Chapter 1

The Resource-Constrained Multi-Mode Project Scheduling Problem

José Coelho and Mario Vanhoucke

Abstract This chapter reports on a new solution approach for the multi-mode resource-constrained project scheduling problem (MRCPSP, $MPS|prec|C_{max}$). This problem type aims at the selection of a single activity mode from a set of available modes in order to construct a precedence and a (renewable and non-renewable) resource-feasible project schedule with a minimal makespan. The problem type is known to be $\mathcal{NP}$-hard and has been solved using various exact as well as (meta-)heuristic procedures. The new algorithm splits the problem type into a mode assignment and a single mode project scheduling step. The mode assignment step is solved by a satisfiability (SAT) problem solver and returns a feasible mode selection to the project scheduling step. The project scheduling step is solved using an efficient meta-heuristic procedure from literature to solve the resource-constrained project scheduling problem (RCPSP). However, unlike many traditional meta-heuristic methods in literature to solve the MRCPSP, the new approach executes these two steps in one run, relying on a single priority list. Straightforward adaptations to the pure SAT solver by using pseudo boolean non-renewable resource constraints has led to a high quality solution approach in a reasonable computational time. Computational results show that the procedure can report similar or sometimes even better solutions than found by other procedures in literature, although it often requires a higher CPU time.

José Coelho
Department of Sciences and Technology, Universidade Aberta, Rua da Escola Politécnica, 147, 1269-001, Lisbon (Portugal) e-mail: Jose.Coelho@uab.pt
Mario Vanhoucke
Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2, 9000 Gent (Belgium) e-mail: mario.vanhoucke@ugent.be
Technology and Operations Management Area, Vlerick Business School, Reep 1, 9000 Gent (Belgium) e-mail: mario.vanhoucke@vlerick.com
Department of Management Science and Innovation, University College London, Gower Street, London WC1E 6BT (United Kingdom) e-mail: m.vanhoucke@ucl.ac.uk

Key words: Project scheduling · Resource constraints · Multi-mode · SAT · Makespan minimization

1.1 Introduction

This chapter presents a novel meta-heuristic approach to solve the non-preemptive multi-mode resource-constrained project scheduling problem (MRCPSP) within the presence of both limited renewable and non-renewable resource constraints, as proposed in Coelho and Vanhoucke (2011). The MRCPSP is an extension of the well-known RCPSP to the presence of multiple activity modes where each activity can be executed under a different duration and a corresponding renewable and nonrenewable resource use. For a recent survey on MRCPSP we refer to Weglarz et al. (2011), and to Chap. 25 of this handbook. The chapter is organized as follows. Sect. 1.2 introduces the notation and describes the problem formulation in detail. In Sect. 1.3 we present our approach to solve the scheduling problem type under study and give illustrative examples. Moreover, it is shown that the solution approach is very general and can be used for various other scheduling extensions. Sect. 1.4 enhances this solution approach to cope with excessive memory requirements. Sect. 1.5 reports comparative computational results and Sect. 1.6 contains the conclusions.

1.2 Model Formulation

The multi-mode project scheduling problem with multiple renewable and non-renewable resources ($MPS|prec|C_{max}$ in the three field classification) can be stated as follows. A set of activities V, numbered from a dummy start node 0 to a dummy end node n + 1, is to be scheduled without pre-emption on a set $\mathscr{R}$ of renewable resources and a set of $\mathscr{R}^n$ of non-renewable resources. Each renewable resource $k \in \mathscr{R}$ has a constant availability R_k per period while each non-renewable resource $k \in \mathscr{R}^n$ is restricted to R_k units over the complete planning horizon. Each non-dummy activity $i \in V$ can be executed in one of M_i modes $(p_{im}, r_{ikm}, r^n_{ikm})$ with $m \in \{1, 2, \ldots, M_i\}$. The selection of an activity mode involves a deterministic duration p_{im} for each activity i which requires r_{ikm} units of resource $k \in \mathscr{R}$ and r^n_{ikm} units of resource $k \in \mathscr{R}^n$. The start and end dummy activities representing the start and completion of the project have only one mode with a duration and renewable and non-renewable resource requirements equal to zero. A project network is represented by a topologically ordered activity-on-node format where E is the set of pairs of activities between which a finish-start precedence relationship with time lag 0 exists. We assume graph $G = (V, E)$ to be acyclic. A schedule S is defined by a vector of activity start times and is said to be feasible if all precedence and renewable and non-renewable resource constraints are satisfied. The objective of the problem type is to find a feasible schedule within the lowest possible project makespan, and hence, the problem type can be represented as $m, 1T|cpm, disc, mu|C_{max}$ using the classification scheme of Herroelen et al. (1999) or as $MPS|prec|C_{max}$ following the classification scheme of Brucker et al. (1999). The multi-mode resource-constrained project scheduling problem can be formulated as follows (see Talbot 1982):

$$\text{Min.} \sum_{t=ES_{n+1}}^{LS_{n+1}} t x_{n+1,1,t} \tag{1.1}$$

$$\text{s. t.} \sum_{m=1}^{M_i} \sum_{t=ES_i}^{LS_i} (t + p_{im}) x_{imt} \leq \sum_{m=1}^{M_j} \sum_{t=ES_j}^{LS_j} t x_{jmt} \quad ((i,j) \in E) \tag{1.2}$$

$$\sum_{m=1}^{M_i} \sum_{t=ES_i}^{LS_i} x_{imt} = 1 \quad (i \in V) \tag{1.3}$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{ikm} \sum_{s=max(t-p_{im},ES_i)}^{min(t-1,LS_i)} x_{ims} \leq R_k \quad (k \in \mathscr{R};\ t = 1,\ldots,UB) \tag{1.4}$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{ikm}^{n} \sum_{t=ES_i}^{LS_i} x_{imt} \leq R_k \quad (k \in \mathscr{R}^n) \tag{1.5}$$

$$(x_{imt} \in \{0,1\};\ i \in V;\ m = 1,\ldots,M_i;\ t = 1,\ldots,UB) \tag{1.6}$$

where x_{imt} is equal to 1 if activity i is performed in mode m and started at time instance t, and 0 otherwise. Eq. 1.1 minimizes the total project makespan. The constraints of Eq. 1.2 take the finish-start precedence relations with a time-lag of zero into account. Constraints of Eq. 1.3 secure that each non-preemptable activity is performed exactly once in exactly one mode. The renewable resource constraints are satisfied thanks to constraints of Eq. 1.4 where UB is an upper bound on the project makespan. The constraints of Eq. 1.5 restricts the use of the non-renewable resources over the complete time horizon. The constraints of Eq. 1.6 force the decision variables to be binary values. Note that the abbreviations ES_i and LS_i are used to denote the earliest and latest start for activity i given the project upper bound UB using traditional forward and backward critical path calculations.

Consider an example project taken from Kolisch and Drexl (1997) that will be used throughout the remainder of this chapter with 5 non-dummy activities, one renewable resource with an availability of R_1 = 4 and one non-renewable resource with an availability R_1^n = 8. Fig. 1.1 shows the activity-on-node network with the different activity modes below each node. The right part of the figure displays the optimal renewable resource profile, resulting in a total project makespan of 7 time units.

