European Journal of Operational Research 245 (2015) 70–80

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A branch-price-and-cut algorithm for multi-mode resource leveling ☆

Eamonn T. Coughlan [a], Marco E. Lübbecke [b], Jens Schulz [a,*]

[a] Institut für Mathematik, Technische Universität Berlin, MA 5-1, Straße d. 17. Juni 136, 10623 Berlin, Germany
[b] Operations Research, RWTH Aachen University, Kackertstraße 7, 52072 Aachen, Germany

ARTICLE INFO

Article history:
Received 4 October 2013
Accepted 22 February 2015
Available online 5 March 2015

Keywords:
Project scheduling
Resource leveling
Dantzig–Wolfe decomposition
Integer programming
Branch-price-and-cut

ABSTRACT

We propose a new mixed integer programming formulation and solution algorithm for a multi-mode resource-constrained project scheduling problem with availability constraints (calendars) and the objective to minimize the resource availability cost. Our model exploits the problem structure and has an exponential number of variables, which necessitates a column generation approach. The linear programming relaxation is strengthened by adding valid inequalities that need to be carefully separated in order to show the desired effect. Integer optimal solutions are obtained by an exact state-of-the-art branch-price-and-cut algorithm.

Classical time-indexed mixed integer programming formulations for similar problems quite often fail already on instances with only 30 jobs, depending on the network complexity and the total freedom of arranging jobs. A reason is the typically very weak linear programming relaxation. Our model can be interpreted as a non-trivial Dantzig–Wolfe reformulation of a time-indexed formulation. In particular, for larger instances our reformulation gives significantly tighter dual bounds, enabling us to optimally solve instances with 50 multi-mode jobs. This outperforms the classical formulation by far.

1. Introduction

We consider a complex project scheduling problem that is motivated by an industrial application from chemical engineering. It was recently introduced in Megow, Möhring, and Schulz (2011) as *turnaround scheduling*: For the inspection and renewal of parts in chemical plants, these are shut down and disassembled, work is done, and plants are finally rebuilt. This induces a partial ordering of jobs to be done. Jobs are *multi-mode*, that is, they can be sped-up to a certain extent by investing in more workers. The time horizon and the number of workers hired for each job determine production downtime and working cost. We thus face a time-cost tradeoff, and a solution approach to turnaround scheduling is to binary search over the time horizon. For a fixed time horizon—which is assumed throughout this paper—the problem turns into a resource leveling problem, i.e., we need to decide how many workers are hired—this number is then fixed for the entire time horizon.

Workers, or more generally renewable resources, are of different specialization or *type*. Each resource type is associated with *availability periods* that can be thought of as working shifts. We assume that each job needs (possibly several units of) exactly one resource type for the following reason. The processing time of a job decreases with an increased resource usage. A model with multiple resource types per job must encode how the processing time of a job is shortened depending on which resource types are increased. Planners would need to encode all possible combinations of resource assignments to a job, and even worse, estimate the resulting processing times, possibly respecting synergy or interference between resources. It is unrealistic to assume valid data about this in a practical setting on one hand, and hardly doable, if at all, with standard software like MS Project™ on the other hand. Reliable data are difficult to obtain even for one resource type (the actual work increases with increased resource usage due to, e.g., communication overhead). Nonetheless, the need of multiple resources was studied by other authors, e.g., Santos and Tereso (2011) and can be modeled via generalized precedence constraints. These blend easily with our approach, if absolutely needed.

We want to *level* the resources, i.e., we minimize the maximum capacity requirement of each resource type, at a minimum total resource cost. *Balancing* the resource usage, i.e., to flatten the variability of the resource usage over time, is not an issue at our higher-level planning stage.

☆ A preliminary version appeared in Coughlan, Lübbecke, and Schulz (2010). Our new computational results reveal that the tighter dual bounds are a crucial ingredient for solving the proposed instances. With new cutting planes and a customized branching strategy, we increase the critical size of hard instances from before 30 to 50 jobs.

* Corresponding author. BASF SE, Scientific Computing, Carl-Bosch-Straße 31, 67056 Ludwigshafen, Germany. Tel.: +49 621 60 94996.

