *BDD* encoding, all resource constraints are fully controlled by the SAT solver. We argue in the Section VI that this is a key point in the good performance obtained with *BDD*.

V. Optimization

Our system uses the Yices 2's [21] SMT solver API to check the satisfiability of the encodings. Yices 2 has shown to be a competitive SMT solver when considering the linear arithmetic theory [20]. The solving process of our system is presented in Algorithm 1. It basically consists on the following steps:

1) Compute the preprocessings.
2) Detect infeasibility and end the process, or find a feasible schedule of modes.
3) Use the obtained schedule of modes to compute an UB with parallel SGS heuristic, as described in Section III-C.
4) Find the optimum makespan.

Recall that we only encode the constraints over non-renewable resources for feasibility check (step 2). This is denoted as *encode_MRCPSP_SAT* in Algorithm 1. For the last step of computing the optimum makespan, we have implemented a search procedure, namely *optimize_feasible*, which calls the SMT solver successively constraining the value of the variable S_{n+1} to be smaller or equal to UB. It is described in Algorithm 2. This optimization algorithm is based on a linear search schema starting from a feasible UB. Every time a feasible schedule is found, UB is updated to take its makespan minus one, and the SMT solver is called again. This procedure is repeated until we find the biggest infeasible UB.

Algorithm 1 solve_MRCPSP

```
Output: Optimum makespan if feasible. Otherwise return infeasible.
  $INS \leftarrow read\_MRCPSP\_instance()$
  //$INS$ contains instance data $(V, A, M, p, E, R, B, b)$
  $PREP \leftarrow preprocessing()$
  //$PREP$ contains preprocessed data $(E^*, ES, LS, B', b')$
  $ENC \leftarrow encode\_MRCPSP\_SAT(INS, PREP)$
  $(SAT, MODEL) \leftarrow smt\_check(ENC)$ //Check feasibility, and give a model if any
  if $SAT$ then
    $LB \leftarrow l_{0,n+1}$ //Trivial lower bound
    $SM \leftarrow get\_schedule\_of\_modes(MODEL)$
    $UB \leftarrow parallelSGS(INS, SM)$ //Heuristic solution
    $OPT \leftarrow optimize\_feasible(LB, UB, INS, PREP)$
    return $OPT$
  else
    return $INFEASIBLE$
  end if
```

Algorithm 2 optimize_feasible

```
Require: The instance is feasible.
Input: $LB$, $UB$, instance data ($INS$), preprocessed data ($PREP$).
Output: Optimum makespan
  $ENC \leftarrow encode\_MRCPSP(LB, UB, INS, PREP)$
  $smt\_assert\_encoding(ENC)$
  $SAT \leftarrow smt\_check()$
  if $SAT$ then
    $MODEL \leftarrow smt\_get\_model()$
    $MAKESPAN \leftarrow smt\_get\_makespan(MODEL)$
    $UB \leftarrow MAKESPAN - 1$
  end if
  while $SAT$ and $UB \geq LB$ do
    $smt\_assert(S_{n+1} \leq UB)$ //Bound the makespan
    $smt\_compress\_TW(UB, PREP)$ //Optional
    $SAT \leftarrow smt\_check()$
    if $SAT$ then
      $MODEL \leftarrow smt\_get\_model()$
      $MAKESPAN \leftarrow smt\_get\_makespan(MODEL)$
      $UB \leftarrow MAKESPAN - 1$
    end if
  end while
  if $SAT$ then
    return $UB$
  else
    return $UB + 1$
  end if
```

Moreover we compress the time windows at each iterative step of the optimization process. On the one hand, we assert new single atom clauses of the form $(S_i \leq LS_i)$ with the updated latest start times for the current UB, therefore bounding the value of S_i. On the other one hand, we also assert $(\neg y_{i,t})$ for the time instants t that are excluded from the time window of A_i, thus avoiding the solver the work of propagating these values that become trivial. This compression corresponds to the instruction *smt_compress_TW*, and is not mandatory for consistency. However, we show in Section VI that the compression supposes a noticeable speedup in the solving process. Notice that, since we decrease UB linearly, we never find an infeasible UB until the optimum is detected, so that we do not have to retract any constraint and we can maintain the learning of the SMT solver.

