Modeling and Solving Project Scheduling with Calendars

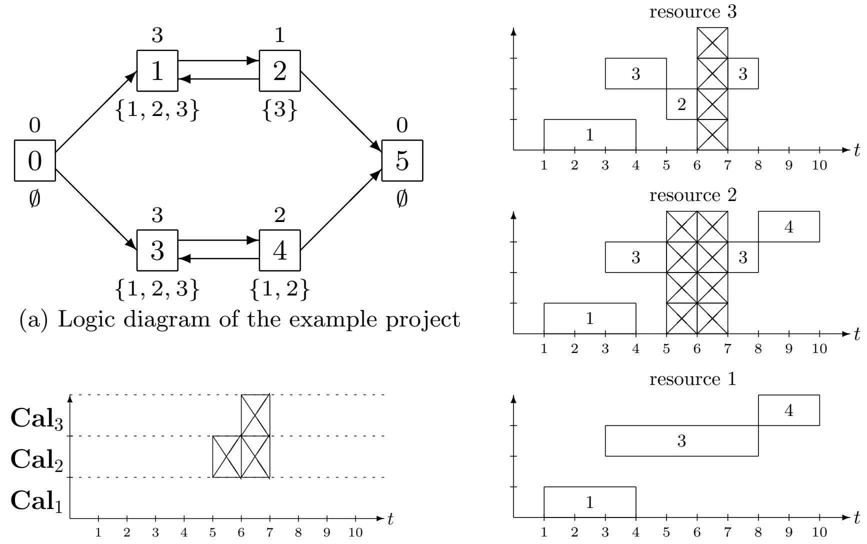
Stefan Kreter[1], Andreas Schutt[2,3], and Peter J. Stuckey[2,3]

[1] Operations Research Group, Institute of Management and Economics,
Clausthal University of Technology, 38678 Clausthal-Zellerfeld, Germany
[2] Optimisation Research Group, National ICT Australia
[3] Department of Computing and Information Systems,
The University of Melbourne, Victoria 3010, Australia
stefan.kreter@tu-clausthal.de,{andreas.schutt,peter.stuckey}@nicta.com.au

Abstract. Resource-constrained project scheduling with the objective of minimizing project duration (RCPSP) is one of the most studied scheduling problems. In this paper we consider the RCPSP with general temporal constraints and calendar constraints. Calendar constraints make some resources unavailable on certain days in the scheduling period and force activity execution to be delayed while resources are unavailable. They arise in practice from, *e.g.*, unavailabilities of staff during public holidays and weekends. The resulting problems are challenging optimization problems. We develop not only four different constraint programming (CP) models to tackle the problem, but also a specialized propagator for the cumulative resource constraints taking the calendar constraints into account. This propagator includes the ability to explain its inferences so it can be used in a lazy clause generation solver. We compare these models, and different search strategies on a challenging set of benchmarks using a lazy clause generation solver. We close 83 of the open problems of the benchmark set, and show that CP solutions are highly competitive with existing Mip models of the problem.

1 Introduction

The resource-constrained project scheduling problem with general temporal and calendar constraints (RCPSP/max-cal) is an extension of the well-known RCPSP and RCPSP/max (see, *e.g.*, [14, Chap. 2]) through calendars. The RCPSP/max-cal can be given as follows. For a set of activities, which require time and renewable resources for their execution, execution time intervals must be determined in a way that minimum and maximum time lags between activities are satisfied, the prescribed resource capacities are not exceeded, and the project duration is minimized. The difference with RCPSP/max is that a calendar is given for each renewable resource type that describes for each time period whether the resource type is available or unavailable. Time periods of unavailability can occur, *e.g.*, due to weekends or public holidays. The activities and time lags are dependent on the resource calendars, too, and some activities can be interrupted for the



(a) Logic diagram of the example project

(b) Time periods of unavailability

(c) Possible resource allocation

Fig. 1: Illustrative Example for RCPSP/max-cal

duration of a break while others cannot be interrupted due to technical reasons. For the interruptible activities a start-up phase is given during which the activity is not allowed to be paused. Concerning the renewable resource types one distinguishes resource types that stay engaged or are blocked, respectively, during interruptions of activities that require it and resource types that are released and can be used to carry out other activities during interruptions.

Our motivation for developing CP models for the RCPSP/max-cal and using lazy clause generation to solve it lies in the very good results obtained by [18,19,20] solving RCPSP and RCPSP/max by lazy clause generation.

Example 1. Figure 1 shows an illustrative example with six activities and three renewable resource types. The project start (activity 0) and the project end (activity 5) are fictitious activities, *i.e.*, they do not require time or resources. A logic diagram of the project is given in Fig. 1(a) where each activity is represented by a node with the duration given above and the set of resource types used by the activity below the node. The arcs between the nodes represent time lags.

The calendars of the three renewable resource types are depicted in Fig. 1(b). If there is a box with an X for a resource type k and time t, then resource type k is not available at time t. Resource type 1 is always available and can be thought of as a machine. Resource types 2 and 3 can be thought of as different kinds of staff where resource type 2 (3) has a five-day (six-day) working week. In addition, assume that resource type 1 stays engaged or is blocked, respectively, during a break of an activity that requires resource type 1 for its execution while resource types 2 and 3 are released during interruptions of activities.

A possible resource allocation of the three renewable resource types is shown in Fig. 1(c). Activity 3 requires all renewable resource types for its execution.

Since resource type 2 is not available in periods 6 and 7, activity 3 is interrupted during these periods. While resource type 1 stays engaged during the interruption, resource type 3 can be used to carry out activity 2 in period 6. □

Few authors have dealt with calendars in project scheduling so far. A time planning method for project scheduling with the same calendar for each resource type is introduced in [23]. In [6] the RCPSP/max with different calendars for each renewable resource type is investigated for the first time but the start-up phase of the interruptible activities are not taken into account. [6] proposes methods to determine the earliest and latest start and completion times for the project activities and priority rule methods. Procedures to determine the earliest and latest start times if a start-up phase is taken into account are presented in [7] and [14, Sect. 2.11]. In addition, they sketch how priority-rule methods for the RCPSP/max can be adapted for calendars. In the approach in [7] and [14, Sect. 2.11] all resources stay engaged during interruptions of activities. Within the priority-rule methods in [6,7], and [14, Sect. 2.11] the procedures to determine the earliest and latest start times must be carried out in each iteration. Recently, a new time planning method, three binary linear model formulations, and a scatter search procedure for the RCPSP/max-cal were developed in [9]. Moreover, Kreter et al. [9] introduce a benchmark test set which is based on the UBO test set for RCPSP/max [8]. The time planning method determines all time and calendar feasible start times for the activities and absolute time lags depending on the start times of the activities once in advance and then uses this throughout the scatter search.