E-mail addresses: coughlan@math.tu-berlin.de (E.T. Coughlan), marco.luebbecke@rwth-aachen.de (M.E. Lübbecke), jens.schulz@basf.com, jschulz@math.tu-berlin.de (J. Schulz).

http://dx.doi.org/10.1016/j.ejor.2015.02.043
0377-2217/

From a methodological point of view, when it comes to optimally solving larger instances of such scheduling problems, mixed integer programming (MIP) has not fared very well. This is mainly due to the weak dual bounds obtained from linear programming (LP) relaxations of classical models. We therefore propose a model which can be interpreted as a non-trivial Dantzig–Wolfe reformulation of a time-indexed formulation and thus provides much stronger bounds. Variables represent entire sub-schedules, thus are exponential in number, and necessitate a dynamic generation of variables to solve the LP relaxation. Yet, the relation to classical models via Dantzig–Wolfe reformulation enables us to exploit the literature on valid inequalities for scheduling problems, and we demonstrate how to use them to strengthen the formulation. Constraint programming (CP) is usually considered a promising approach for solving scheduling problems, due its expressiveness of logical constraints, and we hybridize some CP elements in our otherwise MIP based approach.

1.1. Our Contribution

We propose a new model for a multi-mode project scheduling problem with availability constraints (working shifts) and the objective to minimize the resource availability cost. Our model is generic in the way resource calendars are respected. For its solution we design a full-fledged branch-price-and-cut algorithm. This does not only produce very strong dual bounds, as did a few previous studies in project scheduling before, but it is also able to provide optimal primal solutions. With this it is the first approach in this area of complex project scheduling that "works on both ends." Our model builds on a non-trivial Dantzig–Wolfe reformulation of a classical time-indexed model, and we exploit the relationship between these two in designing branching rules and deriving cutting planes. We finally conduct an experimental study which shows how the methodological state-of-the-art in branch-and-price should be instantiated in a complex scheduling context. Our implementation is able to optimally solve instances with 50 multi-mode jobs, thus outperforming a state-of-the-art MIP solver by far that is only able to solve 25 percent of the instances we solve.

2. Formal problem description

We assume familiarity with resource-constrained project scheduling (RCPSP) in general; for a recent survey refer to Hartmann and Briskorn (2010). We are given a set $\mathcal{J}$ of non-preemptable jobs and a set $\mathcal{R}$ of renewable resources. Precedence constraints are given as an acyclic digraph $G = (\mathcal{J}, E)$ with $(i, j) \in E$ if and only if job i has to be completed before job j starts. Each job j may be run in exactly one out of a discrete set $\mathcal{M}_j$ of modes. Processing job j in mode $m \in \mathcal{M}_j$ takes p_{jm} time units and requires r_{jmk} units of resource $k \in \mathcal{R}$. Each job needs exactly one resource for execution, so we write r_{jm} if the resource is clear from the context.

All jobs have to be completed before the end of the time horizon T. Each resource $k \in \mathcal{R}$ has a set $\mathcal{I}_k := \{[a_1, b_1], \ldots, [a_{i_k}, b_{i_k}]\}$ of i_k availability periods, also called *shifts*, where $a_1 < b_1 < \cdots < a_{i_k} < b_{i_k}$. A job requiring resource k can only be executed during a time interval $I \in \mathcal{I}_k$. Fig. 1 schematically presents a schedule with two resources and three availability periods each that are indicated by the bold lines. Jobs are represented by boxes, the length of which represents processing time and the height corresponds to resource usage (for simplicity all identical in this example). The arrows indicate precedence constraints. We use a parameter δ_{kt} which is one if resource k is available at time t, i.e., $t \in I$ for some $I \in \mathcal{I}_k$, and zero otherwise. Each resource $k \in \mathcal{R}$ is associated with a per unit cost c_k. For each resource k, the available capacity, which is a variable, is denoted by R_k.

To represent a solution, we denote by $S = (S_1, \ldots, S_n)$ the vector of start times of jobs, and by $M = (m_1, \ldots, m_n)$ the vector of modes in which jobs are executed. For a given *schedule* (S, M), denote by $A(S, M, t) := \{j \in \mathcal{J} : S_j \le t < S_j + p_{jm_j}\}$ the set of jobs *active* at time t. The amount $r_k(S, M, t) := \sum_{j \in A(S,M,t)} r_{jm_jk}$ of resource k used at time t must never exceed the provided capacity R_k. Thus, we obtain resource constraints with *calendars*: $r_k(S, M, t) \le R_k \cdot \delta_{kt}$, $\forall k \in \mathcal{R}$, $\forall t$. In addition to this *resource feasibility* a feasible schedule must obey *precedence feasibility*, i.e., $S_i + p_{im_i} \le S_j$ for all $(i, j) \in E$.