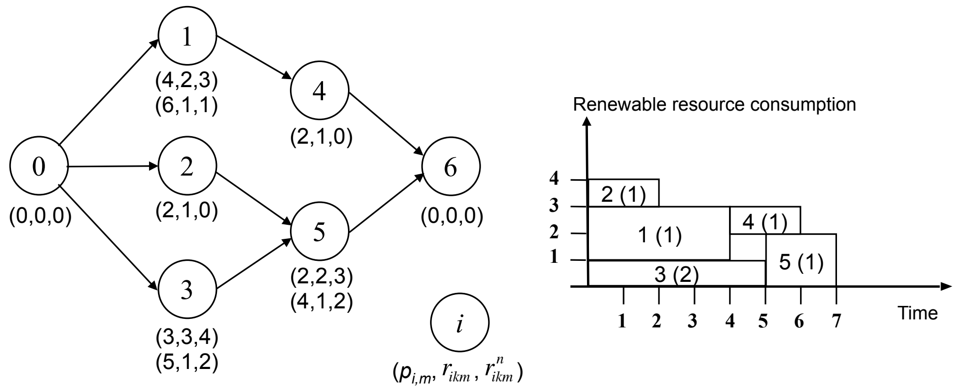

Fig. 1.1 A fictitious example project with optimal resource profiles (Source: Kolisch and Drexl 1997)

1.3 Solution Approach

The MRCPSP can be easily modeled as an RCPSP instance where each multi-mode activity i is split into M_i single-mode sub-activities among which exactly one sub-activity needs to be selected for execution. Consequently, the project network of Fig. 1.1 can be transformed into an RCPSP network with $\sum_{i=1}^{n} M_i$ non-dummy sub-activities as displayed in Fig 1.2, where the first number below the node denotes the sub-activity duration and the two other numbers below the node the renewable and non-renewable resource requirements. Consequently, the MRCPSP can be split into a mode assignment step taking the non-renewable resource constraints into account (i.e. constraints Eq. 1.3 and Eq. 1.5) and a single-mode resource-constrained project scheduling step taking the renewable resource and precedence constraints (constraints Eq. 1.2 and Eq. 1.4) into account. The mode assignment and the project scheduling steps will be discussed in Sects. 1.3.1 and 1.3.2, respectively. In literature, most meta-heuristic search procedures for the MRCPSP make use of an activity list and a mode list, and run the mode assignment step and the project scheduling step iteratively. In the solution approach of the current chapter, these steps will be performed in a single run, making use of only one priority list per run (which acts both as an activity list and a mode list). This new feature will be discussed in Sect. 1.3.2.

1.3.1 Mode Assignment

The mode assignment step boils down to the assignment of a single mode from the set of modes to each activity while not violating the limited non-renewable resource availability constraints, and can be easily modelled as a Boolean satisfiability prob-

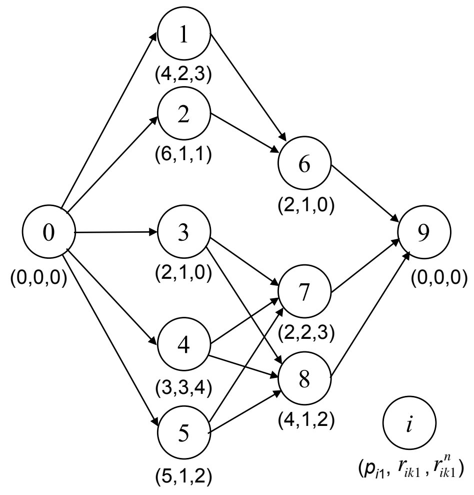

Fig. 1.2 The single-mode RCPSP network of Fig. 1.1 without any activity mode restrictions

lem instance. The boolean satisfiability problem (SAT) is a well-known decision problem where an expression of boolean variables (referred to as literals) linked by means of the logical operators *and*, *or* and *not* is questioned to be true or false. The problem has been studied extensively in literature (see e.g. the paper by Marques-Silva and Sakallah (1999), amongst others) and is known to be $\mathcal{NP}$-complete (see Cook 1971).

Obviously, the solution of the mode assignment step needs to be checked and possibly adjusted for the non-renewable resource infeasibility, and hence, this step can be easily modeled as a SAT instance. The enumeration of all feasible mode combinations is practically impossible due to the huge number of possible combinations. Moreover, Kolisch and Drexl (1997) have shown that finding a feasible mode combination for the MRCPSP is $\mathcal{NP}$-complete when two or more non-renewable resources are taken into account. Therefore, the choice of selecting a SAT algorithm above an alternative enumeration algorithm is based on the following logic:

- A SAT algorithm allows a simple mode feasibility check and a scheduling step using a single activity list instead of two separate lists as normally done in literature.
- In a SAT algorithm it is easy to implement learning (Sect. 1.3.3.2) which can be used over different mode combination searches.

Consequently, the mode assignment step can be represented in the *conjunctive normal form (CNF)* which is a conjunction of clauses linked by the "and" operator. A SAT instance contains several clauses to deal with the various mode assignments and/or the non-renewable resource constraints. The clauses for the network of Figs.

1.1 and 1.2 can be represented as follows:

Single-mode activities:	$x_0 + x_3 + x_6 + x_9 = 4$
Mode assignment for activity 1:	$x_1 + x_2 = 1$
Mode assignment for activity 3:	$x_4 + x_5 = 1$
Mode assignment for activity 5:	$x_7 + x_8 = 1$
Non-renewable resource constraint:	$3x_1 + x_2 + 4x_4 + 2x_5 + 3x_7 + 2x_8 \leq 8$

with x_i a 0/1 variable of sub-activity i to denote whether the mode has been assigned (1, true) or not (0, false). All mode assignment constraints can be easily translated into the CNF where the 0/1 x_i variables are now boolean variables (literals). The non-renewable resource constraint is known as a pseudo boolean constraint. These type of constraints can be solved by a pseudo boolean solver (Chai and Kuehlmann, 2005, Markov et al., 2002) or can be translated into the CNF (Bailleux et al., 2006). However, the non-renewable resource constraint also contains activity information (e.g. x_1 and x_2 are variables from the same activity and hence, only one variable can be set to true), but none of the previous methods takes this additional information into account. The enumeration scheme of Sect. 1.3.1.1 translates the pseudo boolean non-renewable resource constraints into CNF using this extra activity information. The mode assignment constraints can be translated into CNF as follows[1]:

Single-mode activities:	$x_0 \wedge x_3 \wedge x_6 \wedge x_9$
Mode assignment for activity 1:	$(x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$
Mode assignment for activity 3:	$(x_4 \vee x_5) \wedge (\overline{x_4} \vee \overline{x_5})$
Mode assignment for activity 5:	$(x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8})$
Non-renewable resource constraint:	Enumeration scheme discussed in 1.3.1.1

In the remaining sections, the translation of pseudo boolean non-renewable resource constraint clauses to a CNF is explained in detail. In Sect. 1.4, an adapted pseudo boolean solver is presented that incorporates these non-renewable resource constraint clauses and both approaches are compared.