In CP, the works [3,4] respectively propose calendar constraints/rules for ILOG Schedule and Cosytech CHIP. The former [3] was generalized to intensity functions of activities in IBM ILOG CP Optimizer, while breaks of activities extend the length between their start and end times, only resource types that stay engaged can be modeled directly. The latter [4] introduces constraint rules in the global constraint `diffn` for parallel machine scheduling.

A practical application where calendars must be considered as well as other additional constraints can be found in batch scheduling [21]. Problems that are related to the RCPSP/max-cal are treated in [22,5]. An alternative approach to include calendars into project scheduling that makes use of calendar independent start-start, start-end, end-start, and end-end time lags is proposed in [22] and [5] studies the RCPSP with non-preemptive activity splitting, where an activity in process is allowed to pause only when resource levels are temporarily insufficient.

2 Problem description

In this section we describe the RCPSP/max-cal formally and give an example instance. We use identifiers and definitions from [9]. In what follows, we assume that a project consists of a set $V := \{0, 1, \ldots, n, n+1\}$, $n \geq 1$, of activities, where 0 and $n+1$ represent the begin and the end of the project, respectively. Each activity i has a processing time $p_i \in \mathbb{N}_0$. Activities i with $p_i > 0$ are called real activities and the set of real activities is denoted by $V^r \subset V$. Activities 0 and $n+1$ as well as milestones, which specify significant events of the project and have a duration of $p_i = 0$, form the set $V^f = V \setminus V^r$ of fictitious activities.

A project completion deadline $\overline{d} \in \mathbb{N}$ has to be determined in order to define the time horizon of the calendars and the time axis is divided into intervals $[0,1), [1,2), \ldots, [\overline{d}-1, \overline{d})$ where a unit length time interval $[t-1,t)$ is also referred to as time period t. The set of renewable resource types is denoted by $\mathcal{R}$ and for each renewable resource type $k \in \mathcal{R}$ a resource capacity $R_k \in \mathbb{N}$ is given that must not be exceeded at any point in time. The amount of resource type k that is used constantly during the execution of activity $i \in V$ is given by $r_{ik} \in \mathbb{N}_0$. For fictitious activities $i \in V^f$ $r_{ik} := 0$ holds for all $k \in \mathcal{R}$. For each resource type a resource calendar is given.

Definition 1. *A calendar for resource $k \in \mathcal{R}$ is a step function $\mathbf{Cal}_k(\cdot) : [0, \overline{d}) \to \{0,1\}$ continuous from the right at the jump points, where the condition*

$$\mathbf{Cal}_k(t) := \begin{cases} 1, & \text{if period } [\lfloor t \rfloor, \lfloor t+1 \rfloor) \text{ is a working period for } k \\ 0, & \text{if period } [\lfloor t \rfloor, \lfloor t+1 \rfloor) \text{ is a break period for } k \end{cases}$$

is satisfied.

With $\mathcal{R}_i := \{k \in \mathcal{R} \mid r_{ik} > 0\}$ indicating the set of resource types that is used to carry out activity $i \in V$, an activity calendar $\mathbf{C}_i(\cdot) : [0, \overline{d}) \to \{0,1\}$ can be determined from the resource calendars as follows:

$$\mathbf{C}_i(t) := \begin{cases} \min_{k \in \mathcal{R}_i} \mathbf{Cal}_k(t), & \text{if } \mathcal{R}_i \neq \emptyset \\ 1, & \text{otherwise.} \end{cases}$$

Then, for every activity i and a point in time $t \in T := \{0, 1, \ldots, \overline{d}\}$ functions $next\_break_i(t)$ and $next\_start_i(t)$ give the start time and the end time of the next break after time t in calendar $\mathbf{C}_i$, respectively.

$$next\_break_i(t) := \min\{\tau \in T \mid \tau > t \wedge \mathbf{C}_i(\tau) = 0\}$$
$$next\_start_i(t) := \min\{\tau \in T \mid \tau > t \wedge \mathbf{C}_i(\tau) = 1 \wedge \mathbf{C}_i(\tau - 1) = 0\}$$

When calendars are present, we have to distinguish activities that can be interrupted for the duration of a break in the underlying activity calendar and activities that are not allowed to be interrupted. The set of (break-)interruptible activities is denoted by $V^{bi} \subset V$ and the set of non-interruptible activities is given by $V^{ni} = V \setminus V^{bi}$, where $V^f \subseteq V^{ni}$ holds. The execution of an activity $i \in V^{bi}$ must be interrupted at times t with $\mathbf{C}_i(t) = 0$, and the execution must be continued at the next point in time $\tau > t$ with $\mathbf{C}_i(\tau) = 1$. $S_i \in T$ indicates the start time and $E_i \in T$ represents the end of activity $i \in V$. Since the jump points in the calendars $\mathbf{Cal}_k$, $k \in \mathcal{R}$, are all integer valued, the points in time where an activity is interrupted or continued are integer valued, too. The completion time of activity $i \in V$ can be determined by $E_i(S_i) := \min\{t \mid \sum_{\tau=S_i}^{t-1} \mathbf{C}_i(\tau) = p_i\}$. For each activity $i \in V$ a start-up phase $\varepsilon_i \in \mathbb{N}_0$ is given during which activity i is not allowed to be interrupted. For all activities $i \in V^{ni}$ $\varepsilon_i := p_i$ holds. We assume that the underlying project begins at time 0, *i.e.*, $S_0 := 0$. Then, the project duration equals S_{n+1}. In addition, we assume that no activity $i \in V$ can be in execution before the project start, *i.e.*, $S_i \geq 0$, or after the project end, *i.e.*, $E_i \leq S_{n+1}$.

Between the activities a set A of minimum and maximum time lags is given. W.l.o.g. these time lags are defined between the start times of the activities (see [6,9]). For each time lag $\langle i, j \rangle \in A$, a resource set $\mathcal{R}_{ij} \subseteq \mathcal{R}$ and a length $\delta_{ij} \in \mathbb{Z}$ are given, from which we can compute a calendar $\mathbf{C}_{ij}(\cdot) : [0, \overline{d}) \to \{0, 1\}$ for each time lag by

$$\mathbf{C}_{ij}(t) := \begin{cases} \min_{k \in \mathcal{R}_{ij}} \mathbf{Cal}_k(t), & \text{if } \mathcal{R}_{ij} \neq \emptyset \\ 1, & \text{otherwise} \end{cases}$$

i.e., at least tu time units must elapse after the start of activity i before activity j can start where $tu = \min\{t \mid \sum_{\tau=S_i}^{t-1} \mathbf{C}_{ij}(\tau) = \delta_{ij}\}$.

