Constraints

# Using Constraint Programming for Solving RCPSP/max-cal

--Manuscript Draft--

| | |
|---|---|
| **Manuscript Number:** | CONS-D-16-00021 |
| **Full Title:** | Using Constraint Programming for Solving RCPSP/max-cal |
| **Article Type:** | Original Research |
| **Keywords:** | RCPSP/max+cal;  Constraint Programming;  Lazy Clause generation;  Calendars;  General Temporal Constraints |
| **Abstract:** | Resource-constrained project scheduling with the objective of minimizing project duration (RCPSP) is one of the most studied scheduling problems.<br>In this paper we consider the RCPSP with general temporal constraints and calendar constraints.<br>Calendar constraints make some resources unavailable on certain days in the scheduling period and force activity execution to be delayed while resources are unavailable.<br>They arise in practice from, e.g., unavailabilities of staff during public holidays and weekends.<br>The resulting problems are challenging optimization problems.<br>We develop not only six 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.<br>This propagator includes the ability to explain its inferences so it can be used in a lazy clause generation solver.<br>We compare these models, and different search strategies on a challenging set of benchmarks using the lazy clause generation solver Chuffed and IBM CPLEX CP Optimizer, respectively.<br>We close all but 8 of the open problems of the benchmark set, extend the benchmark set by instances with up to 500 activities, and show that CP solutions are highly competitive with existing MIP models of the problem. |

**Noname manuscript No.**
(will be inserted by the editor)

# Using Constraint Programming for Solving RCPSP/max-cal

**Stefan Kreter · Andreas Schutt · Peter J. Stuckey**

Received: date / Accepted: date

**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 six 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 the lazy clause generation solver CHUFFED and IBM CPLEX CP Optimizer, respectively. We close all but 8 of the open problems of the benchmark set, extend the benchmark set by instances with up to 500 activities, and show that CP solutions are highly competitive with existing MIP models of the problem.

A preliminary version of this paper appears as Kreter et al (2015).

S. Kreter
Operations Research Group, Institute of Management and Economics,
Clausthal University of Technology, 38678 Clausthal-Zellerfeld, Germany
E-mail: stefan.kreter@tu-clausthal.de

A. Schutt · P.J. Stuckey
Decision Sciences, Data61, CSIRO, Australia
E-mail: {andreas.schutt,peter.stuckey}@data61.csiro.au

A. Schutt · P.J. Stuckey
Department of Computing and Information Systems,
The University of Melbourne, Victoria 3010, Australia

**Keywords** RCPSP/max-cal · Constraint Programming · Lazy Clause Generation · Calendars · General Temporal Constraints

## 1 Introduction

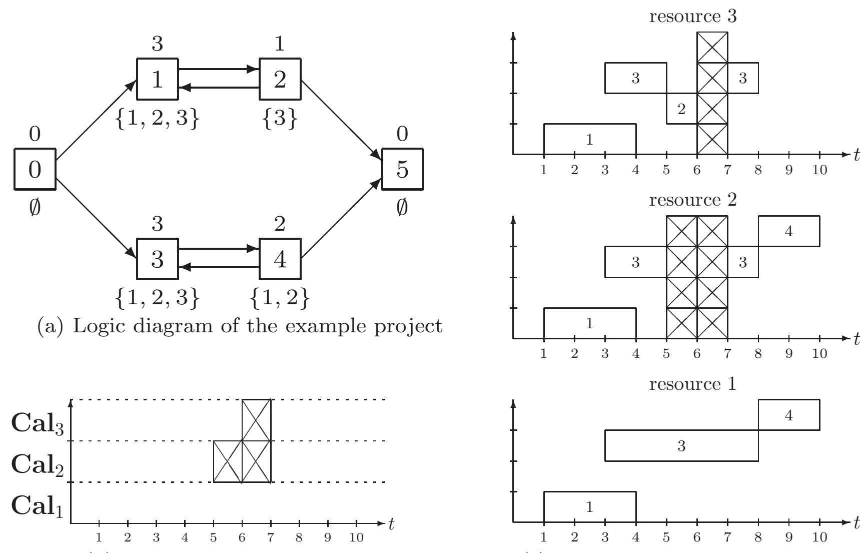
The resource-constrained project scheduling problem with general temporal [1] and calendar constraints (RCPSP/max-cal) is an extension of the well-known RCPSP and RCPSP/max (see, *e.g.*, Neumann et al 2003, Chap. 2) through calendars. The RCPSP/max-cal can be given as follows. For a set of activities, 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 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 (LCG) to solve it lies in the very good results obtained by Schutt et al (2011, 2013b, 2015) 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 resource 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

[1] General temporal constraints are also called generalized precedence constraints or minimum and maximum time lags.



(a) Logic diagram of the example project

(b) Time periods of unavailability

(c) Possible resource allocation

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

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 Zhan (1992). In Franck (1999) 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. Franck (1999) 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 Franck et al (2001a) and Neumann et al (2003, Sect. 2.11). In addition, they sketch how priority-rule methods for the RCPSP/max can be adapted for calendars. In the approach in Franck et al (2001a) and Neumann et al (2003, Sect. 2.11) all resources stay engaged during interruptions of activities. Within the priority-rule methods in Franck (1999), Franck et al (2001a), and Neumann et al (2003, 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 Kreter et al (2016). Moreover, Kreter et al (2016) introduce a benchmark test set which is based on the UBO test set for RCPSP/max (cf. Franck et al 2001b). 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 by Baptiste (1994) and Beldiceanu (1998), respectively, propose calendar constraints/rules for ILOG Schedule and Cosytech CHIP. The former (Baptiste 1994) was generalized to intensity functions of activities in IBM CPLEX 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 (Beldiceanu 1998) 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 (cf. Schwindt and Trautmann 2000). Problems that are related to the RCPSP/max-cal are treated in Trautmann (2001) and Cheng et al (2015). 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 Trautmann (2001) and Cheng et al (2015) study the RCPSP with non-preemptive activity splitting, where an activity in process is allowed to pause only when resource levels are temporarily insufficient.

The paper is organised as follows. In Sect. 2, we give a formal problem description of the RCPSP/max-cal, introduce important concepts, and present a detailed example. In Sect. 3, six different ways to model the RCPSP/max-cal are presented. The first five approaches use propagators that are able to explain their inferences and, therefore, can be used in a lazy clause generation solver. More precisely, 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 and fifth model. This propagator is named `cumulative_calendar`. The sixth model shows how RCPSP/max-cal can be modeled within IBM CPLEX CP Optimizer. In Sect. 4 the `cumulative_calendar` propagator, that is made up of the four parts time-table consistency check and filtering as well as a time-table-edge-finding (TtEf) consistency check and filtering, is described in detail. In Sect. 5 we give experimental results of our models and compare them to the best known results on a benchmark set for the RCPSP/max-cal containing instances with up to 100 activities. In addition, we extend this test set by instances with 200 and 500 activities and give also computational results for these instances. Finally, we conclude in Sect. 6.

## 2 Problem description

In this section we describe the RCPSP/max-cal formally and give an example instance. We use identifiers and definitions from Kreter et al (2016). 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 of fictitious activities $V^f = V \setminus V^r$.

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 $\mathbf{C}_i(\cdot) : [0, \overline{d}) \to \{0, 1\}$ is derived 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 $\mathit{next\_break}_i(t)$ and $\mathit{next\_start}_i(t)$ give the start time and the end time of the next break after time $t$ in calendar $\mathbf{C}_i$, respectively.

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

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.*, $E_i \leq S_{n+1}$.

A set $A$ of minimum and maximum time lags is given between pairs of activities. W.l.o.g. these time lags are defined between the start times of the activities. A detailed description how various time lags can be converted into start to start ones and corresponding examples can be found in Franck (1999) and Kreter et al (2016). 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, similar to the activity calendars, 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 $tu_{ij}(S_i)$ time units must elapse after the start of activity $i$ before activity $j$ can start where $tu_{ij}(S_i) = \min\{t \mid \sum_{\tau=S_i}^{t-1} \mathbf{C}_{ij}(\tau) = \delta_{ij}\} - S_i$ if $\delta_{ij} > 0$ and $tu_{ij}(S_i) = \min\{t \mid \sum_{\tau=t}^{S_i-1} \mathbf{C}_{ij}(\tau) = -\delta_{ij}\} - S_i$ if $\delta_{ij} \leq 0$.