1.3.1.1 Non-Renewable Resource Constraint Clauses

The construction of the constraint clauses for the non-renewable resource constraints is based on a simple yet efficient enumeration scheme. The enumeration scheme starts for the initial non-renewable resource constraint (e.g. $3x_1 + x_2 + 4x_4 + 2x_5 + 3x_7 + 2x_8 \leq 8$ for the example project) and gradually reduces the size of this constraint by iteratively setting boolean variables to true. More precisely, the scheme enumerates all activity mode variables for a single activity at each level of the enumeration tree and creates a node at that level for each variable that is set to true. In

[1] In general, a constraint $y_1 + y_2 + \ldots + y_n = 1$ can be represented in the CNF as $(y_1 \vee y_2 \vee \ldots \vee y_n) \wedge (\overline{y_1} \vee \overline{y_2}) \wedge \ldots \wedge (\overline{y_1} \vee \overline{y_n}) \wedge (\overline{y_2} \vee \overline{y_3}) \wedge \ldots \wedge (\overline{y_{n-1}} \vee \overline{y_n})$.

doing so, the size of the non-renewable resource constraint is gradually reduced and clauses are added when necessary.

The minimum and maximum non-renewable resource demand for an activity i are defined as:

$$r_{ik}^{min} = \min_{m=1,\ldots,M_i} r_{ikm}^{n} \text{ and } r_{ik}^{max} = \max_{m=1,\ldots,M_i} r_{ikm}^{n} \tag{1.7}$$

The total minimum and maximum remaining non-renewable resource demand is the sum of the individual minimal and maximal resource requests, as:

$$r_{k}^{sum-min} = \sum_{i \in U} r_{ik}^{min} \text{ and } r_{k}^{sum-max} = \sum_{i \in U} r_{ik}^{max} \tag{1.8}$$

where $U \subset V$ is the set of activities that have not been evaluated at previous levels of the search tree. Likewise, $R_k^{`}$ is used to denote the remaining non-renewable resource availability after reduction of the resource use of the boolean variables (i.e. mode selections) that have been set to true.

The remaining non-renewable resource constraint at each node will be evaluated to detect whether backtracking with or without adding constraint clauses is possible. The two evaluation rules applied at each node of the tree can be defined as follows:

1. If the remaining non-renewable resource constraint is satisfied, continue with the other node(s) at the current level of the tree without the insertion of a clause. Formally, if $r_k^{sum-max} \leq R_k^{`}$ then the constraint is satisfied.
2. If the remaining non-renewable resource constraint is violated, continue with the other node(s) at the current level of the tree and insert a clause that consists of the negation of all activity modes selected up to the current branch of the tree. Formally, if $r_k^{sum-min} > R_k^{`}$ then the remaining non-renewable constraint is impossible to satisfy.

The enumeration scheme of the example project can be graphically presented in Fig. 1.3.

The enumeration search of the example has found only one conflict clause as $(\overline{x_1} \vee \overline{x_4})$ at node 3. Indeed, at node 3 of the tree, $r_k^{sum-min} = 2 > 1$ and hence, this constraint cannot be satisfied. The conflict $(\overline{x_1} \vee \overline{x_4})$ is added and the algorithm continues with node 4 of the tree. Note that, as an example, $r_k^{sum-max} = 7 \leq 7$ at node 5 of the enumeration tree, and hence, the remaining constraint is satisfied and the algorithm backtracks to the previous level.

In order to improve the efficiency of the enumeration scheme, a number of additional node reduction rules are applied during the search which can be summarized as follows:

1. If an activity has an equal non-renewable resource demand for all its modes then this activity can be deleted from the search and the total non-renewable resource availability has to be reduced by this resource demand.
2. If the difference between the non-renewable resource demand of activity i at mode m and its corresponding minimal non-renewable resource demand is larger

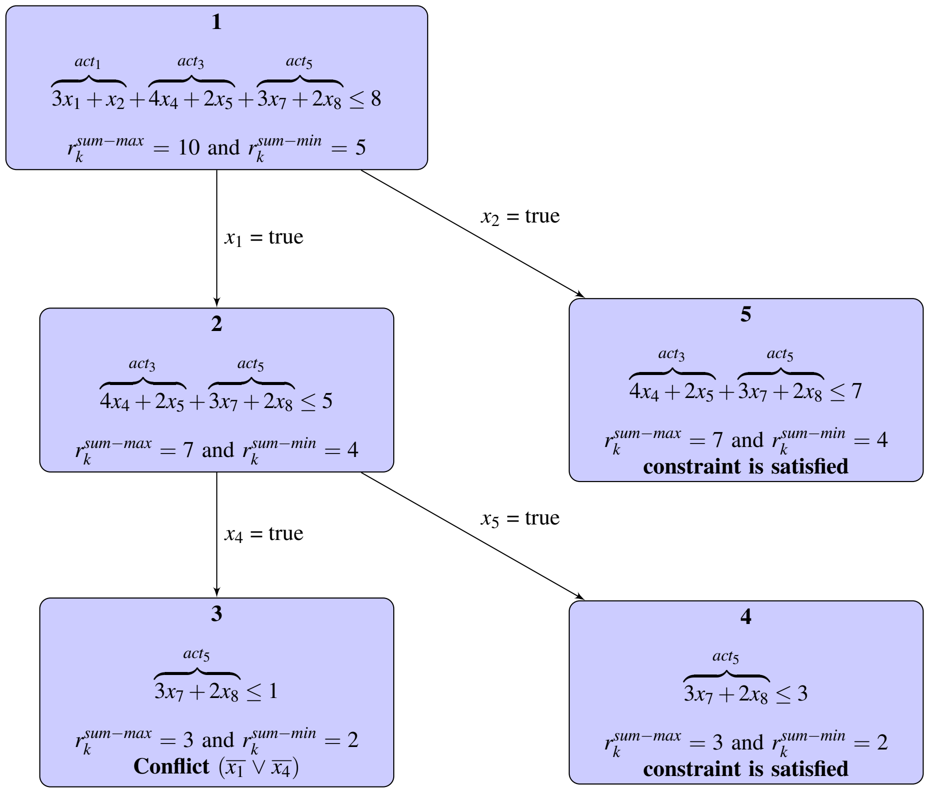
1
$\underbrace{3x_1+x_2}_{act_1}+\underbrace{4x_4+2x_5}_{act_3}+\underbrace{3x_7+2x_8}_{act_5}\leq 8$
$r_k^{sum-max}=10$ and $r_k^{sum-min}=5$
x_1 = true
x_2 = true
2
$\underbrace{4x_4+2x_5}_{act_3}+\underbrace{3x_7+2x_8}_{act_5}\leq 5$
$r_k^{sum-max}=7$ and $r_k^{sum-min}=4$
5
$\underbrace{4x_4+2x_5}_{act_3}+\underbrace{3x_7+2x_8}_{act_5}\leq 7$
$r_k^{sum-max}=7$ and $r_k^{sum-min}=4$
constraint is satisfied
x_4 = true
x_5 = true
3
$\underbrace{3x_7+2x_8}_{act_5}\leq 1$
$r_k^{sum-max}=3$ and $r_k^{sum-min}=2$
Conflict $(\overline{x_1}\vee\overline{x_4})$
4
$\underbrace{3x_7+2x_8}_{act_5}\leq 3$
$r_k^{sum-max}=3$ and $r_k^{sum-min}=2$
constraint is satisfied

Fig. 1.3 Enumeration scheme of the example project

than the difference between the resource's availability and its total mimimum remaining resource demand, then the activity mode can be set to false and removed from the formula. Formally, for each mode m of activity i, if $r^n_{ikm}-r^{min}_{ik} > R^{`}_k - r_k^{sum-min}$ then remove the activity mode from the search.