With parameter ρ_k we indicate whether renewable resource types $k \in \mathcal{R}$ stay engaged or are blocked, respectively, during interruptions of activities that require it ($\rho_k = 1$) or are released and can be used to carry out other activities during interruptions ($\rho_k = 0$). A vector $S = (S_0, S_1, \ldots, S_{n+1})$ of all activity start times is called a schedule. Given a schedule S and point in time t the set of all real activities $i \in V^r$ that are started before but not completed at time t is called the active set and can be determined by $\mathcal{A}(S, t) := \{i \in V^r \mid S_i \leq t < E_i(S_i)\}$. Then, the resource utilization $r_k^{\text{cal}}(S, t)$ of resource $k \in \mathcal{R}$ at time t according to schedule S can be computed by

$$r_k^{\text{cal}}(S, t) := \sum_{i \in \mathcal{A}(S,t) \mid \mathbf{C}_i(t) = 1} r_{ik} + \sum_{i \in \mathcal{A}(S,t) \mid \mathbf{C}_i(t) = 0} r_{ik}\, \rho_k.$$

With the introduced notation the following mathematical formulation for the RCPSP/max-cal can be given (cf. [6]):

$$\begin{array}{llll} \text{Minimize} & S_{n+1} & & (1) \\ \text{subject to} & \sum_{t=S_i}^{S_i+\varepsilon_i-1} \mathbf{C}_i(t) = \varepsilon_i & i \in V & (2) \\ & \sum_{t=S_i}^{S_j-1} \mathbf{C}_{ij}(t) - \sum_{t=S_j}^{S_i-1} \mathbf{C}_{ij}(t) \geq \delta_{ij} & \langle i, j \rangle \in A & (3) \\ & r_k^{\text{cal}}(S, t) \leq R_k & k \in \mathcal{R}, t \in T \setminus \{\overline{d}\} & (4) \\ & S_i \in T & i \in V & (5) \end{array}$$

The aim of the RCPSP/max-cal is to find a schedule that minimizes the project makespan (1) and satisfies the calendar constraints (2), time lags (3), and resource capacities (4).

Each project can be represented by an activity-on-node network where each activity $i \in V$ is represented by a node and each time lag $\langle i, j \rangle \in A$ is given by an arc from node i to node j with weights δ_{ij} and $\mathcal{R}_{ij}$. The activity duration as well as the start-up phase is given above node i in an activity-on-node network and the resource requirements of activity $i \in V$ are given below node i. For the case where time lags depend on calendars, the label-correcting and triple algorithm (see, *e.g.*, [2, Sects. 5.4 and 5.6]) can be adapted and integrated in

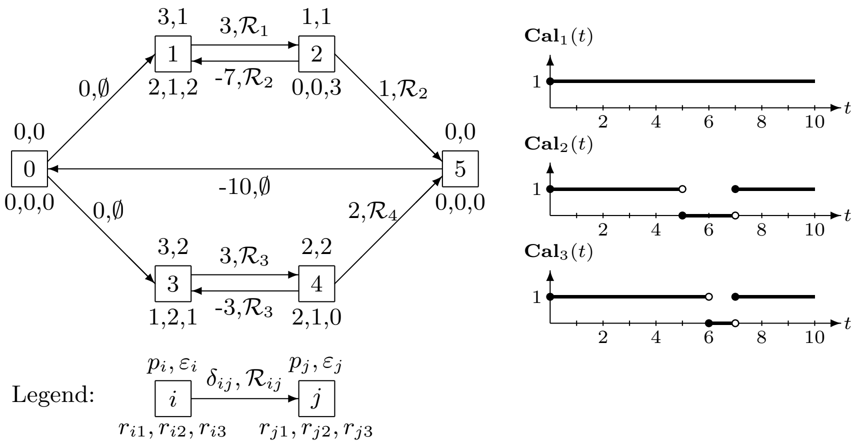


Fig. 2: Activity-on-node network and resource calendars

a time planning procedure that determines a set W_i for each activity $i \in V$ containing all start times that are feasible due to the time lags and calendar constraints, *i.e.*, this procedure determines the solution space of the resource relaxation of the RCPSP/max-cal (problem (1)–(3), (5)) [9]. In addition to the sets W_i, the time planning procedure in [9] determines the "absolute" durations of each activity and time lag with respect to the activities start times. The absolute duration of an activity $i \in V$ is denoted by $p_i(S_i) := E_i(S_i) - S_i$ and the absolute time lag for $\langle i, j \rangle \in A$ by $d_{ij}(t)$ for each $t \in W_i$.

Example 2. Figure 2 shows the problem of Ex. 1 again, but now filled with information for the activites start-up phases and resource requirements as well as information for the time lags.

Activities $0, 2, 4$, and 5 are non-interruptible while activities 1 and 3 form the set V^{bi} and therefore can be interrupted for the duration of a break in the underlying activity calendar. By applying the determination rules from above $\mathbf{Cal}_1 = \mathbf{C}_0 = \mathbf{C}_5 = \mathbf{C}_{01} = \mathbf{C}_{03} = \mathbf{C}_{50}$, $\mathbf{Cal}_2 = \mathbf{C}_1 = \mathbf{C}_3 = \mathbf{C}_4 = \mathbf{C}_{12} = \mathbf{C}_{34} = \mathbf{C}_{43} = \mathbf{C}_{45}$, and $\mathbf{Cal}_3 = \mathbf{C}_2 = \mathbf{C}_{21} = \mathbf{C}_{25}$ hold for the activity and time lag calendars. Since both time lags between activities 3 and 4 depend on the same calendar and $p_3 = \delta_{34} = -\delta_{43}$, activity 4 must be started when activity 3 ends or more precisely at the next point in time after the end of activity 3 where the calendar equals 1. The arc from the project end (node 5) to the project start (node 0) represents an upper bound on the planning horizon of $\overline{d} = 10$.

For the given example the time planning procedure from [9] determines the sets $W_0 = \{0\}$, $W_1 = \{0, 1, 2, 3, 4\}$, $W_2 = \{3, 4, 5, 7, 8, 9\}$, $W_3 = \{0, 2, 3\}$, $W_4 = \{3, 7, 8\}$, and $W_5 = \{5, 6, 7, 8, 9, 10\}$. For example, activity 4 cannot start at times 5 or 6 since there is a break in calendar $\mathbf{C}_4$ from 5 to 7. Moreover, activity 4 cannot start at time 4 because it has to be executed without interruptions. Due to the time lag between activities 3 and 4, activity 3 cannot start at time 1, because if activity 3 started at time 1 activity 4 must start at time 4.