With parameter $\rho_k$ 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. Franck 1999):

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

$$\text{subject to} \quad \sum_{t=S_i}^{S_i+\varepsilon_i-1} \mathbf{C}_i(t) = \varepsilon_i \qquad i \in V \tag{2}$$

$$\sum_{t=S_i}^{S_j-1} \mathbf{C}_{ij}(t) - \sum_{t=S_j}^{S_i-1} \mathbf{C}_{ij}(t) \geq \delta_{ij} \qquad \langle i,j\rangle \in A \tag{3}$$

$$r_k^{\text{cal}}(S,t) \leq R_k \qquad k \in \mathcal{R}, t \in T \setminus \{\overline{d}\} \tag{4}$$

$$S_i \in T \qquad i \in V \tag{5}$$

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 $\delta_{ij}$ 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.*, Ahuja et al 1993, Sects. 5.4 and 5.6) can be adapted and integrated in 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)) (cf. Kreter et al 2016). In addition to the sets $W_i$, the time planning procedure in Kreter et al (2016) 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 Kreter et al (2016) determines the sets $W_0 = \{0\}$, $W_1 = \{0, 1, 2, 3, 4\}$, $W_2 = \{3, 4, 5, 7, 8, 9\}$, $W_3 =$

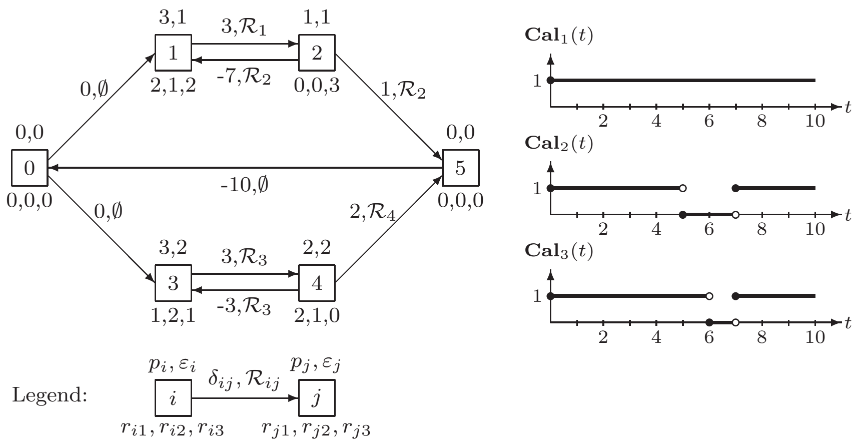


Fig. 2: Activity-on-node network and resource calendars

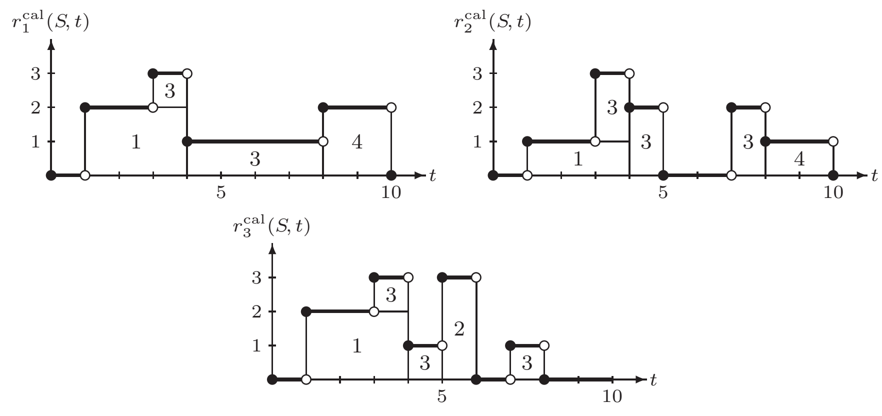


Fig. 3: Resource profiles of schedule $S = (0, 1, 5, 3, 8, 10)$

$\{0, 2, 3\}$, $W_4 = \{3, 7, 8\}$, 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 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. $\square$

## 3 Models for RCPSP/max-cal

In this section, we present six different ways of modeling the RCPSP/max-cal. The first five approaches use propagators that are able to explain their inferences and, therefore, can be used in a lazy clause generation solver. More precisely, 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 and fifth model. The sixth model shows how RCPSP/max-cal can be modeled within IBM CPLEX CP Optimizer.

### 3.1 Model `timeidx` (time indexed formulation)

In preprocessing, the time planning procedure of Kreter et al (2016) 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 $t$.

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

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

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 (cf. Aggoun and Beldiceanu 1993) that explains its propagation (cf. Schutt 2011). 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:

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

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

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 8, resource 3 is the only resource where $\mathcal{R}_k^{times} \neq \emptyset$. We can see in Fig. 3 on page 8 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 Models `cumucal_a` and `cumucal_b` (global calendar propagator)

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

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

with $p$ being the vector of all constant processing times $p_i$ and $\mathbf{C}$ being the vector of all activity calendars $\mathbf{C}_i$, $i \in V$, and resource calendar $\mathbf{Cal}_k$. The `cumulative_calendar` propagator is made up of four parts, a time-table consistency check and filtering as well as a time-table-edge-finding (TtEf) consistency check and filtering, which are described in detail in Sect. 4.

In order to reach a model with a reduced time for preprocessing and a reduced size of the resulting FlatZinc model, we introduce model `cumucal_b`. Instead of running the whole time planning procedure from Kreter et al (2016), only the label correcting algorithm for project scheduling problems with calendars from Franck et al (2001a) is used to determine the earliest and latest feasible start time for each activity $i \in V$. All start times of activity $i \in V$ between the earliest and latest feasible start time for that the start-up phase $\varepsilon_i$ is satisfied are given in set $W'_i$. The difference between sets $W'_i$ and $W_i$ is that $W'_i$ may contain some start times of activity $i$ which are infeasible due to the temporal constraints, *i.e.*, $tu_{ij}(S_i) + tu_{ji}(S_i + tu_{ij}(S_i)) > 0$ can hold for $S_i \in W'_i$ but not for $S_i \in W_i$ since the time planning procedure of Kreter et al (2016) identifies such infeasible start times. Within model `cumucal_b` the mentioned infeasible start times in set $W'_i$ are identified during search of the LCG solver. In addition and in order to reach a small FlatZinc model for formulation `cumucal_b`, within preprocessing we identify temporal constraints that does not depend on the calendars. A temporal constraint $\langle i, j\rangle \in A$ does not depend on any calendar if $\mathbf{C}_{ij}(t) = 1$ holds for all $t \in T$, if $\mathbf{C}_{ij}(t) = \mathbf{C}_i(t)$ holds for all $t \in T$ and $0 < \delta_{ij} \leq \varepsilon_i$, or if $\mathbf{C}_{ij}(t) = \mathbf{C}_j(t)$ holds for all $t \in T$ and $0 < -\delta_{ij} < \varepsilon_j$. Let $A' \subseteq A$ be the set of these temporal constraints $\langle i, j\rangle$ for which $S_j - S_i \geq \delta_{ij}$ guarantees the given time lag because $W'_i$ contains for every activity $i \in V$ only start times that satisfy the start-up phase $\varepsilon_i$. For the other temporal constraints $\langle i, j\rangle \in A \setminus A'$ we determine $tu_{ij}(t)$ for every $t \in T$ and save the values in an array $\boldsymbol{d}_{\mathbf{C}_{ij},\delta_{ij}}$. If there exist two temporal constraints $\langle i, j\rangle$, $\langle i', j'\rangle \in A \setminus A'$ with $\mathbf{C}_{ij} = \mathbf{C}_{i'j'}$ and $\delta_{ij} = \delta_{i'j'}$, then $\boldsymbol{d}_{\mathbf{C}_{ij},\delta_{ij}} = \boldsymbol{d}_{\mathbf{C}_{i'j'},\delta_{i'j'}}$ holds and $\boldsymbol{d}_{\mathbf{C}_{ij},\delta_{ij}}$ needs to appear only once in the data file. Model `cumucal_b` can be given by: Minimize $S_{n+1}$ subject to (22) and

$$S_i \in W'_i \qquad i \in V \qquad (23)$$

$$S_j - S_i \geq \delta_{ij} \qquad \langle i, j\rangle \in A' \qquad (24)$$

$$\texttt{element}(S_i, \boldsymbol{d}_{\mathbf{C}_{ij},\delta_{ij}}, d'_{ij}) \qquad \langle i, j\rangle \in A \setminus A' \qquad (25)$$

$$S_j - S_i \geq d'_{ij} \qquad \langle i, j\rangle \in A \setminus A'. \qquad (26)$$

## 3.5 Model `cpopt` (IBM CPLEX CP Optimizer formulation)

A formulation for solving RCPSP/max-cal with IBM CPLEX CP Optimizer can be given as described in the following. We use several concepts of CP Optimizer which are described in Laborie (2009).