No node improvement rules can be applied due to the small size of the example network. As an example, since $r^n_{ikm}-r^{min}_{ik}$ is equal to 2, 0, 0, 2, 0, 0, 1 and 0 for activities 1 to 5 of Fig. 1.1 and $R_k - r_k^{sum-min}$ = 8 - 5 = 3, no activity mode can be removed from the search (improvement rule 2).

1.3.1.2 The Activity List SAT Mode Assignment Procedure

The procedure presented in this chapter makes use of a list with a dual role. First, the list serves as a *variable list* to solve the CNF and guarantees the selection of a single mode for each activity satisfying the non-renewable constraints, if possible (see Sect. 1.3.1). Second, the list serves as a traditional *activity list* for the construction of a project schedule based on the selected modes (see Sect. 1.3.2). In the remainder of this chapter, we refer to this list as an activity list AL.

Despite the $\mathcal{NP}$-hardness of the SAT, research has proposed many advanced algorithms able to solve problem instances with up to thousands of literals and millions of constraints, which is far beyond the size of the SAT instances solved in our case (see e.g. the SAT competition results of Kullmann 2006). Therefore, we rely on the efficient DPLL algorithm of Davis et al. (1962). The DPLL algorithm is a complete, backtracking-based algorithm for deciding the satisfiability of propositional logic formulae in the CNF. The algorithm sequentially selects literals (i.e. variables) from a pre-defined variable list and assigns a truth value to it in order to simplify the CNF formula. If this assignment leads to an unsatisfiable simplified formula (referred to as a conflict), the opposite value (false) is set to the selected literal and the algorithm continues. The unit propagation rule checks whether a clause is a unit clause, i.e. it contains only a single unassigned literal. When this is the case, this clause can only be satisfied by assigning the necessary value to make this literal true.

Assume a simple activity list AL = $\{0,1,2,3,4,5,6,7,8,9\}$ and a set L denoting the set of assigned literals at a given moment (where x and $\bar{x}$ can not belong to L at the same moment). The DPLL algorithm for the example CNF = $x_0 \wedge x_3 \wedge x_6 \wedge x_9 \wedge (x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_4 \vee x_5) \wedge (\overline{x_4} \vee \overline{x_5}) \wedge (x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8}) \wedge (\overline{x_1} \vee \overline{x_4})$ runs as follows:

1) Activity list AL = $\{0,1,2,3,4,5,6,7,8,9\}$
   Unit clause rule x_0, x_3, x_6, x_9: $L = \{x_0, x_3, x_6, x_9\}$
   CNF:
   $(x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_4 \vee x_5) \wedge (\overline{x_4} \vee \overline{x_5}) \wedge (x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8}) \wedge (\overline{x_1} \vee \overline{x_4})$

2) Selection of variable from AL, not in L: x_1: $L = \{x_0, x_3, x_6, x_9, x_1\}$
   CNF: $\overline{x_2} \wedge (x_4 \vee x_5) \wedge (\overline{x_4} \vee \overline{x_5}) \wedge (x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8}) \wedge \overline{x_4}$
   Unit clause rule $\overline{x_2}$, $\overline{x_4}$: $L = \{x_0, x_3, x_6, x_9, x_1, \overline{x_2}, \overline{x_4}\}$
   CNF: $x_5 \wedge (x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8})$
   Unit clause rule: x_5: $L = \{x_0, x_3, x_6, x_9, x_1, \overline{x_2}, \overline{x_4}, x_5\}$
   CNF: $(x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8})$

3) Selection of variable from AL, not in L: x_7: $L = \{x_0, x_3, x_6, x_9, x_1, \overline{x_2}, \overline{x_4}, x_5, x_7\}$
   CNF: $\overline{x_8}$
   Unit clause rule: $\overline{x_8}$: $L = \{x_0, x_3, x_6, x_9, x_1, \overline{x_2}, \overline{x_4}, x_5, x_7, \overline{x_8}\}$

Note that no conflict has been generated during the DPLL algorithm since each assignment has led to a satisfiable simplified CNF formula. The selection of literals can be translated into sub-activity durations as shown in the last column of Table 1.1. The sub-activity durations (equal to zero when the corresponding activity mode has not been selected) of the table are input for the scheduling step of Sect. 1.3.2. An illustrative example to show the presence of conflict generation and algorithmic backtracking will be given in Sect. 1.3.3.2.

Table 1.1 The SAT mode assignment solution

		Fig. 1.1	Fig. 1.2		
i	m	$(p_{im}, r_{i1m}, r^n_{i1m})$	i	L	p_{i1}
0	1	(0,0,0)	0	x_0	0
1	1	(4,2,3)	1	x_1	4
	2	(6,1,1)	2	$\overline{x_2}$	0
2	1	(2,1,0)	3	x_3	2
3	1	(3,3,4)	4	$\overline{x_4}$	0
	2	(5,1,2)	5	x_5	5
4	1	(2,1,0)	6	x_6	2
5	1	(2,2,3)	7	x_7	2
	2	(4,1,2)	8	$\overline{x_8}$	0
6	1	(0,0,0)	9	x_9	0

1.3.2 RCPSP Scheduling Step

The project scheduling step is a resource-constrained project scheduling problem where each activity has a single execution mode determined by the mode assignment step. The solution of SAT provides positive (in case the literal has been set to true) or zero (literal is set to false) duration sub-activities and hence determines the characteristics of the RCPSP instance. In this chapter, the scheduling step is performed based on the decomposition based genetic algorithm of Debels and Vanhoucke (2007). These authors present a genetic algorithm that makes use of an activity list to construct resource feasible schedules based on a forward and backward serial generation scheme and they have shown that their procedure outperforms the current state-of-the-art procedures.

While the details of this genetic algorithm will not be repeated here, a basic overview of the different elements is given along the following lines:

- Dual population: The population based heuristic splits the total number of generated schedules into two separate populations containing *left- and right-justified schedules*, inspired by the promising results found by Valls et al. (2005).
- Representation of a schedule: Based on the remarks by Debels et al. (2006) who have illustrated that a random key representation is very effective thanks to the use of the *topological ordering notation* (Valls et al., 2003), this notation has been adapted to the dual population heuristic, as follows: the random key elements are equal to the activity finishing times for a left-justified schedule, and equal to the starting times for a right-justified schedule.
- Parent selection: Parents are selected using a *2-tournament selection* where two population elements from the population are chosen randomly, and the element with the best objective function value is selected. Afterwards, one element is randomly labelled as the father and the other element as the mother.
- Crossover operator: The combination of the genes of both parents is done by a two-point crossover operator based on a modified version of the peak crossover operator of Valls et al. (2008) that makes use of the *resource utilization ratio*.