VI. Experiments

We have run our experiments on a 8GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz. In all experiments we use Yices 2.4.2 as the core SMT solver, and the timeout is 3600 seconds. Our executions are made on the j30 [15] and MMLIB50 [16] sets of instances. Both sets contain instances with 3 execution modes per activity and 2 renewable and 2 non-renewable resources. There are 552 feasible instances and 88 infeasible instances in j30, each one with 30 activities, and 540 feasible instances of 50 activities in MMLIB50. We provide comparative results with the system presented in [3], which uses a failure directed search for a Constraint Programming optimizer (we refer to this system as *FDS*). To our best knowledge this system has reported the best results for MRCPSP.

In Table I we analyze the time required to determine the infeasibility of the infeasible instances of j30. We show the performance of simplifying the *ITE* and *BDD* encodings for feasibility checks as explained in Subsection III-C (i.e. only finding a schedule of modes which satisfies the non-renewable resource constraints). We also include in Table I the time required if we use the full *ITE* and *BDD* encodings, and finally the time required by *FDS*. It can be seen that our system is better than *FDS* in any case for proving infeasibility, having three orders of magnitude of difference when using the simpli-

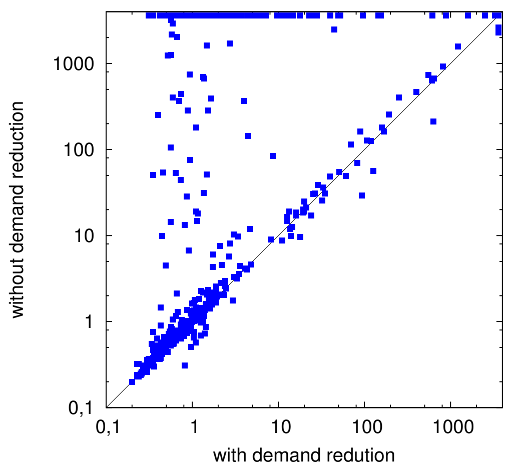


Fig. 2. Comparison of solving time (in seconds) with the *ITE* encoding using the non-renewable resource demand reduction (axis x) and without using it (axis y).

fied *BDD* encoding. It is specially noticeable the performance improvement achieved by simplifying the encoding, since the computation time is reduced two orders of magnitude both for *ITE* and *BDD*.

In Figure 2 we can see, on the feasible instances of j30 set and for *ITE* encoding, how using the non-renewable resource demand reduction presented in Subsection III-E supposes a very noticeable speedup. A total of 102 more instances were solved with *ITE* thanks to including this new preprocessing in it. We have also observed a very important speedup with *BDD*, although not as high as with *ITE*. Also, looking at Figure 2 we can see that there are some instances highly benefited from this preprocessing and some others that are not affected at all. We have observed that the most benefited instances are the ones whose activities have the highest demands on the resources.

TABLE I
SOLVING TIMES IN SECONDS AND NUMBER OF INSTANCES SOLVED OF THE INFEASIBLE INSTANCES OF J30 SET.

solver	**25%**	**median**	**75%**	**max**	**mean**	**solved**
ITE simp.	0.11	0.2	0.52	8.83	0.57	88
ITE full	7.55	14.6	28.44	416.68	27.49	88
BDD simp.	0.03	0.05	0.08	0.58	0.94	88
BDD full	1.39	1.97	2.75	5.63	2.23	88
FDS	27.15	53.07	107.12	462.7	91.48	88

We have evaluated on the feasible instances of j30 how invoking the procedure *smt_compress_TW* of Algorithm 2 helps to boost the solving process. Using the compression in *ITE* let us to solve 4 more instances than not using it, and the solving time is in average reduced a 55.22% (considering only the instances solved in both cases). Regarding the *BDD* encoding, we have observed that this compression does not suppose a clear improvement of the solving time as happens with *ITE* (neither a worsening), what suggests that *BDD* propagates better the implications of reducing the upper bound.

Figure 3 reflects the performance differences between *ITE* and *BDD*, based on the satisfiable instances of j30. The computation of the Binary Decision Diagrams requires some time, and it is penalizing the overall performance of the easiest to solve instances, what makes *ITE* clearly the best option for the easiest instances. Regarding the hardest instances, *BDD* supposes an important speedup, and it is able to solve 9 more instances than *ITE*. An indicator of the performance of the encodings is the number of conflicts encountered during the solving process. With *ITE*, the average number of conflicts encountered by the theory solver during the last optimization iteration was 31964, while with *BDD* was 216 (only considering the instances solved in both cases). It is not surprising that *BDD* has few theory conflicts, since the constraints over resources are fully encoded with Boolean formulas. But despite this fact, the number of Boolean conflicts is also significantly smaller with *BDD* (24813) compared to *ITE* (87227), which may suggest that with *BDD* the lemmas learned from the conflicts achieve a better prune of the search space. It can be seen in Table II how the advantage of *BDD* in front of *ITE* is still greater in MMLIB50 set.