The activities of RCPSP/max-cal are modeled by interval variables (*IloIntervalVar*), the renewable resources by cumulative functions (*IloCumulFunctionExpr*), and the calendars by step functions (*IloNumToNumStepFunction*). Activities (interval variables) are linked to the calendars (step functions) by using the *setIntensity* function of the interval variables. For every activity we forbid that the start or end lies within a calendar break of the corresponding activity calendar using *IloForbidStart* and *IloForbidEnd* constraints, respectively. In addition, we guarantee that activity $i \in V$ starts at a time and calendar feasible point in time by introducing constraints $\min W_i \leq \mathit{IloStartOf}(i) \leq \max W_i$ and $\mathit{IloStartOf}(i) \neq t$ for all $\min W_i < t < \max W_i : t \notin W_i$. Resource requirements of the activities are linked to the resources (cumulative functions) by *IloPulse* constraints. Since *IloPulse* increases the cumulative function at the start of an activity and decreases the cumulative function at the end of an activity it cannot be modeled directly that some resources are released during interruptions of activities that require the resource. If we would set the capacity of the cumulative function to the resource capacity $R_k$ feasible solutions could be excluded from search. Therefore, we identify for every resource with $\rho_k = 0$ points in time $R_k^{times}$ where there exist an activity that can be in execution and another activity that can be interrupted like in model `2cap`. By using *IloAlwaysIn* constraints we set the resource capacity of the regarded cumulative function to $2 \cdot R_k$ at all points in time out of set $R_k^{times}$ and to $R_k$ at all other points in time in the planning horizon. To ensure that infeasible solutions are excluded from search we introduce decomposed constraints similar to constraints (14) and (15) in model `2cap`. Last, temporal constraints are modeled by integer variables (*IloIntVar*) and start-to-start constraints (*IloStartBeforeStart*) using element constraints (*IloElement*) to ensure that the correct values $d'_{ij}$ are taken into account for all $\langle i, j \rangle \in A$.

## 3.6 Time Granularity Considerations

All models depend on the granularity chosen for the time, but some of them are affected less when the time granularity increases. All models include $|V|$ start time variables and the constraints (6)-(8), which are made of $|V|$ membership constraints for the eligible start times and $|A|$ elements and $|A|$ linear constraints for the time lags. Except model `cumucal`, the model also includes $|V|$ integer variables representing the absolute duration depending on the start times, which is modeled by $|V|$ element constraints (9). The sizes of both kind of element constraints also depend on the size of $T$.

The difference lies in the modeling of the resource constraints. Model `timeidx` requires $|\mathcal{R}| \times |T|$ linear constraints, $|V^r| \times |T|$ reified constraints and auxiliary Boolean variables, whereas model `2cap` requires only a subset of these constraints and auxiliary variables and the other two models none of them. All models except model `timeidx` make use of the global constraint `cumulative`. The implemented time-table consistency check and filtering have a respective runtime complexity $\mathcal{O}(x \log x)$ and $\mathcal{O}(x^2)$, where $x$ is the number of (input) tasks. Additional, a runtime $\mathcal{O}(x)$ is needed for explaining a resource overload or a start time bound update. In case of the global constraint `cumulative_calendar` in model `cumucal`, those algorithms have a respective complexity $\mathcal{O}(xy \log(xy) + xz)$, $\mathcal{O}(x^2 zy)$, and $\mathcal{O}(xz)$ where $y - 1$ is maximal possible number of interruptions of any task and $z$ the maximal possible absolute duration of any task. We also implemented the time-table-edge-finding consistency check and filtering having a respective runtime complexity $\mathcal{O}(x^2 z)$ and $\mathcal{O}(x^2 |T|)$. For explanation generation an additional runtime $\mathcal{O}(xz)$ is required. Thus, $z$ increases if the time granularity increases.

## 4 The Global `cumulative_calendar` Propagator

This section presents the new global propagator `cumulative_calendar` with explanation, which is an extension of the global propagator `cumulative` with non-trivial adaptions in order to account for calendar constraints. The new propagator is composed of the time-table (Tt) and time-table-edge-finding (TtEf) propagation whereas each propagation is divided in a consistency check and filtering algorithm. These algorithms are based on the `cumulative` propagator of Schutt et al (2011, 2013a).

Both propagations use the compulsory (mandatory) part of an activity. The *compulsory part* (see Lahrichi 1982) 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.

### 4.1 Time-table Consistency Check

Within the time-table 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 $\Omega$, the following conditions hold:

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

In comparison to standard CP solvers, lazy clause generation solvers also represent integer domains by Boolean variables. Each variable $x$ with initial domain $D^0(x) = \{l, \ldots, u\}$ can be represented by two sets of Boolean variables $[\![x = d]\!], l \leq d \leq u$ and $[\![x \leq d]\!], l \leq d < u$ defining the domain $D(x)$. The former are created when they or their negation appear in the model, whereas the latter are created upfront when the domain size is small otherwise only when they appear in a nogood. 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 (see Schutt et al 2011) at $TimeD$, which is the nearest integer to the mid-point of $[s, e)$.

$$\bigwedge_{i \in \Omega} [\![back(i, TimeD + 1) \leq S_i]\!] \wedge [\![S_i \leq TimeD]\!] \rightarrow false$$

where

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

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 that set $\Omega'$. Sometimes more than one such subset exists. In this situation the lexicographic least set of activities is chosen as in Schutt et al (2011).

## 4.2 Time-table Filtering

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 $(\mathfrak{A}, \mathfrak{B}, \mathfrak{C})$ where $\mathfrak{A} = [s, e)$ is a time interval, $\mathfrak{B}$ the set of all activities that have a calendar compulsory part in the time interval $\mathfrak{A}$, and $\mathfrak{C}$ the sum of the resource requirements $r_{ik}$ of all activities in $\mathfrak{B}$. Here, we only consider profiles with a maximal time interval $\mathfrak{A}$ with respect to $\mathfrak{B}$ and $\mathfrak{C}$, *i.e.*, no other profile $([s', e'), \mathfrak{B}, \mathfrak{C})$ 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 $D_h = ([s_h, e_h), \mathfrak{B}_h, \mathfrak{C}_h)$ with $e_0 = lb(S_i)$ and $e_p = LB(i)$ such that

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

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 $t_0 = lb(S_i)$, $t_m + 1 = LB(i)$, $\forall 1 \leq l \leq m : t_{l-1} + p_i(t_{l-1}) \geq t_l \wedge (1 - \rho_k) \cdot \mathbf{C}_i(t_l) + \rho_k = 1$ 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 $t_l$, $1 \leq l \leq m$

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

Note that the condition $(1 - \rho_k) \cdot \mathbf{C}_i(t_l) + \rho_k = 1$ can be relaxed for the selection of the time points $t_l$ if $\rho_k = 0$, *i.e.*, a time point can be selected for that the activity $i$ has a break. But for that time point it must hold that there would always be a resource overload before and after that time point when the activity $i$ is scheduled at any time in $[back(i, t_l + 1), t_l)$.

*Example 4* We illustrate time-table consistency check and time-table filtering for the example network from Fig. 2. by using two different cases. For the first case (time-table 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 $TimeD = 6$:

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

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 $(\mathfrak{A}, \mathfrak{B}, \mathfrak{C})$ with $\mathfrak{A} = [3, 5)$, $\mathfrak{B} = \{3\}$, and $\mathfrak{C} = 1$ the condition $\mathfrak{C} + r_{23} = 1 + 3 > 3 = R_3$ 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 $[\![3 \le S_2]\!] \wedge [\![1 \le S_3]\!] \wedge [\![S_3 \le 3]\!] \rightarrow [\![4 \le S_2]\!]$ and for $t_1 = 4$ it is $[\![4 \le S_2]\!] \wedge [\![2 \le S_3]\!] \wedge [\![S_3 \le 4]\!] \rightarrow [\![5 \le S_2]\!]$. $\square$

## 4.3 Time-table-edge-finding Consistency Check

TtEf propagation was developed by Vilím (2011) and an explaining version of it for use in lazy clause generation solvers by Schutt et al (2013a). Both papers show that with TtEf good lower bounds for RCPSP can be achieved. In the following we show how the explaining version of TtEf by Schutt et al (2013a) can be adapted to take calendars into account and therefore can be used within `cumulative_calendar`. The basic idea of TtEf is to regard intervals and the unavoidable workload (energy) of all activities within these intervals according to the current domains of the start time variables. If the energy within an interval is higher than the number of working periods of the regarded resource $k$ within the interval times the resource capacity, an inconsistency is detected and explained. The overall workload of an activity $i \in V$ (energy $en_i$) equals its processing time ($p_i$) times its resource requirement ($r_{ik}$) if the resource under consideration is released during breaks of activities that require it ($\rho_k = 0$). If the resource under consideration stays engaged during breaks of activities that require it ($\rho_k = 1$) $en_i$ represents a lower bound on the overall resource usage of activity $i$ since the true resource usage depends on the start time of activity $i$. The following equation holds.

$$en_i := \begin{cases} r_{ik} \cdot p_i, & \text{if } \rho_k = 0 \\ r_{ik} \cdot \min\{p_i(t) \mid t \in W_i \wedge lb(S_i) \le t \le ub(S_i)\}, & \text{if } \rho_k = 1 \end{cases}$$

To compute the energy of activities within a time interval the treatment of activities is split into a fixed and a free part. The energy of the fixed part of activity $i \in V$ is named $en_i^{TT}$ and equals the sum of the lengths of its calendar compulsory parts, $p_i^{TT}$, times the resource requirement, *i.e.*, $en_i^{TT} := p_i^{TT} \cdot r_{ik}$ with $p_i^{TT} := \sum_{t=ub(S_i)}^{E_i(lb(S_i))-1} C_{itk}$ and $C_{itk} := \rho_k + (1-\rho_k)\mathbf{C}_i(t)$. The energy of the free part is $en_i^{EF} := en_i - en_i^{TT}$. The activities with a positive free part are summarized in set $V^{EF} := \{i \in V \mid en_i^{EF} > 0\}$.