This ratio measures the resource utilization at time unit t allowing the selection of time intervals for which the resource utilization is high, so-called peaks, and time intervals with low resource utilization.

- Local search: The local search is based on an iterative *forward and backward search* (Li and Willis, 1992) to improve the two separate populations containing left- and right-justified schedules.
- Decomposition approach: The genetic algorithm has been extended to a so-called *decomposition-based heuristic* which iteratively solves subparts of the project leading to the best results in literature.

While most research papers rely on two separate lists to solve the MRCPSP (one to determine the assignment of modes and a second to feed a schedule generation scheme), the algorithm presented here relies on a single list that takes both the mode assignments and the activity scheduling step into account.

Theorem 1.1. *Using a single activity list for the SAT mode assignment and the RCPSP scheduling steps does not exclude optimal solutions.*

Proof. It has been shown in Sect. 1.3 that a project network with multi-mode activities can be represented by a set of sub-activities while a corresponding feasible schedule can be represented by two subsets of this set of all sub-activities: One subset contains sub-activities with a positive duration and a corresponding starting time and the remaining subset contains sub-activities with a zero duration. It will be shown that any active project schedule has at least one sub-activity list AL that, when used as input lists by both the mode assignment step (SAT) and the scheduling step (RCPSP), leads to this schedule. If the existence of such an AL can be shown for any feasible active schedule, then this existence also holds for the the optimal project schedule. Consider to that purpose a feasible solution, i.e. an active project schedule where each activity has a starting time (= scheduling step) and a positive duration defined by the selected mode (= mode assignment step). These activities belong to the subset of sub-activities with a positive duration while the remaining subset with zero duration sub-activities is obviously not visible in the project schedule. The existence of an AL that leads to the feasible project schedule using the scheduling and mode assignment steps can be shown through the following three steps.

1. Each schedule can be easily represented by an activity list using the unique standardized random key (RK) representation presented by Debels et al. (2006). This representation consists of the starting times of the sub-activities with a positive duration in increasing order followed by set of remaining activity modes that are not part of the feasible schedule (i.e. the sub-activities with a duration of zero) and is an extension of the topological order representation of Valls et al. (2003, 2004). Since a feasible project schedule can be uniquely defined by its activity starting times, such an AL can always be constructed.
2. It has been shown by Debels et al. (2006) that such a standardized RK or AL has a unique correspondence with a project schedule where each positive dura-

tion activity has a starting time equal to its AL value, and this schedule can be generated by the use of the well-known serial schedule-generation scheme.
3. This unique RK or AL will also result in the mode assignments of the feasible project schedule, i.e. those sub-activities with a positive duration in the project schedule will be set to true in the SAT step, while the zero duration sub-activities will be set to false. Since the AL defines the ranking of sub-activities that will be selected by the SAT step, the SAT mode assignment step applied to this AL will select the first subset of sub-activities and set their values to true (i.e. with a positive duration). This will never generate a conflict since the activity list corresponds to a feasible project schedule. The remaining sub-activities will be set to false due to the clause that only one mode can be selected per activity, leading to the mode assignment represented in the feasible project schedule. □

The use of a single priority list for both the mode assignment step and the activity scheduling step is unique and in contrast with most meta-heuristic procedures to solve the MRCPSP in literature. The new solution approach presented in the current chapter transforms the MRCPSP instance to an RCPSP instance, where each multi-mode activity i is split into M_i single-mode sub-activities, and SAT restrictions are added to assure that only one sub-activity will be selected. Moreover, before applying the serial schedule-generation scheme to the RCPSP, the mode assignment is called using the same priority list as used for the RCPSP, and sub-activities that are not selected are set to zero. The example of Fig. 1.2 displays the general RCPSP instance used throughout the search to a high quality solution, without any SAT restriction forcing that only one mode can be selected per activity. A single priority list will transform this figure into a resource feasible schedule (i.e. the scheduling step) where exactly one mode per activity is selected (i.e. the mode assignment step).

1.3.3 Advantages of SAT Solvers

1.3.3.1 Pre-Processing

The selection of the branching literals is an important factor for the efficiency (Hooker and Vinay, 1995). Obviously, infeasible instances and instances with tight non-renewable resource constraints might consume a lot of CPU time using the activity list discussed previously, since this list does not contain guiding information to select variables to branch. Therefore, at the initial start of the MRCPSP search, a random AL could lead to a high CPU consumption. Consequently, a pre-processing run using a selection rule (we use the greedy heuristic rule of Marques-Silva and Sakallah 1999) improves the decision assignment at each stage of the search process and leads to two advantages:

- Feasibility check: When the non-renewable resource constraints are impossible to satisfy, the algorithm stops and there is no need to start the AL search.

- Clause learning: The information gathered during the initial pre-processing run can be saved to improve the remaining AL runs. This is explained in the next section.

1.3.3.2 Learning

Clause learning is an important technique in SAT, since Marques-Silva and Sakallah (1999) have shown that recording conflict-inducing clauses can help to prevent the occurrence of similar conflicts later on in the search. Rather than reducing the CPU time of a single search of a SAT instance, in our case there is a need to reduce the total CPU time used to repeatedly solve the SAT instance for all generated activity lists.

Therefore, we made a simplification and only introduced learning clauses with a maximum number of literals. More precisely, when a conflict arises due to an assignment at level three (or above) then a clause is added with the negation of the decision assignments used up to that level to prevent this conflict to occur again in the next searches of this instance using other activity lists.

The maximum level of three guarantees that only clauses with three or less literals are added to the SAT instance, keeping the instance size relatively stable. Computational tests have been done with a maximum of 4 to 10, and have shown that it leads to a higher CPU consumption.

In order to illustrate the effect of clause learning, a computational experiment has been set up that compares the SAT learning effect with an enumeration scheme that evaluates mode assignments in a similar way as the SAT procedure. More precisely, the enumeration scheme has exactly the same performance than the SAT algorithm under a single run, leading to exactly the same mode assignments, but it operates on modes rather than on literals and does not include clause learning. The test runs are done on test data truncated after a predefined number of schedules and compare the average number of backtracks used in both the enumeration and SAT approach since they can be considered as a proxy for the total computational time of the mode assignment step. Since the average number of backtracks varies heavily from instance to instance, a graph has been constructed for an example project instance shown in Fig. 1.4 that illustrates the main results of the experiments. In our computational tests, we have seen that approximately 25% of the J30 problem instances clearly benefit from clause learning. As an example, the J308_6 instance benefits most from the SAT learning, and the computational tests have shown that the number of backtracks is equal to 131,362,000 and 5,500,000 for the enumeration approach and the SAT approach, respectively, under a stop criterion of 100 generated schedules. This number increases to 481,240,000 for the enumeration approach when the number of generated schedules is set to 1,000, while it stays relatively constant for the SAT approach, which illustrates that the incorporation of learning can lead to huge time reductions for some instances.