For the time- and calendar-feasible schedule $S = (0, 1, 5, 3, 8, 10)$ the resource profiles are given in Fig. 3. As already mentioned in the introduction resource

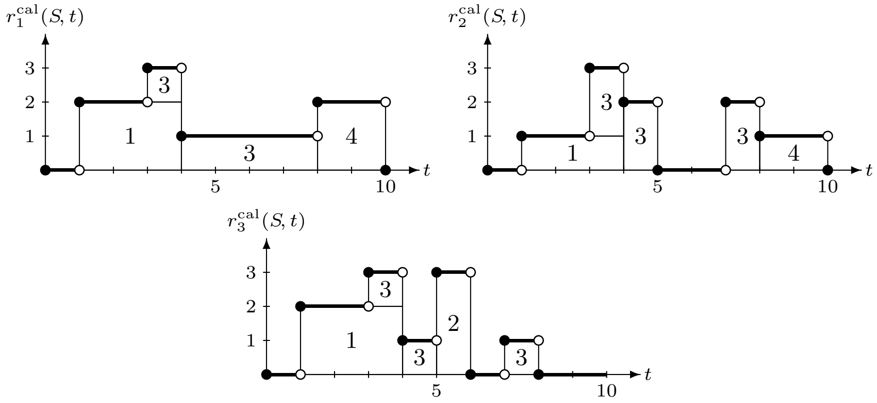


Fig. 3: Resource profiles of schedule $S = (0, 1, 5, 3, 8, 10)$

type 1 stays engaged during interruptions ($\rho_1 = 1$) while resource types 2 and 3 are released during interruptions ($\rho_2 = \rho_3 = 0$). If the inequality $R_k \geq 3$ is fullfilled for each $k \in \mathcal{R}$, schedule S is resource feasible and therefore a feasible solution for the given example. □

3 Models for RCPSP/max-cal

In this section, we present four different ways of modeling the RCPSP/max-cal. The first three approaches use only well-known constraints from finite domain propagation, while a new constraint to model the resource restrictions of the RCPSP/max-cal and a corresponding propagator are used in the fourth model.

3.1 Model `timeidx` (time indexed formulation)

In preprocessing, the time planning procedure of [9] is used to determine the sets W_i of all time- and calendar-feasible start times for each activity $i \in V$ and

$$S_i \in W_i \qquad i \in V \tag{6}$$

must be satisfied. Since the absolute time lags between the activities are dependent on the start time of activity i for each $\langle i, j \rangle \in A$, element constraints are used to ensure that the correct values are taken into account.

$$\texttt{element}(S_i, \boldsymbol{d}_{ij}, d'_{ij}) \qquad \langle i, j \rangle \in A \tag{7}$$

Thereby, $\boldsymbol{d}_{ij}$ is an array that contains for all $S_i \in W_i$ the corresponding $d_{ij}(S_i)$ value. Then, the constraints modelling time lags are

$$S_j - S_i \geq d'_{ij} \qquad \langle i, j \rangle \in A \tag{8}$$

Absolute durations of the activities $i \in V$ are used and the correct assignment is ensured again by element constraints, where $\boldsymbol{p}_i$ is an array containing for all $S_i \in W_i$ the coresponding $p_i(S_i)$ value.

$$\texttt{element}(S_i, \boldsymbol{p}_i, p'_i) \qquad i \in V \tag{9}$$

We implement the resource constraints using a time-indexed decomposition with binary variables b_{it} for each real activity $i \in V^r$ and point in time $t \in T$ where b_{it} is true when i runs at t.

$$b_{it} \leftrightarrow S_i \leq t \wedge t < S_i + p'_i \qquad i \in V^r, t \in T \tag{10}$$

$$\sum\nolimits_{i \in V^r} b_{it}\, r_{ik}\, (\mathbf{C}_i(t) + (1 - \mathbf{C}_i(t))\, \rho_k) \leq R_k \qquad k \in \mathcal{R}, t \in T \tag{11}$$

Model `timeidx` can now be given by: Minimize S_{n+1} subject to (6) – (11).

3.2 Model `2cap` (doubling resource capacity)

Usually global propagators should be used to implement the resource constraints, since more information is taken into account during propagation. This model and the next make use of the global `cumulative` propagator [1] that explains its propagation [16]. If the resource $k \in \mathcal{R}$ under investigation stays engaged during interruptions of activities that require k for their execution, *i.e.*, $\rho_k = 1$, the global cumulative propagator can be used directly with the absolute activity durations. If we regard the absolute duration of each activity $i \in V$ and assume that activity i requires r_{ik} units of resource $k \in \mathcal{R}$ with $\rho_k = 0$ at each point in time $\{S_i, \ldots, E_i(S_i) - 1\}$, there can be resource overloads at break times of an activity even if the corresponding schedule is feasible. One way to handle resources $k \in \mathcal{R}$ with $\rho_k = 0$ is to determine points in time $\mathcal{R}_k^{times}$ where there exist an activity that can be in execution and another activity that can be interrupted, double the resource capacity R_k, introduce a set V_k^d of dummy activities that require exactly R_k units of resource k at each point in time $t \in T \setminus \mathcal{R}_k^{times}$, and use the global cumulative propagator:

$$\texttt{cumulative}(S, p', r_k, R_k) \qquad k \in \mathcal{R} : (\rho_k = 1 \vee \mathcal{R}_k^{times} = \emptyset) \tag{12}$$

$$\texttt{cumulative}(S \cup S^d, p' \cup p^d, r_k \cup r_k^d, 2\, R_k) \quad k \in \mathcal{R} : (\rho_k = 0 \wedge \mathcal{R}_k^{times} \neq \emptyset) \tag{13}$$

Note that r_k is a vector containing the resource requirements on resource k of all activities $i \in V$ and that the vectors S^d, p^d, and r_k^d contain start times, absolute durations, and resource requirements on resource k, respectively, for all $j \in V_k^d$. In addition, some decomposed constraints from (10) and (11) are required to enforce non-overload of resource k at times $\mathcal{R}_k^{times}$.

$$b_{it} \leftrightarrow S_i \leq t \wedge t < S_i + p'_i \qquad i \in V^r, t \in \bigcup\nolimits_{k \in \mathcal{R} : \rho_k = 0} \mathcal{R}_k^{times} \tag{14}$$

$$\sum_{i \in V^r} b_{it} r_{ik}\, \mathbf{C}_i(t) \leq R_k \qquad k \in \mathcal{R} : \rho_k = 0, t \in \mathcal{R}_k^{times} \tag{15}$$

For all $k \in \mathcal{R}$ with $\rho_k = 0$ the set $\mathcal{R}_k^{times}$ is defined as follows.

$$\mathcal{R}_k^{times} := \{t \in T \mid \exists\, i, j \in V : r_{ik} > 0 \wedge r_{jk} > 0 \wedge \min W_i \leq t < E_i(\max W_i) \wedge \\ \min W_j \leq t < E_j(\max W_j) \wedge \mathbf{C}_i(t) \neq \mathbf{C}_j(t)\}$$

Model `2cap` can be achieved by deleting constraints (10) and (11) from model `timeidx` and adding constraints (12)–(15) instead.