For every pair of activities $\{a,b\} \subseteq V^{EF}$ the interval $[s,e)$ with $s := lb(S_a)$ and $e := E_b(ub(S_b))$ is investigated. Set $V^{EF}(a,b) := \{i \in V^{EF} \mid s \leq lb(S_i) \wedge E_i(ub(S_i)) \leq e\}$ contains all activities that have to be executed within interval $[s,e)$. The unavoidable resource usage within interval $[s,e)$ is denoted by $energy(a,b)$ and consists of three parts. The first part is the energy of the calendar compulsory parts in the time window $[s,e)$. This energy is defined by $ttEn(a,b) := ttAfter[s] - ttAfter[e]$ where $ttAfter[\tau] := \sum_{t\geq\tau}\sum_{i\in V: ub(S_i)\leq t<E_i(lb(S_i))} r_{ik}\cdot C_{itk}$ is the total energy of all calendar compulsory parts occurring at time $\tau$ and after. The second part is the energy of the free parts of activities $i \in V^{EF}(a,b)$ and in the third part we consider activities $i \in V^{EF} \setminus V^{EF}(a,b)$ for which a portion of their free part must be run within $[s,e)$. The energy contributed by such activities is defined by

$$rsEn(a,b) := \sum_{i\in V^{EF}\setminus V^{EF}(a,b): s\leq lb(S_i)} \left( r_{ik} \sum_{\max(E_i(lb(S_i)),ub(S_i))}^{e-1} C_{itk} \right).$$

Summarizing we can compute $energy(a,b)$ by

$$energy(a,b) := \sum_{i\in V^{EF}(a,b)} en_i^{EF} + ttEn(a,b) + rsEn(a,b).$$

The TtEf consistency check detects inconsistencies of the form

$$R_k \sum_{t=s}^{e-1} \mathbf{Cal}_k(t) - energy(a,b) < 0,$$

*i.e.*, the break periods of resource $k$ are considered if $\rho_k = 0$. The algorithm we use for the TtEf consistency check is shown in Alg. 1. Since the calendars are already considered in the introduced calculation specifications Alg. 1 is nearly the same as Alg. 1 in Schutt et al (2013a).

The algorithm iterates on each end time in decreasing order. For each end time the algorithm first checks if a resource overload is not possible with this end time (lines 5–6), and if so skips to the next. Otherwise it examines each possible start time, updating the free energy used $E$ for the new interval (lines 14–15), and calculating the energy available $avail$ in the interval (line 16). If this is negative it explains the overload in the interval and returns $false$. If not it updates the minimum available energy and examines the next interval (line 20). In the line 12, it is checked whether a resource overload is not possible with the start time or an earlier one given the end time from the outer loop,

**Algorithm 1:** TtEf consistency check

```
Input: $X$ an array of activities sorted in non-decreasing order of the earliest start
       time.
Input: $Y$ an array of activities sorted in non-decreasing order of the latest
       completion time.
1  $e := \infty$; $minAvail := \infty$;
2  for $y := n$ down to $1$ do
3      $b := Y[y]$;
4      if $E_b(ub(S_b)) = e$ then continue;
5      if $e \neq \infty$ and $minAvail \neq \infty$ and
         $minAvail \geq R_k \cdot (\sum_{t=E_b(ub(S_b))}^{e-1} \mathbf{Cal}_k(t)) - ttAfter[E_b(ub(S_b))] + ttAfter[e]$
         then
6          continue;
7      $e := E_b(ub(S_b))$;
8      $E := 0$; $minAvail := \infty$;
9      for $x := n$ down to $1$ do
10         $a := X[x]$;
11         if $e \leq lb(S_a)$ then continue ;
12         if $minAvail \neq \infty$ and
             $minAvail \leq R_k \cdot (\sum_{t=lb(S_a)}^{e-1} \mathbf{Cal}_k(t)) - E - ttEn(X[1], b) - \sum_{i=1}^{x} en_{X[i]}^{EF}$
             then break ;
13         $s := lb(S_a)$; $lst_a^{EF} := \max(E_a(lb(S_a)), ub(S_a))$;
14         if $E_a(ub(S_a)) \leq e$ then $E := E + en_a^{EF}$ ;
15         else if $lst_a^{EF} < e$ then $E := E + \min(en_a^{EF}, r_{ak} \cdot (\sum_{t=lst_a^{EF}}^{e-1} C_{atk}))$ ;
16         $avail := R_k \cdot (\sum_{t=s}^{e-1} \mathbf{Cal}_k(t)) - E - ttEn(a, b)$;
17         if $avail < 0$ then
18             explainOverload($s$, $e$);
19             return $false$;
20         if $avail < minAvail$ then $minAvail := avail$ ;
21 return $true$;
```

and if so the algorithm terminates with the inner loop and continues with the outer loop. No corresponding check appears in Schutt et al (2013a). Note that both checks (line 5 and 12) reduced the number of considered time intervals for more than 75% in preliminary experiments.

Regarding the complexity, Alg. 1 iterates over at most $n^2$ activity pairings, the data for earliest completion time $E_i(lb(S_i))$, lastest completion time $E_i(ub(S_i))$, total energy of all calendar compulsory parts $ttAfter[.]$, and the number of working days in a specific interval $\sum_{t=i}^{j} \mathbf{Cal}_k(t)$, is pre-computed and is accessible in $O(1)$ for Alg. 1. Currently, the data $\sum_{t=lst_a^{EF}}^{e-1} C_{atk}$ in line 15 is not pre-computed and is calculated in at most $z$ steps where $z$ is the maximal possible absolute duration of any task. Therefore, the overall runtime complexity is $\mathcal{O}(n^2 z)$.

In order to explain a resource overload we need to introduce some more notations. A lower bound $p_i(a, b)$ on the number of time periods that activity

$i \in V$ requires resource $k \in \mathcal{R}$ within time interval $[s, e)$ can be computed by

$$p_i(a,b) := \begin{cases} en_i/r_{ik} & \text{, if } i \in V^{EF}(a,b) \\ \min(en_i/r_{ik}, \sum_{t=ub(S_i)}^{e-1} C_{itk}) & \text{, if } i \notin V^{EF}(a,b) \wedge s \leq lb(S_i) \\ \sum_{t=\max(s,ub(S_i))}^{\min(e,E_i(lb(S_i)))-1} C_{itk} & \text{, else.} \end{cases}$$

The first case is clear, because the activity $i$ must be executed within the time interval. The second case covers when an activity $i$ is partly run within the time interval and counts the time periods within that interval when started at latest as possible. But it also ensures if the resource $k$ stays engaged during breaks and the activity $i$ has more breaks as usual when run as latest that $p_i(a,b)$ does not exceed the lower bound on $p_i$. The third case takes into account the compulsory parts of all other activities that are not covered by the two other cases.

Now, a simple explanation that only considers the current bounds on activities' start times $S_i$ can be given.

$$\bigwedge_{i \in V : p_i(a,b) > 0} [\![lb(S_i) \leq S_i]\!] \wedge [\![S_i \leq ub(S_i)]\!] \rightarrow false$$

This explanation can be generalized by only ensuring that activity $i$ requires resource $k$ for at least $p_i(a,b)$ time units and by taking the extra energy $\Delta := energy(a,b) - 1 - R_k \sum_{t=s}^{e-1} \mathbf{Cal}_k(t)$ of the resource overload into account. We try to maximally widen the bounds of activities $i \in V$ with $p_i(a,b) > 0$, starting with the activities with non-empty free parts. With $\Delta_i$ we denote the number of time periods that $p_i(a,b)$ can be reduced. It holds that $p_i(a,b) \geq \Delta_i \geq 0$ and $\sum_{i \in V : p_i(a,b) > 0} \Delta_i \cdot r_{ik} \leq \Delta$. The two following functions are needed for describing the explanation. The function

$$left_{i,k}^{(a,b)}(x) := \min\{t \in [0, lb(S_i)] \mid \forall t' \in [t, lb(S_i)] \text{ with } \mathbf{C}_i(t') = 1 : \sum_{\tau=\max(t',s)}^{\min(E_i(t'),e)-1} C_{i\tau k} \geq x\}$$

returns the smallest point in time for which activity $i$ requires resource $k$ for at least $x$ time units within the interval $[s, e)$ if activity $i$ starts at that time or after up to a time of $lb(S_i)$ and the function

$$right_{i,k}^{(a,b)}(x) := \max\{t \in [ub(S_i), e] \mid \forall t' \in [ub(S_i), t] \text{ with } \mathbf{C}_i(t') = 1 : \sum_{\tau=\max(t',s)}^{\min(E_i(t'),e)-1} C_{i\tau k} \geq x\}$$

returns the biggest start time of activity $i$ for which $i$ requires resource $k$ for at least $x$ time periods within the interval $[s, e)$ if activity $i$ starts at that time or before down to a time of $ub(S_i)$. Note that for all possible start times $t \in [left_{i,k}^{(a,b)}(x), right_{i,k}^{(a,b)}(x)]$ the activity $i$ requires at least $x$ time units within the interval $[s, e)$. With both functions we formulate the generalized explanation for a resource overload.