Table II also contains results comparing our system and *FDS* in j30 and MMLIB50 sets. It can be seen that, although *FDS* goes faster in the easiest instances, we scale better using *BDD* and are able to solve more instances than them in both sets. In j30, we solve one instance more than *FDS* (only 8 remain unsolved), and the mean solving time is slightly smaller. But the advantage of *BDD* is specially noticeable in MMLIB50, which is the hardest set, having a third quartile around one order of magnitude smaller, a mean solving time a 22,6% smaller, and solving 30 instances more.

As a summary of the performance state of our system, it completely solves all the instance sets but j30 from PSPLib, both with *ITE* and *BDD*, never exceeding the timeout of 500 seconds. In j30, we close 2 instances that *FDS* is unable to solve with the defined experimental settings. Regarding MMLIB50, the best results referenced at its website were achieved by [25] using metaheuristics. We have been able with *BDD* to certify with an exact method the optimality of 445 instances, and we have improved the upper bound of a total of 51 instances with respect to [25]. Compared to *FDS*, we solve 33 instances that it is unable to solve.

TABLE II
SOLVING TIMES IN SECONDS AND NUMBER OF INSTANCES SOLVED OF THE FEASIBLE INSTANCES OF J30 SET AND OF MMLIB50 SET. THE UNSOLVED INSTANCES HAVE BEEN COUNTED AS 3600 SECONDS.

set	**solver**	**25%**	**median**	**75%**	**mean**	**solved**
j30 (feasible)	**ITE**	0.5	0.91	2.25	141.92	535
	BDD	1.42	2.8	5.08	89.48	544
	FDS	0.02	0.04	0.86	98.17	543
MMLIB50	**ITE**	1.37	7.56	timeout	1251.03	359
	BDD	3.93	9.87	155.79	692.17	445
	FDS	0.05	1.34	1028.94	894.20	415

VII. CONCLUSION AND FUTURE WORK

We have shown that SMT is a competitive approach for the MRCPSP. With some classical and a new preprocessing

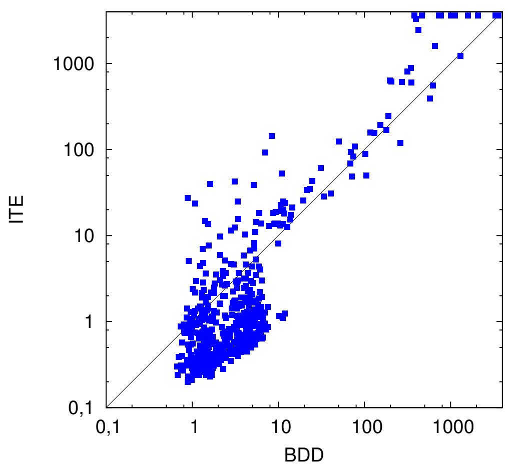


Fig. 3. Comparison of solving time (in seconds) with the *ITE* encoding and the *BDD* encoding for the feasible instances of j30 set.

method, using two SMT encodings, and using an off-the-shelf SMT solver as a decision procedure, we have achieved a robust and exact MRCPSP solver. Our results show that a combination of Boolean encoding for constraints over resources and LIA encoding for constraints over precedences is specially competitive in solving hard instances. It is worth noting that SMT provides not only efficiency, but also an expressive language for this kind of problems. Hence, frameworks based on SMT should rapidly gain acceptance in the community.

As future work we plan to further refine our encodings to improve performance and solve more instances. On the one hand, we consider the simplification of the BDDs of the Pseudo-Boolean constraints based on notions of incompatibility between activities and between execution modes. Also, with few modifications we could replace the theory of linear integer arithmetic by *integer difference logic*. It is a fragment of LIA that is usually handled with specific efficient theory solvers, and that is very suitable to model differences between integer variables (e.g. precedences between start times).

Acknowledgement

This work was supported by grant TIN2015-66293-R (MINECO/FEDER, UE) and grant MPCUdG2016/055 (UdG).

We thank Albert Oliveras and Enric Rodríguez-Carbonell for their valuable comments and suggestions, and also to Petr Vilím, co-author of [3], to provide us the code to compare the performances between our systems in a same machine.