The graph shows that the initial performance of both mode assignment procedures is similar under a low stop criterion, but that the SAT approach benefits from

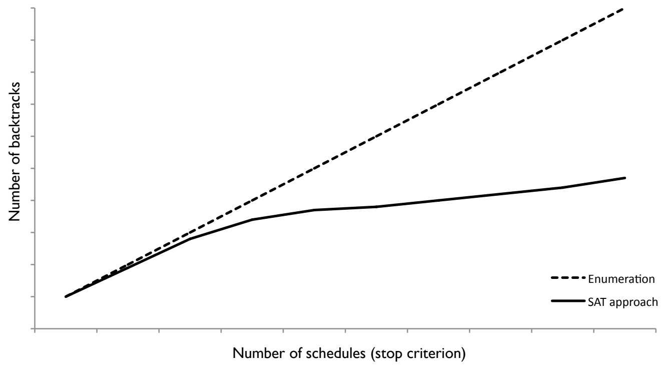

Fig. 1.4 An example graphical representation of the number of backtracks for two mode assignment procedures

learning while the enumeration approach is not able to do so. Indeed, while the number of backtracks is more or less linear with the number of predefined schedules set as a stop criterion for the enumeration approach, the SAT procedure is able to reach a relatively horizontal increase after a certain number of generated schedules, thanks to the incorporation of clause learning.

A small illustrative example is given along the following lines to show the presence of conflict generation, algorithmic backtracking and the use of learning clauses. Assume a CNF = $(\overline{a} \vee \overline{c} \vee d) \wedge (\overline{b} \vee \overline{c} \vee \overline{d}) \wedge (\overline{a} \vee c \vee \overline{d}) \wedge (\overline{b} \vee c \vee d)$ and a simple activity list AL = $\{a, b, c, d\}$. The DPLL algorithm for the example CNF runs as follows:

1) Level 1. Selection of variable a from the AL, not in L: $L = \{a\}$
CNF = $(\overline{c} \vee d) \wedge (\overline{b} \vee \overline{c} \vee \overline{d}) \wedge (c \vee \overline{d}) \wedge (\overline{b} \vee c \vee d)$

2) Level 2. Selection of variable b from the AL, not in L: $L = \{a, b\}$
CNF = $(\overline{c} \vee d) \wedge (\overline{c} \vee \overline{d}) \wedge (c \vee \overline{d}) \wedge (c \vee d)$
This instance can never be true, but the algorithm does not detect this since it does not generate a conflict

3) Level 3. Selection of variable c from the AL, not in L: $L = \{a, b, c\}$
CNF = $d \wedge \overline{d}$
Unit clause rule $d, \overline{d}$: $L = \{a, b, c, d, \overline{d}\}$
A conflict is generated since d and $\overline{d}$ cannot belong to L
Selection of other literal of variable c from AL, not in L: $L = \{a, b, \overline{c}\}$
CNF = $d \wedge \overline{d}$
Unit clause rule $d, \overline{d}$: $L = \{a, b, \overline{c}, d, \overline{d}\}$
A conflict is generated since d and $\overline{d}$ cannot belong to L
There are no other values for the variable c so the algorithm backtracks to level 2

4) Level 2. A learning clause $(\overline{a} \vee \overline{b})$

If this clause would exist at the start of the procedure, the unit clause rule of step 1 would assign $\overline{b}$ and the conflict would not be generated

Selection of other literal of variable b from AL, not in L: $L = \{a, \overline{b}\}$

CNF = $(\overline{c} \vee d) \wedge (c \vee \overline{d})$

5) Level 3. Selection of variable c from the AL, not in L: $L = \{a, \overline{b}, c\}$

CNF = d

Unit clause rule d: $L = \{a, \overline{b}, c, d\}$

This is the first valid assignment found with this activity list

The learning clause $(\overline{a} \vee \overline{b})$ will be inserted such that during the next search this conflict will not be generated, leading to a time saving

1.4 Adapted Pseudo Boolean Solver Approach

In the model presented earlier, each non-renewable constraint is the subject of the enumeration scheme of Sect. 1.3.1.1, which leads to a set of clauses that need to be stored as an input file for the SAT solver (i.e. the DPLL procedure) that is called for each activity list generated during the search. When the size of the project network instance becomes relatively large, both in terms of the number of project activities and the number of non-renewable resource constraints, the number of clauses translated from the pseudo boolean non-renewable resource constraints can grow exponentially, leading to a large SAT input file and excessive use of memory. Table 1.2 illustrates the exponential growth of the number of clauses and the corresponding memory need for the PSPLIB (Kolisch and Sprecher, 1996) instances. The column "SAT(3)" shows that the total disk space required to store the PSPLIB instance with the SAT approach of Sect. 1.3 grows very quickly up to almost 50 GB, using a stop criterion of 1 million clauses. From the J14 set on, several instances exceed this limit and are truncated in an early stage, as shown by column "#Ins-M".

Table 1.2 A comparison between the pure SAT and the SAT(k) approach

Set	#Var	#Ins	M (SAT(3))	M (SAT(4))	Avg.Cl(3)	Avg.Cl(4)	#Ins-M
J10	32	536	119	1.5	5,883	21.4	0
J12	38	547	801	2.1	32,435	25.4	0
J14	44	551	5,342	2.4	186,436	29.4	24
J16	50	550	14,445	2.7	454,055	33.4	162
J18	56	552	23,112	3.0	653,028	37.4	320
J20	62	554	26,759	3.4	694,061	41.4	378
J30	92	640	49,554	5.8	750,030	61.5	480

The abbreviations in the first row of the table can be explained along the following lines:

- #Var : Average number of variables in the SAT instance (equal to the number of activity modes).
- #Ins : Number of instances in each set.
- M (SAT(3)) : Total disk space of SAT instance using the SAT approach of Sect. 1.3 (in MB).
- M (SAT(4)) : Total disk space of SAT instance using the adapted SAT approach of this section (in MB).
- Avg.Cl(3) : Average number of clauses generated using the SAT approach of Sect. 1.3.
- Avg.Cl(4) : Average number of clauses generated using the adapted SAT approach of this section.
- #Ins-M : Number of instances leading to memory problems (i.e. $\geq$ 1 million added clauses) using the SAT approach of Sect. 1.3. Note that all instances can be solved by the adapted SAT approach as briefly discussed hereafter (i.e. #Ins-M = 0).

In order to avoid the heavy computational burden and the excessive memory requirement of the SAT(3) solution approach, the pseudo boolean non-renewable resource constraints are not translated to CNF using the enumeration scheme of Sect. 1.3.1.1, but are used directly in the DPLL algorithm. Consequently, the two evaluation rules and the two node reduction rules of Sect. 1.3.1.1 are still applicable, but are dynamically used during the DPLL search. In doing so, this approach avoids the excessive memory increase of the SAT input file due to the enumeration of the boolean non-renewable resource constraints in advance. The introduction of this approach leads to a dramatic reduction in the disk space (and hence the memory use) and the number of constraint clauses, as shown in columns “SAT(4)” and “Avg.Cl(4)” of Table 1.2. The reduction of memory has a beneficial effect on the initial memory allocation computation time, but does not speed up the rest of the search of the SAT solver in any way. Indeed, the results and the number of steps in the DPLL algorithm are the same for the SAT(3) and the SAT(4) approaches.