$$\bigwedge_{i \in V : p_i(a,b) - \Delta_i > 0} [\![ left_{i,k}^{(a,b)}(p_i(a,b) - \Delta_i) \leq S_i ]\!] \wedge \\ [\![ S_i \leq right_{i,k}^{(a,b)}(p_i(a,b) - \Delta_i) ]\!] \rightarrow false$$

## 4.4 Time-table-edge-finding Filtering

Now, let us regard the case when the lower bound of the start time variable for activity $i$ can be increased from its current value $lb(S_i)$ to a new value $LB(i)$ using TtEf filtering. As for time time-table filtering the case of decreasing upper bounds is analogous and omitted. The basic idea of TtEf lower bounds propagator is to start the activity under consideration at its current earliest start time and investigate if this leads to a resource overload in any time window $[lb(S_a), E_b(ub(S_b)))$ $(\{a, b\} \in V^{EF})$. Then, the lower bound on the start time can be increased. The algorithm we use for TtEf lower bounds propagation is shown in Alg. 2. As the algorithm for TtEf consistency check, Alg. 2 iterates over all latest completion times in decreasing order and for each of this points in time over the earliest start times (in decreasing order as well). Note that Alg. 1 has two checks for skipping loops (see line 5 and 12), as the implemented version of Alg. 2. However, we omitted the rules in the pseudo-code for Alg. 2 to improve the readability.

The used energy $E$ in the current interval is updated in lines 10–16 and the activity $u$ with the largest resource demand before its current calendar compulsory parts *enReqU* is stored. If *enReqU* is greater than the available energy in the interval, $lb(S_u)$ can be increased, since a resource overload would occur if $u$ starts at $lb(S_u)$ (lines 19–25). After all task intervals are visited the bounds are actually changed by *updateBound*. The procedure $explainUpdate(s, e, i, oldbnd, newbnd)$ is used to explain the bound change of activity $i$ from old bound *oldbnd* to *newbnd*. A simple explanation for a lower bound update for activity $u$ from $lb(S_u)$ to $LB(u)$ that only considers the current bounds on activities' start times can be given.

$$[\![ lb(S_u) \leq S_u ]\!] \wedge \bigwedge_{i \in V \setminus \{u\} : p_i(a,b) > 0} [\![ lb(S_i) \leq S_i ]\!] \wedge [\![ S_i \leq ub(S_i) ]\!] \rightarrow [\![ LB(u) \leq S_u ]\!]$$

The given simple explanation can be generalized as described in the case of an resource overload. Thereby, the lower bound for activity $u$ on the left hand side is decreased as much as possible ensuring that the same propagation holds

**Algorithm 2:** TtEf lower bounds propagator on the start times

```
Input: $X$ an array of activities sorted in non-decreasing order of the earliest start time.
Input: $Y$ an array of activities sorted in non-decreasing order of the latest completion time.
1  for $i \in V^{EF}$ do $lb'(S_i) := lb(S_i)$;
2  $e := \infty$; $z := 0$;
3  for $y := n$ down to $1$ do
4      $b := Y[y]$; $e := E_b(ub(S_b))$;
5      $E := 0$; $minAvail := \infty$; $minBegin := -1$; $enReqU := 0$;
6      for $x := n$ down to $1$ do
7          $a := X[x]$;
8          if $e \leq lb(S_a)$ then continue;
9          $s := lb(S_a)$;
10         if $E_a(ub(S_a)) \leq e$ then $E := E + en_a^{EF}$ ;
11         else
12             $ect_a^{EF} := \min(E_a(lb(S_a)), ub(S_a))$; $lst_a^{EF} := \max(E_a(lb(S_a)), ub(S_a))$;
13             $enIn := \min(en_a^{EF}, r_{ak} \cdot (\sum_{t=lst_a^{EF}}^{e-1} C_{atk}))$;
14             $E := E + enIn$;
15             $enReqA := r_{ak} \cdot \sum_{t=lb(S_a)}^{\min(ect_a^{EF}, e)-1} C_{atk}) - enIn$;
16             if $enReqA > enReqU$ then $u := a$; $enReqU := enReqA$;
17         $avail := R_k \cdot (\sum_{t=s}^{e-1} \mathbf{Cal}_k(t)) - E - ttEn(a, b)$;
18         if $avail < minAvail$ then $minAvail := avail$; $minBegin := s$ ;
19         if $enReqU > 0$ and $avail - enReqU < 0$ then
20             $rest := \left\lfloor (avail + \min(en_u, r_{uk} \cdot \sum_{t=ub(S_u)}^{e-1} C_{utk}))/r_{uk} \right\rfloor$;
21             $lbU := \min\{t \in [lb(S_u), e] \mid \sum_{\tau=t}^{e-1} C_{u\tau k} \leq rest\}$;
22             if $lb'(S_u) < lbU$ then
23                 $expl := explainUpdate(s, e, u, lb'(S_u), lbU)$;
24                 $Update[{+}{+}z] := (u, \texttt{lb}, lbU, expl)$;
25                 $lb'(S_u) := lbU$;
26 for $z' := 1$ to $z$ do $updateBound(Update[z'])$ ;
```

when $u$ is started at that decreased lower bound.

$$[\![left_{u,k}^{(a,b)}(1 + \sum_{t=LB(u)}^{e-1} C_{utk}) \leq S_u]\!] \wedge \bigwedge_{i \in V \setminus \{u\} : p_i(a,b) - \Delta_i > 0} [\![left_{i,k}^{(a,b)}(p_i(a,b) - \Delta_i) \leq S_i]\!] \wedge [\![S_i \leq right_{i,k}^{(a,b)}(p_i(a,b) - \Delta_i)]\!] \rightarrow [\![LB(u) \leq S_u]\!]$$

Note that for the available energy units $\Delta$ for widening the bounds $0 \leq \Delta < r_{uk}$ holds.

*Example 5* We illustrate TtEf consistency check for the example project from Fig. 4. In the root node of the search tree the values $lb(S_1) = 2$, $ub(S_1) = 3$, $E_1(lb(S_1)) = 8$, $E_1(ub(S_1)) = 9$ and $lb(S_i) = 2$, $ub(S_i) = 4$, $E_i(lb(S_i)) = 4$, $E_i(ub(S_i)) = 8$ for $i = 2, 3, 4$ can be determined (see also Fig. 5). The only activity with calendar compulsory parts is activity 1 with $[3, 5)$ and $[7, 8)$. We assume that the needed resource $k$ has a capacity of four units, $R_k = 4$, and

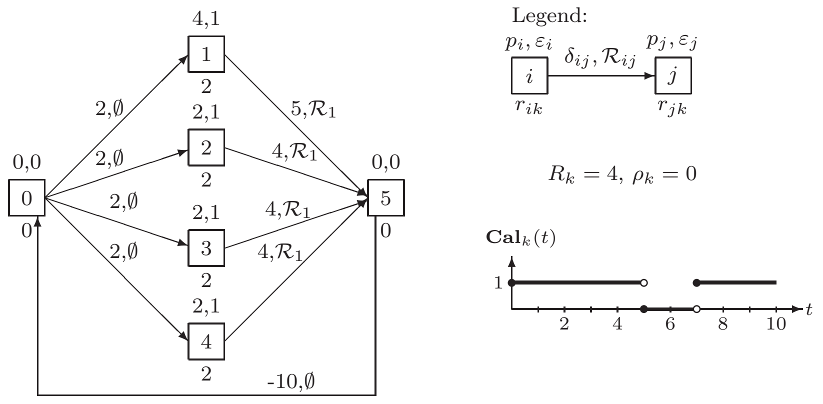


Fig. 4: Example project

is released during breaks of activities that require it, $\rho_k = 0$. When applying time-table consistency check and filtering to the example under consideration, neither inconsistency can be detected nor bounds of the start times can be updated. The set of activities with a positive free part is $V^{EF} = \{1, 2, 3, 4\}$ and we regard the activities $(a, b) = (2, 3)$ with the corresponding interval $[s, e) = [lb(S_2), E_3(ub(S_3))) = [2, 8)$. The energy of the calendar compulsory parts in $[2, 8)$ is $ttEn(2, 3) = 6$. Activities 2, 3, and 4 have to be executed completely within interval $[2, 8)$, *i.e.*, $V^{EF}(2, 3) = \{2, 3, 4\}$, and the energy of the free parts is $en_i^{EF} = en_i = 4$, $i \in V^{EF}(2, 3)$. The only activity in the set $V^{EF} \setminus V^{EF}(2, 3)$ is activity 1 with $\max(E_1(lb(S_1)), ub(S_1)) = 8 = e$. Therefore, $rsEN(2, 3) = 0$ holds and $energy(2, 3) = 12 + 6 + 0 = 18$ can be computed, *i.e.*, at least 18 units of resource $k$ are required within interval $[2, 8)$ to execute the project activities. Since only $R_k \sum_{t=s}^{e-1} \mathbf{Cal}_k(t) = 16$ units of resource $k$ are available within $[2, 8)$, TtEf consistency check detects an inconsistency and, hence, the example project is infeasible.