References

[1] A. Schutt, T. Feydy, P. Stuckey, and M. Wallace, "Explaining the Cumulative Propagator," *Constraints*, vol. 16, no. 3, pp. 250–282, 2011.

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

[3] P. Vilím, P. Laborie, and P. Shaw, "Failure-directed search for constraint-based scheduling," in *Integration of AI and OR Techniques in Constraint Programming*. Springer, 2015, pp. 437–453.

[4] P. Brucker, A. Drexl, R. Mhring, K. Neumann, and E. Pesch, "Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods," *European Journal of Operational Research*, vol. 112, no. 1, pp. 3 – 41, 1999.

[5] W. Herroelen, E. Demeulemeester, and B. Reyck, "A classification scheme for project scheduling," in *Project Scheduling*, ser. International Series in Operations Research & Management Science, J. Wglarz, Ed. Springer US, 1999, vol. 14, pp. 1–26.

[6] J. Blazewicz, J. K. Lenstra, and A. R. Kan, "Scheduling Subject to Resource Constraints: Classification and Complexity," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11 – 24, 1983.

[7] A. Sprecher and A. Drexl, "Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm," *European Journal of Operational Research*, vol. 107, no. 2, pp. 431 – 450, 1998.

[8] G. Zhu, J. F. Bard, and G. Yu, "A Branch-and-Cut Procedure for the Multimode Resource-Constrained Project-Scheduling Problem," *INFORMS J. on Computing*, vol. 18, no. 3, pp. 377–390, Jan. 2006.

[9] J. C. Zapata, B. M. Hodge, and G. V. Reklaitis, "The multimode resource constrained multiproject scheduling problem: Alternative formulations," *AIChE Journal*, vol. 54, no. 8, pp. 2101–2119, 2008.

[10] T. S. Kyriakidis, G. M. Kopanos, and M. C. Georgiadis, "MILP formulations for single- and multi-mode resource-constrained project scheduling problems," *Computers & Chemical Engineering*, vol. 36, no. 0, pp. 369 – 385, 2012.

[11] R. K. Chakrabortty, R. A. Sarker, and D. L. Essam, "Event based approaches for solving multi-mode resource constraints project scheduling problem," in *Computer Information Systems and Industrial Management*. Springer, 2014, pp. 375–386.

[12] J. Coelho and M. Vanhoucke, "A new approach to minimize the makespan of various resource-constrained project scheduling problems," in *Proceedings of the 14th International Conference on Project Management and Scheduling*. TUM School of Management, 2014, pp. 1–4.

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

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

[15] R. Kolisch and A. Sprecher, "PSPLIB - A Project Scheduling Problem Library," *European Journal of Operational Research*, vol. 96, no. 1, pp. 205–216, 1997.

[16] V. Van Peteghem and M. Vanhoucke, "An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances," *European Journal of Operational Research*, vol. 235, no. 1, pp. 62–72, 2014.

[17] C. Artigues, S. Demassey, and E. Neron, *Resource-constrained project scheduling: models, algorithms, extensions and applications*. John Wiley & Sons, 2013.

[18] J. E. Kelley, "The critical-path method: Resources planning and scheduling," *Industrial scheduling*, vol. 13, pp. 347–365, 1963.

[19] R. Kolisch, "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation," *European Journal of Operational Research*, vol. 90, no. 2, pp. 320–333, 1996.

[20] B. Dutertre and L. de Moura, "Integrating Simplex with DPPL(T)," Computer Science Laboratory, SRI International, Tech. Rep. SRI-CSL-06-01, May 2006, http://yices.csl.sri.com/papers/sri-csl-06-01.pdf.

[21] B. Dutertre, "Yices 2.2," in *Computer-Aided Verification (CAV'2014)*, ser. Lecture Notes in Computer Science, vol. 8559. Springer, July 2014, pp. 737–744.

[22] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[23] M. Bofill, M. Palahí, J. Suy, and M. Villaret, "Solving intensional weighted csps by incremental optimization with bdds," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2014, pp. 207–223.

[24] I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and V. Mayer-Eichberger, "A new look at bdds for pseudo-boolean constraints," *Journal of Artificial Intelligence Research*, pp. 443–480, 2012.

[25] M. J. Geiger, "Iterated variable neighborhood search for the resource constrained multi-mode multi-project scheduling problem," *arXiv preprint arXiv:1310.0602*, 2013.