1.5 Computational Results

This section reports on computational results to evaluate the performance of the algorithm. The algorithm has been coded in C++ and tests have been run on a Dell Dimension DM051 with a Pentium D with a 2.80 GHz processor. The first benchmark test set is the well-known PSPLIB dataset which contains multi-mode project network instances generated by ProGen (Kolisch et al., 1995) with 10, 12, 14, 16, 18, 20 and 30 activities and with 2 renewable and 2 nonrenewable resources. The set is available from the ftp server of the University of Kiel (http://129.187.106.231/psplib/). Results are also compared with a second benchmark dataset taken from Boctor (1993), containing 240 instances with 50 and 100 activities and only renewable resource constraints.

The first computational results have been displayed in Table 1.3 for the new procedure (denoted by “This Work”) under a stop criterion of 5,000 generated schedules. The values are average deviations from the optimal solution. The table shows that the new procedure is able to provide comparable results for some of the state-of-the-art procedures, but cannot outperform the best known results. The new procedure has also been truncated after 50,000 and 500,000 schedules in order to show the potential of the procedure to produce high-quality solutions. Although a comparison with the state-of-the-art results is not fair anymore, the table shows that near-optimal solution can be produced with the new procedure under high stop criterion values.

Table 1.3 Computational results for the PSPLIB dataset under a 5,000 schedule limit stop criterion

	J10	J12	J14	J16	J18	J20
Van Peteghem and Vanhoucke (2010)	0.01%	0.09%	0.22%	0.32%	0.42%	0.57%
Wang and Fang (2012)	0.12%	0.14%	0.43%	0.59%	0.90%	1.28%
Wang and Fang (2011)	0.10%	0.21%	0.46%	0.57%	0.94%	1.39%
Elloumi and Fortemps (2010) - v1	0.21%	0.29%	0.77%	0.91%	1.30%	1.62%
Elloumi and Fortemps (2010) - v2	0.14%	0.24%	0.80%	1.14%	1.53%	2.09%
Lova et al. (2009)	0.06%	0.17%	0.32%	0.44%	0.63%	0.87%
Jarboui et al. (2008)	0.03%	0.09%	0.36%	0.44%	0.89%	1.10%
Ranjbar et al. (2009)	0.18%	0.65%	0.89%	0.95%	1.21%	1.64%
Alcaraz et al. (2003)	0.24%	0.73%	1.00%	1.12%	1.43%	1.91%
Józefowska et al. (2001)	1.16%	1.73%	2.60%	4.07%	5.52%	6.74%
This Work (5,000)	0.07%	0.16%	0.32%	0.48%	0.56%	0.80%
	0.4s	0.5s	0.7s	0.9s	1.0s	1.2s
This Work (50,000)	0.00%	0.01%	0.05%	0.06%	0.08%	0.12%
	4.3s	5.4s	6.9s	8.4s	10.1s	11.8s
This Work (500,000)	0.00%	0.00%	0.01%	0.01%	0.01%	0.02%
	43.8s	53.9s	68.4s	82.9s	100.1s	117.0s

Table 1.4 reports results for the J30 instances as the average deviation from the minimal critical path length under four stop criterion values and compares the results with the procedure of Van Peteghem and Vanhoucke (2010) which is described as the best performing procedure up to today. The results show that the new SAT based procedure is not able to outperform the best performing procedure when the stop criterion is set relatively low. However, when both procedures are truncated after a longer time period, the procedure of this chapter reports better results than the best known procedure in literature. Although the SAT procedure needs a higher CPU time for the same stop criterion (defined as a maximum number of generated schedules), it is able to find solutions which have never been found by the genetic algorithm of Van Peteghem and Vanhoucke (2010). It is worth noting that this genetic algorithm was able to report an average deviation from the critical path of 12.92% when the stop criterion was set to 5,000,000 schedules (not shown in the table). These deviations were found after approximately 200 seconds, which corresponds to the time needed for the 500,000 schedules stop criterion of the SAT procedure. However, the latter procedure reports better results with average deviations of 12.41%. It is also worth mentioning that four new best known solutions

have been found with a stop criterion of 50,000 schedules, and another one with a stop criterion of 500,000 schedules. It should be noted that an increase from e.g. 5,000 to 50,000 schedules (i.e. by a factor 10) does not lead to similar CPU time increase (the increase in CPU is equal to $\frac{25.1}{4.5} = 5.57$ which is lower than a factor 10). This can be explained by the introduction of the learning clauses that gradually avoids the generation of identical conflicts, leading to time savings when repeatedly solving the SAT instances. This is not the case for the J10 to J20 instances, which might indicate that the non-renewable resource constraints are not a constraining factor making the clause learning less relevant. However, it should be noted that the computational results must be placed into the right perspective. Although the table shows that our procedure is able to generate high quality solutions which outperform the best known results found in literature, they often come at a higher computational cost. We have used the number of generated schedules as a stop criterion, since this is widely used in the academic literature. However, this approach assumes that the effort for one schedule is essentially the same in all methods (Kolisch and Hartmann, 2006), which is not the case for our procedure. Due to the often CPU intensive search process during the mode assignment step which evaluates multiple mode assignments per schedule, this assumption is clearly violated and hence, the comparison in number of schedules not always fair. Since a fair and unambiguous comparison of CPU times is very hard, we have chosen to report CPU times in the tables, showing the promising character of our procedure in terms of solution quality, although the computational effort is often much higher.

Table 1.4 Computational results for the PSPLIB J30 dataset under four different stop criteria (number of schedules)

	1,000	5,000	50,000	500,000
This Work	20.15%	14.44%	12.77%	12.41%
	2.8s	4.5s	25.1s	210.1s
Van Peteghem and Vanhoucke (2010)	15.30%	13.75%	13.31%	13.09%
	0.05s	0.24s	2.46s	18.03s

However, in order to make the fair comparison complete, Table 1.5 shows a computational comparison similar to Table 1.4 but now truncated after a predefined running time. Since both algorithms have been developed by the same author(s) and tested on the same computer, it can be reasonably assumed that they have been programmed under the same implementation skills. The first rows of each algorithm display the average deviation from the minimal critical path length truncated after 1, 5, 30, 120 and 300 seconds, while the second rows display the number of instances for which a better solution is found than with the other solution procedure. As an example, the SAT procedure is able to find better solutions for 53 instances under a stop criterion of 30 seconds, while the solution procedure of Van Peteghem and Vanhoucke (2010) finds 34 better solutions. All other solutions have the same solution quality. The results show that the new SAT procedure is competitive with the best procedure currently available in the literature and even outperforms it when the running time stop criteria is set to 30 seconds or higher.