By changing the value of $\delta_{15}$ from 5 to 4 in the example network, $ub(S_1)$ changes to 4 and the calendar compulsory parts of activity 1 reduce to $[4, 5)$ and $[7, 8)$. Now, $energy(2, 3) = 16$ can be computed and no inconsistency is detected. But starting activity 1 at $lb(S_1) = 2$ or $lb(S_1) + 1 = 3$ would lead to a resource overload within $[2, 8)$ and, therefore, $lb(S_1)$ can be updated to $LB(1) = 4$ using TtEf filtering and the only feasible start time for activity 1 is $LB(1) = ub(S_1) = 4$ (see also Fig. 6). □

Regarding the runtime complexity, Alg. 2 iterates at most over $n^2$ activity pairings as Alg. 1. Computation of the energy required in each considered time interval is done in the same way as in Alg. 1, thus are the corresponding complexities. In addition, Alg. 2 does bookkeeping for checking and performing lower bound updates on start time variables. If an update is possible (lines 19–25) then the new lower bound is calculated by $\min\{t \in [lb(S_u), e] \mid$

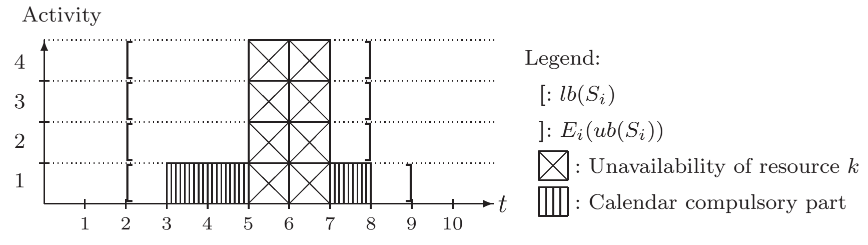


Fig. 5: Lower and upper bounds for the activity start times

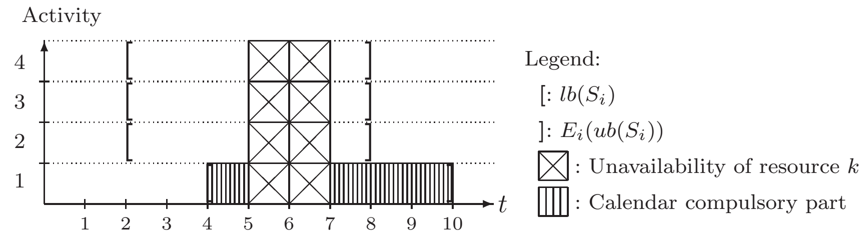


Fig. 6: Lower and upper bounds for the activity start times after TtEf filtering

$\sum_{\tau=t}^{e-1} C_{u\tau k} \leq rest\}$ (line 21). It is computed in $\mathcal{O}(e - lb(S_u))$, which can be approximated by $\mathcal{O}(|T|)$. Thus, the overall worst case complexity is $\mathcal{O}(n^2|T|)$.

## 5 Experiments

We conducted extensive experiments on an Intel Core i7-3820 CPU with 3.60 GHz and 32 GB RAM. We used MiniZinc 2.0.8 (Nethercote et al 2007) and the lazy clause generation (Ohrimenko et al 2009) solver `chuffed` (Chu 2011) rev 707 as well as IBM CPLEX 12.6 CP Optimizer.

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 if not stated differently. For instances with up to 100 activities we used the same benchmarks as in Kreter et al (2016). In addition, we generated test sets with 200 and 500 activities in the exact same manner as Kreter et al (2016) did. All instances as well as corresponding lower and upper bounds on the objective values 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 more activities. Only time granularity investigations in section 5.7 are done on instances with 20 activities.

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

| | | | | | cmp(179) | | all(180) | |
|---|---|---|---|---|---|---|---|---|
| search | opt | feas | inf | unk | avg. rt | avg. cp | avg. rt | avg. cp |
| `alt` | 161 | 0 | 19 | 0 | 0.57 | 4291 | 0.85 | 5711 |
| `ff` | 160 | 1 | 19 | 0 | 3.42 | 16270 | 6.72 | 16180 |
| `hs` | 161 | 0 | 19 | 0 | 0.71 | 5358 | 0.92 | 6349 |
| `vsids` | 161 | 0 | 19 | 0 | 2.16 | 18495 | 2.38 | 19445 |

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

| | | | | | cmp(152) | | all(180) | |
|---|---|---|---|---|---|---|---|---|
| search | opt | feas | inf | unk | avg. rt | avg. cp | avg. rt | avg. cp |
| `alt` | 161 | 8 | 11 | 0 | 4.03 | 14777 | 40.30 | 38358 |
| `ff` | 153 | 13 | 10 | 4 | 8.17 | 18421 | 69.80 | 24273 |
| `hs` | 159 | 10 | 11 | 0 | 19.33 | 70897 | 62.15 | 92406 |
| `vsids` | 145 | 24 | 11 | 0 | 65.63 | 271042 | 138.24 | 237313 |

## 5.1 Comparing Search Strategies within `chuffed`

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.*, Schutt et al 2011, 2013b,a).

`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 (Moskewicz et al 2001). The search `vsids` is combined with Luby restarts (Luby et al 1993) 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_a` model using different search strategies on instances with 50 and 100 activities, respectively. Here, the initial solutions from Kreter et al (2016) were used. Within the global `cumulative_calendar` propagator only time-table consistency check and filtering were activated. The search strategies behave similar with the models `cumucal_b`, `timeidx`, `2cap`, and `addtasks`. 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 (unk). We compare the average runtime in seconds (avg. rt) and average number of choice points to solve (avg. cp), on two subsets of each benchmark. The cmp subset are all the instances where all search strategies proved optimality or infeasibility, and all is the total set of instances.

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.

### 5.2 Comparing Models

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

| | | | | | | | cmp(179) | | all(180) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| model | opt | feas | inf | unk | avg. pt | avg. ist | avg. rt | avg. cp | avg. ft | avg. rt | avg. cp |
| `timeidx` | 160 | 1 | 19 | 0 | 0.06 | 2.40 | 9.18 | 4675 | 6.17 | 12.45 | 4649 |
| `2cap` | 161 | 0 | 19 | 0 | 0.06 | 2.40 | 1.06 | 7295 | 0.34 | 1.75 | 10500 |
| `addtasks` | 160 | 1 | 19 | 0 | 0.06 | 2.40 | 4.28 | 17948 | 0.32 | 7.58 | 17848 |
| `cumucal_a` | 161 | 0 | 19 | 0 | 0.06 | 2.40 | 0.57 | 4291 | 0.26 | 0.85 | 5711 |
| `cumucal_b` | 161 | 0 | 19 | 0 | 0.01 | 0.00 | 0.69 | 5899 | 0.09 | 0.99 | 7521 |

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

| | | | | | | | cmp(153) | | all(180) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| model | opt | feas | inf | unk | avg. pt | avg. ist | avg. rt | avg. cp | avg. ft | avg. rt | avg. cp |
| `timeidx` | 146 | 23 | 11 | 0 | 1.07 | 30.03 | 47.29 | 10198 | 25.34 | 123.35 | 10360 |
| `2cap` | 158 | 11 | 11 | 0 | 1.08 | 30.03 | 6.62 | 19628 | 2.12 | 61.51 | 63519 |
| `addtasks` | 146 | 23 | 11 | 0 | 0.97 | 30.03 | 21.61 | 58483 | 1.90 | 104.58 | 64666 |
| `cumucal_a` | 161 | 8 | 11 | 0 | 1.06 | 30.03 | 3.37 | 12846 | 1.68 | 40.30 | 38358 |
| `cumucal_b` | 160 | 9 | 11 | 0 | 0.06 | 0.00 | 3.80 | 15183 | 0.37 | 42.67 | 43410 |

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_a`). The model `addtasks` performs worst with respect to the average number of nodes, which can be explained by the shorter activities causing weaker time-table propagation in the `cumulative` propagator. The best models are `cumucal_a` and `cumucal_b` that take the advantage of using fixed durations, since the variability is handled directly by the propagator. As expected the average preprocessing time (avg. pt) and the average flattening time (avg. ft) for model `cumucal_b` is significantly lower than for model `cumucal_a`. In addition, for model `cumucal_b` there were no initial solutions given since the initial solution generation from Kreter et al (2016) needs their time planning procedure as a preprocessing step. For all other models the average time for generating the initial solution (avg. ist) is given. Model `cumucal_a` proves the optimality for one more instance with 100 activities than model `cumucal_b` and the average runtime for `cumucal_a` is a bit lower than for `cumucal_b`. Within the global `cumulative` and `cumulative_calendar` propagator, respectively, only time-table consistency check and filtering were activated. In section 5.4 we investigate the influence of TtEf consistency check and filtering within `cumulative_calendar`. But before that, we compare different solvers in the next section.