Example 3. Regarding the example project from Fig. 2 on page 6, resource 3 is the only resource where $\mathcal{R}_k^{times} \neq \emptyset$. We can see in Fig. 3 on page 7 that in time period 6 activity 2 is in execution and activity 3 is interrupted. Hence $\mathcal{R}_3^{times} = \{5\}$. The solution presented in Fig. 3 is resource feasible for $R_3 = 3$ but `cumulative` does not know that activity 3 is interrupted and detects a resource overload if resource limit $R_3 = 3$ is used. By doubling the resource capacity and introducing a set V_3^d of dummy activities requiring 3 resources in all periods but 6, the `cumulative` of (13) does not detect a resource overload. The reason for the decomposed constraint (15) for time point 5 is clear when we imagine another activity $2'$ that requires resource type 3 for its execution and could be in execution in time period 6 just like activity 2, then for any solution where both activities 2 and $2'$ are in execution in time period 6 there is a resource overload, which the `cumulative` does not detect when the resource capacity is doubled. □

3.3 Model `addtasks` (adding split tasks)

Another way to handle resources $k \in \mathcal{R}$ with $\rho_k = 0$ is to introduce for each interruptible activity $i \in V^{bi}$ a set $Add_i := \{a_1^i, a_2^i, \ldots, a_{|Add_i|}^i\}$ of additional (non-interruptible) activities that cover only those points in time $t \in \{S_i, \ldots, E_i(S_i) - 1\}$ with $\mathbf{C}_i(t) = 1$, *i.e.*, resource k is released during an interruption of activity i. For the start times and processing times of activites $a_j^i \in Add_i$ the following equalities must be guaranteed.

$$S_{a_1^i} = S_i \qquad i \in V^{bi} \tag{16}$$

$$S_{a_j^i} = next\_start_i(S_{a_{j-1}^i}) \qquad i \in V^{bi}, j \in \{2, \ldots, |Add_i|\} \tag{17}$$

$$p_{a_j^i} = \min(next\_break_i(S_{a_j^i}), p_i - \sum_{h=1}^{j-1} p_{a_h^i}) \qquad i \in V^{bi}, j \in \{1, \ldots, |Add_i|\} \tag{18}$$

$$r_{a_j^i,k} = r_{ik} \qquad i \in V^{bi}, j \in \{1, \ldots, |Add_i|\} \tag{19}$$

Thereby, $next\_break_i(t)$ gives the start time of the next break after time t in calendar $\mathbf{C}_i$ and $next\_start_i(t)$ gives the end time of the next break as defined in Sect. 2. Finally, the resource requirement of each additional activity $a_j^i \in Add_i$ is set equal to r_{ik} and the global cumulative propagator can be used:

$$\texttt{cumulative}(S, p', r_k, R_k) \qquad k \in \mathcal{R} : \rho_k = 1 \tag{20}$$

$$\texttt{cumulative}(S^a, p^a, r_k^a, R_k) \qquad k \in \mathcal{R} : \rho_k = 0 \tag{21}$$

In constraints (21), the vectors S^a, p^a, and r_k^a contain not only the start times, durations, and resource requirements of the additional activities a_j^i, $i \in V^{bi}, j \in \{1, \ldots, |Add_i|\}$, but also the start times, durations, and resource requirements of the non-interruptible activities $i \in V^{ni}$.

Model `addtasks` can be achieved by deleting constraints (10) and (11) from model `timeidx` as well as adding constraints (16)–(21) instead.

3.4 Model `cumucal` (global calendar propagator)

For our fourth model for RCPSP/max-cal, we created a global `cumulative` propagator that takes calendars into account and named it `cumulative_calendar`. The fourth model (`cumucal`) can be achieved by deleting constraints (9), (10), and (11) from model `timeidx` as well as adding constraints (22)

$$\texttt{cumulative\_calendar}(S, p, r_k, R_k, \mathbf{C}, \rho_k) \qquad k \in \mathcal{R} \tag{22}$$

with p being the vector of all constant processing times p_i and $\mathbf{C}$ being the vector of all activity calendars $\mathbf{C}_i$, $i \in V$.

The `cumulative_calendar` propagator is made up of two parts, a time-table consistency check and filtering. The basic ideas of these two parts are the same as in the `cumulative` propagator of [18], but non-trivial adaptions were necessary to consider calendars. These adaptions are described in the following. The compulsory part [10] of an activity $i \in V$ is the time interval $[ub(S_i), lb(S_i) + p_i(lb(S_i)))$, where $lb(S_i)$ $(ub(S_i))$ represents the current minimum (maximum) value in the domain of S_i. If $\rho_k = 1$ for the resource $k \in \mathcal{R}$ then activity i requires r_{ik} units of resource k at each point in time of its compulsory part. Otherwise $(\rho_k = 0)$, activity i requires r_{ik} units of resource k only at points in time of its compulsory part where $\mathbf{C}_i(t) = 1$. The intervals where an activity requires resource k within its compulsory part are named the *calendar compulsory parts*. At the begin of the `cumulative_calendar` propagator the calendar compulsory parts of all activities are determined and a resource profile including all these parts is built. Within the consistency check, resource overloads in this profile are detected. If an overload of the resource k occurs in the time interval $[s, e)$ involving the set of activities Ω, the following conditions hold:

$$\begin{aligned} & ub(S_i) \le s \ \wedge \ lb(S_i) + p_i(lb(S_i)) \ge e && i \in \Omega \\ & (1 - \rho_k) \cdot \mathbf{C}_i(t) + \rho_k = 1 && i \in \Omega, t \in [s, e) \\ & \sum_{i \in \Omega} r_{ik} > R_k \end{aligned}$$

In a lazy clause generation solver integer domains are represented using Boolean variables. Each variable x with initial domain $D^0(x) = \{l, \ldots, u\}$ is represented by two sets of Boolean variables $[\![x = d]\!], l \le d \le u$ and $[\![x \le d]\!], l \le d < u$ which define which values are in $D(x)$. A lazy clause generation solver keeps the two representations of the domain in sync. In order to explain the resource overload, we use a pointwise explanation [18] at $TimeD$, which is the nearest integer to the mid-point of $[s, e)$.