Table 1.5 Computational results for the PSPLIB J30 dataset under five different stop criteria (CPU time, in seconds)

	1 s	5 s	30 s	120 s	300 s
This Work	31.03%	16.66%	12.75%	12.64%	12.54%
	0	8	53	49	42
Van Peteghem and Vanhoucke (2010)	13.40%	13.07%	12.96%	12.88%	12.73%
	454	243	34	20	17

Finally, Table 1.6 reports results for the dataset of Boctor (1993) as the percentage deviation above the minimal critical path length. The procedure is not able to outperform the state-of-the-art procedures. However, when extending the stop criterion to 50,000 schedules, the deviations decreased to 23.26% and 24.42%, for the 50 and 100 activity instances, respectively. A further increase to 500,000 schedules resulted in deviations of 22.87% and 23.11%. However, using the procedure on these problem instances is not so relevant since these instances have no non-renewable resources. Consequently, for these instances, there are no infeasible activity lists/mode assignment combinations, and hence, the advantage of the SAT approach to deal with inconsistencies of non-renewable resources is no longer present.

Table 1.6 Computational results for the Boctor dataset under two different stop criteria

	50		100	
	1,000	5,000	1,000	5,000
This Work	31.37%	25.11%	38.58%	30.03%
Van Peteghem and Vanhoucke (2010)	27.36%	23.41%	29.70%	24.67%
Lova et al. (2009)	24.89%	23.70%	26.96%	24.85%
Alcaraz et al. (2003)	33.83%	26.52%	41.85%	29.16%

1.6 Conclusions

In this chapter, a novel approach has been presented to solve the multi-mode resource-constrained project scheduling problem (MRCPSP). The algorithm splits the problem into a mode assignment step and a single mode project scheduling step. The mode assignment step is solved using a fast and efficient SAT solver. Due to excessive memory requirements, a number of small and straightforward adaptations to this solver have been implemented to solve the SAT problem instances in less memory. The single mode project scheduling step is solved using a current state-of-the-art RCPSP meta-heuristic from literature. When better RCPSP meta-heuristics become available in the literature, they can easily replace the current one, possibly leading to improved solutions.

The computational results for the MRCPSP have shown to be able to generate solutions comparable with the solution quality found by many state-of-the-art procedures, and outperforms them when the procedures run long enough. Moreover, the procedure was able to find better solutions on 5 problem instances under high stop criterion values. In our future research, it will be shown that the novel solution approach has potential to solve numerous extensions to the well-known MRCPSP problem, and hence, the solution approach will be used for alternative problem formulations or problem extensions.

References

Alcaraz J, Maroto C, Ruiz R (2003) Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. J Oper Res Soc 54:614–626

Bailleux O, Boufkhad Y, Roussel O (2006) A translation of pseudo-boolean constraints to SAT. Journal on Satisfiability, Boolean Modeling and Computation 2:191–200

Boctor F (1993) Heuristics for scheduling projects with resource restrictions and several resource-duration modes. Int J Prod Res 31:2547–2558

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112:3–41

Chai D, Kuehlmann A (2005) A fast pseudo-boolean constraint solver. IEEE T Comput Aid D 24:305–317

Coelho J, Vanhoucke M (2011) Multi-mode resource-constrained project scheduling using rcpsp and sat solvers. Eur J Oper Res 213:73–82

Cook S (1971) The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on Theory of computing, pp 151–158

Davis M, Logemann G, Loveland D (1962) A machine program for theorem proving. Communications of ACM 5, 7:394–397

Debels D, De Reyck B, Leus R, Vanhoucke M (2006) A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. Eur J Oper Res 169:638–653

Debels D, Vanhoucke M (2007) A decomposition-based genetic algorithm for the resource-constrained project scheduling problems. Oper Res 55:457–469

Elloumi S, Fortemps P (2010) A hybrid rank-based evolutionary algorithm applied to multi-mode resource-constrained project scheduling problem. Eur J Oper Res 205:31–41

Herroelen W, Demeulemeester E, De Reyck B (1999) A classification scheme for project scheduling problems. In Weglarz J, editor, Project Scheduling - Recent Models, Algorithms and Applications, Kluwer Academic Publishers, Dortrecht, pp 1–26

Hooker J, Vinay V (1995) Branching rules for satisfiability. J Autom Reasoning 15:359–383

Jarboui B, Damak N, Siarry P, Rebai A (2008). A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Appl Math Comput 195:299–308

Józefowska J, Mika M, Rózycki R, Waligóra G, Weglarz J (2001) Simulated annealing for multi-mode resource-constrained project scheduling. Ann Oper Res 102:137–155

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

Kolisch R, Hartmann S (2006) Experimental investigation of heuristics for resource-constrained project scheduling: An update. Eur J Oper Res 174:23–37

Kolisch R, Sprecher A (1996) PSPLIB - a project scheduling problem library. Eur J Oper Res 96:205–216

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manage Sci 41:1693–1703

Kullmann O (2006) The SAT 2005 solver competition on random instances. Journal on Satisfiability, Boolean Modeling and Computation 2:61–102

Li K, Willis R (1992) An iterative scheduling technique for resource-constrained project scheduling. Eur J Oper Res 56:370–379

Lova A, Tormos P, Cervantes M, Barber F (2009) An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. Int J Prod Econ 117:302–316

Markov I, Sakallah K, Ramani A, Aloul F (2002) Generic ILP versus specialized 0-1 ILP: An update. International Conference on Computer-Aided Design (ICCAD '02), pp 450–457

Marques-Silva J, Sakallah K (1999) GRASP: A Search Algorithm for Propositional Satisfiability. IEEE T Comput 48:506–521

Ranjbar M, De Reyck B, Kianfar F (2009) A hybrid scatter-search for the discrete time/resource trade-off problem in project scheduling. Eur J Oper Res 193:35–48

Talbot F (1982) Resource-constrained project scheduling problem with time-resource trade-offs: The nonpreemptive case. Manage Sci 28:1197–1210

Valls V, Ballestin F, Quintanilla S (2004) A population based approach to the resource-constrained project scheduling problem. Ann Oper Res 131:305–324

Valls V, Ballestin F, Quintanilla S (2005) Justification and RCPSP: A technique that pays. Eur J Oper Res 165(2):375–386

Valls V, Ballestin F, Quintanilla S (2008) A hybrid genetic algorithm for the resource-constrained project scheduling problem. Eur J Oper Res 185(2):495–508

Valls V, Quintanilla S, Ballestin F (2003) Resource-constrained project scheduling: A critical activity reordering heuristic. Eur J Oper Res 149:282–301

Van Peteghem V, Vanhoucke M (2010) A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. Eur J Oper Res 201:409–418

Wang L, Fang C (2011) An effective shuffled frog-leaping algorithm for multi-mode resource-constrained project scheduling problem. Inform Sciences 181:4804–4822

Wang L, Fang C (2012) An effective estimation of distribution algorithm for the multi-mode resource-constrained project scheduling problem. Comput Oper Res 39:449–460

Weglarz J, Józefowska J, Mika M, Waligóra G (2011) Project scheduling with finite or infinite number of activity processing modes - a survey. Eur J Oper Res 208:177–205