## 5.3 Comparing Solvers

Table 5 compares the best results obtained by `chuffed` (model `cumucal_a` with `alt` search) to those obtained by IBM CPLEX 12.6 CP Optimizer (for model `cpopt`), Opturion CPX 1.0.2 (`ocpx`), which is available at `www.opturion.com/cpx`, and the best solution obtained by any mixed-integer linear programming formulation from Kreter et al (2016) (`mip`), which is solved using IBM CPLEX 12.6. For `mip` the runtime limit was set to 3 hours and 8 threads were used. For CP Optimizer and Opturion CPX the runtime limit was set to 10 minutes and one thread was used just like for `chuffed`. For `ocpx` the results for model `timeidx` with a `free` search are given in the table, because this combination of model and search led to the best performance for `ocpx`. It can be seen that `chuffed`, `cpopt` and `ocpx` clearly outperform `mip`. All instances with 50 activities could be solved to optimality or infeasibility was proven for `cumucal_a` and `cpopt`. The results for instances with 100 activities of these two models are given in Table 6.

Model `cumucal_a` solved more instances to optimality and the average runtime is lower than for `cpopt`. In addition, `cumucal_a` propagates effectively as illustrated by the lower number of explored nodes. Overall the `cumucal_a` model closes all open benchmarks of size 50 and 75 of size 100, and clearly, we significantly advance the state of the art.

Within the global `cumulative_calendar` propagator only time-table consistency check and filtering were activated.The initial solutions from Kreter et al (2016) were used in all models considered in this section.

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

| | | | | | | cmp(170) | | all(180) | |
|---|---|---|---|---|---|---|---|---|---|
| model | search | opt | feas | inf | unk | avg. rt | avg. cp | avg. rt | avg. cp |
| `cumucal_a` | `chuffed+alt` | 161 | 0 | 19 | 0 | 0.31 | 2506 | 0.85 | 5711 |
| `cpopt` | `cpopt` | 161 | 0 | 18 | 1 | 2.19 | 43247 | 6.60 | 82545 |
| `timeidx` | `ocpx+free` | 159 | 2 | 19 | 0 | 25.06 | 13383 | 32.70 | 14210 |
| `mip` | `mip` | 153 | 7 | 18 | 2 | 229.82 | – | 757.05 | – |

Table 6: Comparison of solvers on instances with 100 activities.

| | | | | | | cmp(163) | | all(180) | |
|---|---|---|---|---|---|---|---|---|---|
| model | search | opt | feas | inf | unk | avg. rt | avg. cp | avg. rt | avg. cp |
| `cumucal_a` | `chuffed+alt` | 161 | 8 | 11 | 0 | 11.82 | 33029 | 40.30 | 38358 |
| `cpopt` | `cpopt` | 152 | 13 | 11 | 4 | 48.27 | 296605 | 100.38 | 624594 |

## 5.4 Investigation of TtEf within `cumulative_calendar`

In this section we investigate the influence of TtEf within the global propagator `cumulative_calendar`. We regard model `cumucal_a` with only time-table consistency check and filtering (`tt`), with time-table consistency check and filtering as well as TtEf consistency check (`ttefc`), and with time-table consistency check and filtering as well as TtEf consistency check and filtering (`ttef`), respectively. In particular, we consider different variants of `ttefc` and `ttef`. A control parameter ttef_prop_factor within `cumulative_calendar` regulates how often the TtEf propagator should be executed. In this paper, we use the values 100 and 0 for ttef_prop_factor, respectively, *i.e.*, for `ttefc-100` and `ttef-100` the TtEf propagator is executed one time in 100 possible execution times and for `ttefc-0` as well as `ttef-0` the TtEf propagator is executed in every possible execution time. For all test runs the `alt` search was used.

Table 7: Investigation of TtEf on instances with 100 activities.

| | | | | | cmp(171) | | cmp10sec(26) | | all(180) | |
|---|---|---|---|---|---|---|---|---|---|---|
| variant | opt | feas | inf | unk | avg. rt | avg. cp | avg. rt | avg. cp | avg. rt | avg. cp |
| `tt` | 161 | 8 | 11 | 0 | 11.08 | 35449 | 135.01 | 427245 | 40.30 | 38358 |
| `ttefc-0` | 160 | 9 | 11 | 0 | 14.31 | 38453 | 178.04 | 474186 | 43.52 | 36530 |
| `ttefc-100` | 161 | 8 | 11 | 0 | 11.93 | 37232 | 146.08 | 450448 | 40.88 | 40186 |
| `ttef-0` | 160 | 9 | 11 | 0 | 18.00 | 34810 | 220.54 | 418704 | 47.03 | 33069 |
| `ttef-100` | 161 | 8 | 11 | 0 | 10.81 | 34075 | 133.15 | 415753 | 39.82 | 37354 |

Table 7 shows besides the cmp subset and the all subset also the subset cmp10sec which contains all the instances that are hard to solve but for that optimality or infeasibility could be proven, *i.e.*, all instances for which all investigated variants of `cumucal_a` have a runtime between 10 and 600 seconds. It can be seen that `ttefc` and `ttef` perform much better if ttef_prop_factor is set to 100 than set to 0. Actually, `ttef-100` performs a bit better than `tt`.

The obtained results are similar if TtEf is used within model `cumucal_b`.

### 5.5 Investigation of instances with 200 and 500 activities

Since nearly all instances with up to 100 activities are solved to optimality or infeasibility was proven by models `cumucal_a` and `cumucal_b` within a runtime limit of 10 minutes, we are interested in the performance of these models on instances with 200 and 500 activities. Therefore, we generated test sets with 200 and 500 activities in the exact same manner as Kreter et al (2016) did. Table 8 shows the results of models `cumucal_a` and `cumucal_b` with the best variants of the `cumulative_calendar` propagator, *i.e.*, `tt` and `ttef-100`, for instances with 200 activities. It can be seen that the aim of `cumucal_b`, which was to reduce the time for preprocessing and flattening, is absolutely reached. In addition, `cumucal_b` solved more instances to proven optimality than `cumucal_a`. Since for `cumucal_a` the flattening for some instances with 500 activities was aborted and already the average preprocessing time is much more than the time limit of 10 minutes, Table 9 shows only the results of model `cumucal_b` on instances with 500 activities.

Table 8: Investigation of instances with 200 activities.

| | | | | | | | cmp(153) | | all(180) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| model | variant | opt | feas | inf | unk | avg. pt | avg. rt | avg. cp | avg. ft | avg. rt | avg. cp |
| `cumucal_a` | `tt` | 132 | 20 | 26 | 2 | 26.45 | 18.58 | 20822 | 13.95 | 94.80 | 26086 |
| `cumucal_a` | `ttef-100` | 132 | 19 | 26 | 3 | 26.45 | 18.95 | 20636 | 13.94 | 93.52 | 23024 |
| `cumucal_b` | `tt` | 133 | 16 | 26 | 5 | 0.34 | 16.95 | 20580 | 1.95 | 95.82 | 36252 |
| `cumucal_b` | `ttef-100` | 135 | 14 | 25 | 6 | 0.34 | 18.28 | 20833 | 1.95 | 93.81 | 36991 |

Table 9: Investigation of instances with 500 activities.

| | | | | | | | cmp(121) | | all(180) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| model | variant | opt | feas | inf | unk | avg. pt | avg. rt | avg. cp | avg. ft | avg. rt | avg. cp |
| `cumucal_b` | `tt` | 116 | 15 | 5 | 44 | 3.63 | 63.13 | 14777 | 22.03 | 238.33 | 9933 |
| `cumucal_b` | `ttef-100` | 117 | 13 | 5 | 45 | 3.63 | 66.70 | 15034 | 22.02 | 239.87 | 10639 |

It can be seen that even a lot of the instances with 500 activities can be solved to optimality or infeasibility within a runtime limit of 10 minutes and that `ttef-100` is competitive with `tt`. However, for a lot of the hard instances nothing was determined within 10 minutes (see column unk). This indicates that `chuffed` with model `cumucal_b` has stretched its limits for instances with 500 activities.

## 5.6 Lower bound computation

Schutt et al (2013a) showed that TtEf propagation within the `cumulative` propagator leads to good results for destructive lower bound computation for RCPSP instances. In this section, we use lower bound computation for the 8, 16, and 58 instances with 100, 200, and 500 activities, respectively, for that neither optimality nor infeasibility could be proven. Our destructive lower bound computation converges to the optimal makespan from below starting with a trivial lower bound, which is the earliest time and calendar feasible start time of the project end. If infeasibility can be proven for the current makespan, then it is increased by 1. If feasibility can be shown for the current makespan, then it is the optimal makespan. Table 10 shows the results for destructive lower bound computation with different variants of model `cumucal_b` with a runtime limit of one minute. The number of instances for which the regarded variant computed the best lower bound (best) and the number of instances for which the regarded variant computed the exclusively best lower bound (excl), *i.e.*, no other variant was able to reach that lower bound, are given. In addition, the number of instances for which optimality could be proven (opt) and the average improvement from the trivial lower bound in percent (dev) are presented.