$$\forall i \in \Omega : [\![back(i, TimeD + 1) \le S_i]\!] \wedge [\![S_i \le TimeD]\!] \rightarrow false$$

$$back(i, t) := \begin{cases} \max\{\tau \in T \mid \sum_{z=\tau}^{t-1} \mathbf{C}_i(z) = p_i\} & \text{if } \mathbf{C}_i(t-1) = 1 \\ \max\{\tau \in T \mid \sum_{z=\tau}^{t-1} \mathbf{C}_i(z) = p_i - 1\} & \text{if } \mathbf{C}_i(t-1) = 0. \end{cases}$$

The definition by cases for $back(i, t)$ is necessary to guarantee the execution of activity i at time $t-1$, if $S_i = t - back(i, t)$ holds. If for a time t with $\mathbf{C}_i(t-1) = 0$

$back(i,t)$ would be calculated with the first case, then $E_i(t - back(i,t)) < t$ and the explanation would be incorrect.

If there exists a proper subset of activities $\Omega' \subset \Omega$ with $\sum_{i \in \Omega'} r_{ik} > R_k$, the explanation of the resource overload is done on set Ω'. Sometimes more than one such subset exists. In this situation the lexicographic least set of activities is chosen as was done in [18].

Time-table filtering is also based on the resource profile of calendar compulsory parts of all activities. In a filtering without explanations the height of the calendar compulsory parts concerning one time period or a time interval is given. For an activity the profile is scanned through to detect time intervals where it cannot be executed. The lower (upper) bound of an activity's start time is updated to the first (last) possible time period with respect to those time intervals and the activity calendar. If we want to explain the new lower (upper) bound we need to know additionally which activities have the calendar compulsory parts of those time intervals.

A profile is a triple (A, B, C) where $A = [s, e)$ is a time interval, B the set of all activities that have a calendar compulsory part in the time interval A, and C the sum of the resource requirements r_{ik} of all activities in B. Here, we only consider profiles with a maximal time interval A with respect to B and C, *i.e.*, no other profile $([s', e'), B, C)$ exists where $s' = e$ or $e' = s$.

Let us consider the case when the lower bound of the start time variable for activity i can be maximally increased from its current value $lb(S_i)$ to a new value $LB(i)$ using time-table filtering (the case of decreasing upper bounds is analogous and omitted). Then there exists a sequence of profiles $[D_1, \ldots, D_p]$ where $D_h = ([s_h, e_h), B_h, C_h)$ with $e_0 = lb(S_i)$ and $e_p = LB(i)$ such that

$$\forall h : 1 \leq h \leq p; C_h + r_{ik} > R_k \wedge s_h < e_{h-1} + p_i(e_{h-1})$$

In Sect. 2, we introduced $p_i(t)$ only for $t \in W_i$. Note that $p_i(t)$ can be calculated in the same way for $t \notin W_i$, where $p_i(t)$ takes the value $\overline{d} - t$ if less than p_i working periods are following after t in calendar $\mathbf{C}_i$. In addition, if $\rho_k = 0$ is satisfied then

$$\forall h : 1 \leq h \leq p; \exists\, t \in [s_h, e_h) : \mathbf{C}_i(t) = 1$$

Hence each profile D_h pushes the start time of activity i to e_h.

Again we use pointwise explanations based on single time points. Unlike the consistency case, we may need to pick a set of time points no more than the absolute duration of activity i apart to explain the increasing of the lower bound of S_i over the time interval. For a profile with length greater than the absolute processing time of activity i we may need to pick more than one time point in a profile. Let $\{t_1, \ldots, t_m\}$ be a set of time points such that $t_0 = lb(S_i)$, $t_m + 1 = LB(i)$, $\forall 1 \leq l \leq m : t_{l-1} + p_i(t_{l-1}) \geq t_l$ and there exists a mapping $P(t_l)$ of time points to profiles such that $\forall 1 \leq l \leq m : s_{P(t_l)} \leq t_l < e_{P(t_l)}$. Then we build a pointwise explanation for each time point t_l, $1 \leq l \leq m$

$$[\![back(i, t_l + 1) \leq S_i]\!] \wedge \bigwedge_{j \in B_h} ([\![back(j, t_l + 1) \leq S_j]\!] \wedge [\![S_j \leq t_l]\!]) \rightarrow [\![t_l + 1 \leq S_i]\!]$$

Example 4. We illustrate `cumulative_calendar` for the example network from Fig. 2. To explain both the time-table consistency check and the time-table filtering we are using two different cases. For the first case (consistency check), we assume that in the current search node $lb(S_1) = 3, ub(S_1) = 4, lb(S_2) = 8, ub(S_2) = 9, lb(S_3) = ub(S_3) = 3$, and $lb(S_4) = ub(S_4) = 8$ holds, *i.e.*, activities 3 and 4 are already fixed. We examine the `cumulative_calendar` for resource type 1 with a resource capacity of $R_1 = 2$. The calendar compulsory parts are $[4, 8)$ for activity 1, $[3, 8)$ for activity 3, and $[8, 10)$ for activity 4. Note that activity 2 is not taken into account since $r_{21} = 0$ and that the calendar compulsory parts equal the compulsory parts for this example because $\rho_1 = 1$. The compulsory parts of activities 1 and 3 cover the interval $[4, 8)$ and a resource overload of resource 1 occurs, since $r_{11} + r_{31} = 2 + 1 = 3 > 2 = R_1$. A pointwise explanation of the resource overload is done at $TimeD = 6$:

$$[\![3 \leq S_1]\!] \wedge [\![S_1 \leq 6]\!] \wedge [\![3 \leq S_3]\!] \wedge [\![S_3 \leq 6]\!] \rightarrow false$$

For activities $i = 1$ and $i = 3$, respectively, $\mathbf{C}_i(TimeD - 1) = 0$ is satisfied and $back(i, TimeD)$ is calculated through the second case. Without case differentiation for $back(i, t)$ only the first case would be considered, resulting that $back(1, TimeD)$ would equal 1 and the explanation would be wrong.