Table 10: Lower bound computation.

| | 100(8) | | | | 200(16) | | | | 500(58) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| variant | best | excl | opt | dev | best | excl | opt | dev | best | excl | opt | dev |
| `tt` | 3 | 0 | 0 | 35.25 | 11 | 0 | 1 | 13.64 | 43 | 1 | 0 | 4.53 |
| `ttefc-0` | 4 | 0 | 0 | 35.35 | 7 | 0 | 1 | 13.31 | 27 | 0 | 1 | 4.33 |
| `ttefc-100` | 7 | 1 | 0 | 35.67 | 13 | 2 | 1 | 14.05 | 52 | 14 | 1 | 4.55 |
| `ttef-0` | 0 | 0 | 0 | 33.12 | 5 | 0 | 1 | 11.91 | 19 | 0 | 0 | 3.73 |
| `ttef-100` | 6 | 1 | 0 | 35.82 | 11 | 0 | 1 | 13.63 | 43 | 0 | 0 | 4.54 |

Clearly the best variant is `ttefc-100`, which proves optimality for 2 instances, reaches several times the exclusively best lower bound, and improves the trivial lower bounds the most.

## 5.7 Time granularity investigation

In this section, we want to show that the theoretical considerations from section 3.6 are true for our test instances. Therefore, we regard instances with 20 activities and multiply all values that are related to $T$ by a granularity factor, *e.g.*, the completion deadline $\overline{d}$ and the processing times $p_i$, $i \in V$. Table 11 shows the results for models `timeidx`, `2cap`, `addtasks`, and `cumucal_a`.

Table 11: Investigation of time granularity on instances with 20 activities.

| | | | | | | cmp(128) | | all(180) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| model | factor | opt | feas | inf | unk | avg. rt | avg. cp | avg. ft | avg. rt | avg. cp |
| `timeidx` | 1 | 143 | 0 | 37 | 0 | 0.17 | 110 | 0.92 | 0.21 | 209 |
| `timeidx` | 10 | 143 | 0 | 37 | 0 | 3.04 | 239 | 9.98 | 5.91 | 454 |
| `timeidx` | 100 | 102 | 13 | 26 | 39 | 175.66 | 402 | 104.47 | 297.67 | 286 |
| `2cap` | 1 | 143 | 0 | 37 | 0 | 0.01 | 117 | 0.06 | 0.02 | 275 |
| `2cap` | 10 | 143 | 0 | 37 | 0 | 0.07 | 320 | 0.29 | 0.11 | 706 |
| `2cap` | 100 | 143 | 0 | 37 | 0 | 0.99 | 1323 | 2.77 | 1.63 | 2016 |
| `addtasks` | 1 | 143 | 0 | 37 | 0 | 0.03 | 268 | 0.07 | 0.06 | 642 |
| `addtasks` | 10 | 143 | 0 | 37 | 0 | 0.27 | 802 | 0.37 | 0.45 | 1600 |
| `addtasks` | 100 | 143 | 0 | 37 | 0 | 3.41 | 3113 | 3.47 | 6.10 | 4605 |
| `cumucal_a` | 1 | 143 | 0 | 37 | 0 | 0.01 | 92 | 0.05 | 0.01 | 192 |
| `cumucal_a` | 10 | 143 | 0 | 37 | 0 | 0.07 | 288 | 0.29 | 0.10 | 576 |
| `cumucal_a` | 100 | 143 | 0 | 37 | 0 | 0.96 | 1463 | 2.68 | 1.39 | 2214 |

As expected model `timeidx` is strongly influenced by the time granularity. If the granularity factor is set to 100, 52 instances could not be solved to optimality or infeasibility. If one of the global propagators `cumulative` or `cumulative_calendar` is used to model the resource restrictions, the runtime incrementation is not that drastically.

## 6 Conclusion

In this paper we investigate constraint programming with nogood-learning-based solution methods for the resource-constrained project scheduling problem with generalized precedence relations and resource calendars. This is a challenging class of problems for which it can be difficult to find solutions let alone prove optimality. Our contributions are

– investigation of six different CP models
– investigation of four different branching strategies
– adaptation of the cumulative propagator to take calendars into account
– adaption of TtEf propagation to take calendars into account
– closing almost all open problems, but 8
– introduction of new instances with up to 500 activities
– computation of lower bounds

*Acknowledgements* This work was partially supported by Asian Office of Aerospace Research and Development (AOARD) grant FA2386-12-1-4056.

## References

Aggoun A, Beldiceanu N (1993) Extending CHIP in order to solve complex scheduling and placement problems. Mathematical and Computer Modelling 17(7):57–73

Ahuja R, Magnanti T, Orlin J (1993) Network Flows. Prentice Hall, Englewood Cliffs

Baptiste P (1994) Constraint-based scheduling: Two extensions. Master's thesis, University of Strathclyde, Glasgow, Scotland, United Kingdom

Beldiceanu N (1998) Parallel machine scheduling with calendar rules. International Workshop on Project Management and Scheduling

Cheng J, Fowler J, Kempf K, Mason S (2015) Multi-mode resource-constrained project scheduling problems with non-preemptive activity splitting. Computers & Operations Research 53:275–287

Chu GG (2011) Improving combinatorial optimization. PhD thesis, The University of Melbourne, URL http://hdl.handle.net/11343/36679

Franck B (1999) Prioritätsregelverfahren für die ressourcenbeschränkte Projektplanung mit und ohne Kalender. Shaker, Aachen

Franck B, Neumann K, Schwindt C (2001a) Project scheduling with calendars. OR Spektrum 23:325–334

Franck B, Neumann K, Schwindt C (2001b) Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. OR Spektrum 23:297–324

Kreter S, Schutt A, Stuckey PJ (2015) Modeling and solving project scheduling with calendars. In: Pesant G (ed) Principles and Practice of Constraint Programming, Springer International Publishing, Lecture Notes in Computer Science, vol 9255, pp 262–278, DOI 10.1007/978-3-319-23219-5_19, URL http://dx.doi.org/10.1007/978-3-319-23219-5_19

Kreter S, Rieck J, Zimmermann J (2016) Models and solution procedures for the resource-constrained project scheduling problem with general temporal constraints and calendars. European Journal of Operational Research 251(2):387–403

Laborie P (2009) IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. In: Hoeve WJ, Hooker JN (eds) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer Berlin Heidelberg, Lecture Notes in Computer Science, vol 5547, pp 148–162, DOI 10.1007/978-3-642-01929-6_12

Lahrichi A (1982) 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

Luby M, Sinclair A, Zuckerman D (1993) Optimal speedup of Las Vegas algorithms. Information Processing Letters 47:173–180

Moskewicz MW, Madigan CF, Zhao Y, Zhang L, Malik S (2001) Chaff: Engineering an efficient SAT solver. In: Proceedings of Design Automation Conference – DAC 2001, ACM, New York, NY, USA, pp 530–535, DOI 10.1145/378239.379017

Nethercote N, Stuckey PJ, Becket R, Brand S, Duck GJ, Tack G (2007) MiniZinc: Towards a standard CP modelling language. In: Bessière C (ed) Principles and Practice of Constraint Programming  CP 2007, Springer Berlin Heidelberg, Lecture Notes in Computer Science, vol 4741, pp 529–543, DOI 10.1007/978-3-540-74970-7_38

Neumann K, Schwindt C, Zimmermann J (2003) Project Scheduling with Time Windows and Scarce Resources, 2nd edn. Springer, Berlin

Ohrimenko O, Stuckey PJ, Codish M (2009) Propagation via lazy clause generation. Constraints 14(3):357–391

Schutt A (2011) Improving scheduling by learning. PhD thesis, The University of Melbourne, URL http://hdl.handle.net/11343/36701

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2011) Explaining the cumulative propagator. Constraints 16(3):250–282, DOI 10.1007/s10601-010-9103-2

Schutt A, Feydy T, Stuckey PJ (2013a) Explaining time-table-edge-finding propagation for the cumulative resource constraint. In: Gomes CP, Sellmann M (eds) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization

Problems, Springer Berlin Heidelberg, Lecture Notes in Computer Science, vol 7874, pp 234–250, DOI 10.1007/978-3-642-38171-3_16

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2013b) Solving RCPSP/max by lazy clause generation. Journal of Scheduling 16(3):273–289, DOI 10.1007/s10951-012-0285-x

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2015) A satisfiability solving approach. In: Schwindt C, Zimmermann J (eds) Handbook on Project Management and Scheduling, Vol. 1, Springer International Publishing, pp 135–160

Schwindt C, Trautmann N (2000) Batch scheduling in process industries: An application of resource-constrained project scheduling. OR Spektrum 22:501–524

Trautmann N (2001) Calendars in project scheduling. In: Fleischmann B, Lasch R, Derigs U, Domschke W, Rieder U (eds) Operations Research Proceedings 2000, Springer, Berlin, pp 388–392

Vilím P (2011) Timetable edge finding filtering algorithm for discrete cumulative resources. In: Achterberg T, Beck J (eds) Proceedings of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems – CPAIOR 2011, Springer Berlin / Heidelberg, Lecture Notes in Computer Science, vol 6697, pp 230–245, DOI 10.1007/978-3-642-21311-3_22

Zhan J (1992) Calendarization of timeplanning in MPM networks. ZOR – Methods and Models of Operations Research 36:423–438