For the second case (time-table filtering), we assume that in the current search node $lb(S_1) = ub(S_1) = 0, lb(S_2) = 3, ub(S_2) = 8, lb(S_3) = ub(S_3) = 3$, and $lb(S_4) = ub(S_4) = 8$ holds. We examine the `cumulative_calendar` for resource type 3 with a resource capacity of $R_3 = 3$. Activity 2 is the only task where the start time is not fixed and the consistency check detects no resource overload. The calendar compulsory parts are $[0, 3)$ for activity 1, $[3, 5), [7, 8)$ for activity 3, and $[8, 10)$ for activity 4. For the profile (A, B, C) with $A = [3, 5)$, $B = \{3\}$, and $C = 1$ the condition $C + r_{23} = 1 + 3 > 3 = R_3$ is satisfied and therefore the lower bound for variable S_2 can be increased to $LB(2) = 5$. Since the activity duration p_2 equals 1 a pointwise explanation is done for $t_0 = 3$ and $t_1 = 4$. The explanation for $t_0 = 3$ is $[\![3 \leq S_2]\!] \wedge [\![1 \leq S_3]\!] \wedge [\![S_3 \leq 3]\!] \rightarrow [\![4 \leq S_2]\!]$ and for $t_1 = 4$ it is $[\![4 \leq S_2]\!] \wedge [\![2 \leq S_3]\!] \wedge [\![S_3 \leq 4]\!] \rightarrow [\![5 \leq S_2]\!]$. □

3.5 Time Granularity Considerations

All models depend on the granularity chosen for the time. If the granularity increases then the size of T increases respectively. Thus, the number of linear constraints and auxiliary Boolean variables increases for the models `timeidx` and `2cap`, especially for the former. Moreover, filtering algorithms for the element constraints (used in all models) might be negatively affected due to a larger size of the input arrays. The implemented time-table consistency check, filtering, and explanation generation for resource overloads and start time bounds updates in `cumulative_calendar` depend on the granularity, too. Their respective runtime complexity are $\mathcal{O}(x \times y \log(x \times y) + x \times z)$, $\mathcal{O}(x^2 \times z \times y)$, and $\mathcal{O}(x \times z)$ where x is the number of tasks, $y - 1$ is maximal possible number of interruptions of any task and z the maximal possible absolute duration of any task.

4 Experiments and Conclusion

We conducted extensive experiments on Dell PowerEdge R415 machines running CentOS 6.5 with 2x AMD 6-Core Opteron 4184, 2.8GHz, 3M L2/6M L3 Cache and 64 GB RAM. We used MiniZinc 2.0.1 [13] and the lazy clause generation [15] solver `chuffed` rev 707.

A runtime limit of 10 minutes was imposed excluding runtimes needed for pre-processing, initial solution generation, and compiling the MiniZinc models to solver-dependent FlatZinc models. We used the same benchmarks and initial solutions as in [9], which are available at `www.wiwi.tu-clausthal.de/en/chairs/unternehmensforschung/research/benchmark-instances/`.

Since instances with 10 or 20 activities could easily be solved within a few seconds by any combination of solver, model, and search, we concentrate on instances with 50 and 100 activities. The average runtime needed for pre-processing and initial solution generation are less than a few seconds for instances with 50 activities and less than 30 seconds for instances with 100 activities, respectively.

4.1 Comparing Search Strategies

For finding the shortest project duration, we employ a branch-and-bound strategy for which we investigate following four different search combinations. Those seem likely to be most suitable based on our previous experience on solving scheduling problems using lazy clause generation (see, *e.g.*, [18,19,17]).

`ff`: Selects the variable with the smallest domain size and assigns the minimal value in the domain to it.

`vsids`: Selects the literal with the highest activity counter and sets it to true, where the literal is a part of the Boolean representation of the integer variables, *i.e.*, $[\![x = v]\!]$, $[\![x \leq v]\!]$, where x is an integer variable and $v \in \mathcal{D}(x)$. Informally, the activity counter records the recent involvement of the literal in conflicts and all activity counters are simultaneously decayed periodically. The activity counter of a literal is increased during conflict analysis when the literal is related to the conflict. It is an adaption of the variable state independent decaying sum heuristic [12]. The search `vsids` is combined with Luby restarts [11] and a restart base of 100 conflicts.

`hs`: The search starts off with `ff` and then switches to `vsids` after 1000 conflicts.

`alt`: The search alternates between `ff` and `vsids` starting with `ff`. It switches from one to the other after each restart where we use the same restart policy and base as for `vsids`.

Tables 1 and 2 show the results of `chuffed` on the `cumucal` model using different search strategies on instances with 50 and 100 activities, respectively. The search strategies behave similar with the other models. We show the number of instances proven optimal (#opt), not proven optimal but where feasible solutions were found (#feas), proven infeasible (#inf), and where nothing was determined (#un). We compare the average runtime in seconds (avg. rt) and average number

Table 1: Comparison of search strategies on instances with 50 activities.

						cmp(179)		all(180)	
model	search	#opt	#feas	#inf	#un	avg. rt	avg. #cp	avg. rt	avg. #cp
`cumucal`	`alt`	161	0	19	0	1.13	3847	1.71	5236
`cumucal`	`ff`	160	1	19	0	7.54	16401	10.83	16310
`cumucal`	`hs`	161	0	19	0	1.33	5358	1.80	6349
`cumucal`	`vsids`	161	0	19	0	4.27	18495	4.79	19445

Table 2: Comparison of search strategies on instances with 100 activities.

						cmp(140)		all(180)	
model	search	#opt	#feas	#inf	#un	avg. rt	avg. #cp	avg. rt	avg. #cp
`cumucal`	`alt`	158	11	11	0	7.58	11498	56.18	25170
`cumucal`	`ff`	150	16	10	4	13.73	14305	82.51	16420
`cumucal`	`hs`	152	17	11	0	20.33	34900	85.20	45109
`cumucal`	`vsids`	133	36	11	0	76.87	146172	185.61	122457

of choice points to solve (avg. #cp), on two subsets of each benchmark. The cmp subset are all the benchmarks where all solvers proved optimality or infeasibility, and all is the total set of benchmarks.

The `alt` search is clearly the fastest, also leading to the lowest average number of nodes explored in comparison to the rest. Interestingly, the performance of `vsids` significantly decays from instances with 50 activities to those ones with 100 activities in proportion to `alt` and `ff`. This decay also affects `hs`, but not so dramatically. The strength of the `alt` method is the combination of integer based search in `ff` which concentrates on activities that have little choice left, with the robustness of `vsids` which is excellent for proving optimality once a good solution is known.

4.2 Comparing Models

Tables 3 and 4 compare the effect of the different models using `chuffed` and the best search method `alt`. As expected, the time-indexed model, timeidx, is the worst in terms of times due to the large model size, but it propagates effectively as illustrated by the low number of explored nodes (only ever bettered by cumucal). The model addtasks performs worst with respect to the average number of nodes, which can be explained by the shorter activities causing weaker timetable propagation in the `cumulative` propagator. The best model is cumucal that takes the advantage of using fixed durations, since the variability is handled directly by the propagator, and because it generates the smallest model. We also show the average flattening time (avg. ft.) for all benchmarks, where clearly cumucal is advantageous.

4.3 Comparing Solvers

Table 5 compares the results obtained by `chuffed` to those obtained by Opturion CPX 1.0.2 (`ocpx`), which is available at `www.opturion.com/cpx`, and the best

Table 3: Comparison of models on instances with 50 activities.

						cmp(177)		all(180)		
model	search	#opt	#feas	#inf	#un	avg. rt	avg. #cp	avg. ft	avg. rt	avg. #cp
`addtasks`	`alt`	160	1	19	0	9.10	14232	0.84	15.39	17924
`cumucal`	`alt`	161	0	19	0	0.92	3203	0.48	1.71	5236
`timeidx`	`alt`	158	3	19	0	22.20	3484	18.65	31.82	3426
`2cap`	`alt`	161	0	19	0	1.55	5341	0.72	3.12	9619

Table 4: Comparison of models on instances with 100 activities.

						cmp(138)		all(180)		
model	search	#opt	#feas	#inf	#un	avg. rt	avg. #cp	avg. ft	avg. rt	avg. #cp
`addtasks`	`alt`	139	28	10	3	25.88	26344	4.05	139.47	35525
`cumucal`	`alt`	158	11	11	0	3.24	5037	2.45	56.18	25170
`timeidx`	`alt`	131	38	10	1	83.37	4947	78.24	196.63	4031
`2cap`	`alt`	153	16	11	0	6.13	8798	4.05	78.96	30728

Table 5: Comparison of solvers on instances with 50 activities.

						cmp(170)		all(180)		
model	search	#opt	#feas	#inf	#un	avg. rt	avg. #cp	avg. ft	avg. rt	avg. #cp
`cumucal`	`chuffed+alt`	161	0	19	0	0.59	2198	0.48	1.71	5236
`timeidx`	`chuffed+alt`	158	3	19	0	10.45	1662	18.65	31.82	3426
`timeidx`	`ocpx+free`	159	2	19	0	69.95	13383	19.97	83.92	14155
`mip`		153	7	18	2	222.42	—	—	750.25	—

solution obtained by any mixed-integer linear programming formulation from [9] (`mip`), which is solved using CPLEX 12.6 on an Intel Core i7 CPU 990X with 3.47 GHz and 24GB RAM under Windows 7. For `mip` the runtime limit was set to 3 hours and 8 threads were used. To get an idea of the impact of the machine used, we also ran `ocpx` with the deterministic search `ff` on the same Windows machine. `ocpx` was more than 3 times faster on that machine. It can be seen that `chuffed` and `ocpx` clearly outperform the state-of-the-art solution approach, which is `mip`, and that the machine we used is even slower than the machine used in [9].

Overall the cumucal model closes all open benchmarks of size 50 and 75 of size 100, and clearly, we significantly advance the state of the art.

Acknowledgements NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. This work was partially supported by Asian Office of Aerospace Research and Development (AOARD) grant FA2386-12-1-4056.

References

1. Aggoun, A., Beldiceanu, N.: Extending CHIP in order to solve complex scheduling and placement problems. Mathematical and Computer Modelling 17(7), 57–73 (1993)
2. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice Hall, Englewood Cliffs (1993)
3. Baptiste, P.: Constraint-Based Scheduling: Two Extensions. Master's thesis, University of Strathclyde, Glasgow, Scotland, United Kingdom (1994)
4. Beldiceanu, N.: Parallel machine scheduling with calendar rules. International Workshop on Project Management and Scheduling (1998)
5. Cheng, J., Fowler, J., Kempf, K., Mason, S.: Multi-mode resource-constrained project scheduling problems with non-preemptive activity splitting. Computers & Operations Research 53, 275–287 (2015)
6. Franck, B.: Prioritätsregelverfahren für die ressourcenbeschränkte Projektplanung mit und ohne Kalender. Shaker, Aachen (1999)
7. Franck, B., Neumann, K., Schwindt, C.: Project scheduling with calendars. OR Spektrum 23, 325–334 (2001)
8. Franck, B., Neumann, K., Schwindt, C.: Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. OR Spektrum 23, 297–324 (2001)
9. Kreter, S., Rieck, J., Zimmermann, J.: Models and solution procedures for the resource-constrained project scheduling problem with general temporal constraints and calendars. Submitted to European Journal of Operational Research (2014)
10. Lahrichi, A.: Scheduling: The notions of hump, compulsory parts and their use in cumulative problems. Comptes Rendus de l'Académie des Sciences. Paris, Série 1, Matématique 294(2), 209–211 (1982)
11. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. Information Processing Letters 47, 173–180 (1993)
12. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of Design Automation Conference – DAC 2001. pp. 530–535. ACM, New York, NY, USA (2001)
13. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a standard CP modelling language. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming CP 2007. Lecture Notes in Computer Science, vol. 4741, pp. 529–543. Springer Berlin Heidelberg (2007)
14. Neumann, K., Schwindt, C., Zimmermann, J.: Project Scheduling with Time Windows and Scarce Resources. Springer, Berlin, 2 edn. (2003)
15. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints 14(3), 357–391 (2009)
16. Schutt, A.: Improving Scheduling by Learning. Ph.D. thesis, The University of Melbourne (2011), `http://repository.unimelb.edu.au/10187/11060`
17. Schutt, A., Feydy, T., Stuckey, P.J.: Explaining time-table-edge-finding propagation for the cumulative resource constraint. In: Gomes, C.P., Sellmann, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Lecture Notes in Computer Science, vol. 7874, pp. 234–250. Springer Berlin Heidelberg (2013)
18. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints 16(3), 250–282 (2011)

19. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving RCPSP/max by lazy clause generation. Journal of Scheduling 16(3), 273–289 (2013)
20. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: A satisfiability solving approach. In: Schwindt, C., Zimmermann, J. (eds.) Handbook on Project Management and Scheduling, Vol. 1, pp. 135–160. Springer International Publishing (2015)
21. Schwindt, C., Trautmann, N.: Batch scheduling in process industries: An application of resource-constrained project scheduling. OR Spektrum 22, 501–524 (2000)
22. Trautmann, N.: Calendars in project scheduling. In: Fleischmann, B., Lasch, R., Derigs, U., Domschke, W., Rieder, U. (eds.) Operations Research Proceedings 2000. pp. 388–392. Springer, Berlin (2001)
23. Zhan, J.: Calendarization of timeplanning in MPM networks. ZOR – Methods and Models of Operations Research 36, 423438 (1992)