To conclude, we study a multi-mode project scheduling with m renewable resources of unbounded capacity with precedence constraints and working shifts and the objective to minimize the total *resource availability cost*, i.e., minimizing $\sum_{k \in \mathcal{R}} c_k \cdot R_k$.

2.1. Related work

Turnaround scheduling comprises project scheduling with calendars, multi-mode scheduling, and resource leveling; see again Megow et al. (2011) for an industrial application. The zoo of project scheduling problems is large, and we mention only the most related problems. Makespan minimization is a classical scheduling goal. Lower bounding schemes for this objective are presented in Brucker and Knust (2000), where column generation is employed to solve a relaxed problem, allowing preemption and precedence constraints formulated as disjunctions. A variable represents a set of jobs selected to run at a certain point in time. For the case of generalized precedence constraints, lower bounds are derived in Bianco and Caramia (2011) by relaxing resource constraints for jobs which are not precedence related. This allows a dynamic programming approach on a modified activity-on-nodes network. In contrast to minimizing the makespan, other objective functions that measure the variation of resource utilization, e.g., $f(r_k(S, t))$ are of interest in the pre-planning phase; see e.g., Neumann, Schwindt, and Zimmermann (2003).

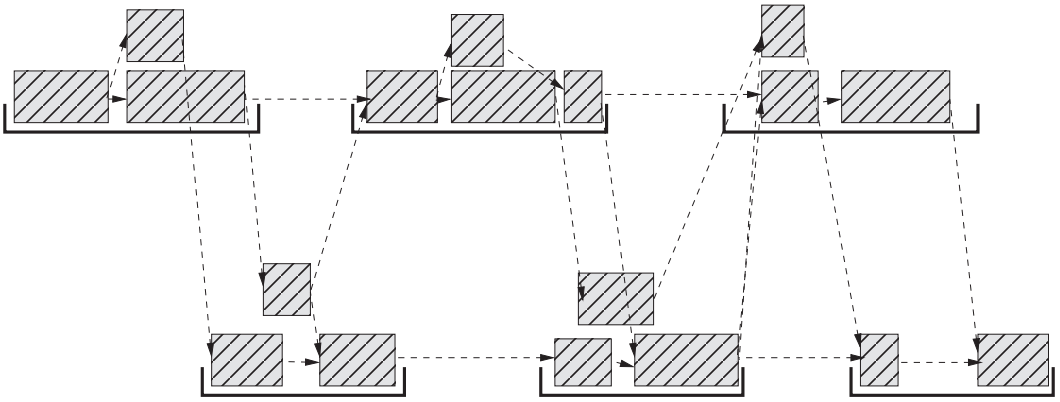


Fig. 1. Schematic representation of turnaround scheduling with two resources. Arrows indicate precedence constraints between jobs; bold lines represent availability periods.

The resource leveling problem with single-modes per job, which is denoted by $PS|temp|\sum c_k \max r_k(S,t)$ with generalized precedence constraints has been considered earlier under the name *resource investment problem*. The special case without generalized precedence constraints $PS|prec|\sum c_k \max r_k(S,t)$ has been considered e.g., in Demeulemeester (1995) and Möhring (1984). These papers competed on the same instance set which contained about 16 jobs and four resources, with a time horizon between 47 and 70 time units. Further computational studies were done containing 15 to 20 jobs and four resources. In the same setting, lower bound computations are proposed in Drexl and Kimms (2001), one based on Lagrangian relaxation, and one based on column generation, where variables represent schedules as in our approach. Twenty jobs with small processing times of about 10 time units can be handled; for 30 jobs the Lagrangian relaxation wins against the column generation approach.

Multi-mode jobs are a key feature of turnaround scheduling. Such problems of the form $MPS|prec|C_{\max}$ have been investigated with renewable and non-renewable resources, with limited capacity, and makespan minimization, known as *multi-mode RCPSP*, see e.g., Demeulemeester and Herroelen (2002) and Hartmann (2001). The notation $MPSm,\infty|prec,shifts|\sum c_k \cdot \max r_k(S,M,t)$ comes close to our model, but in the literature in an MPS problem, a job requires different types of resources.

Calendars have been taken into account in previous algorithms as well. Scheduling problems with fixed processing times and calendars, but without resource capacities, are considered in Zhan (1992). The author provides an exact pseudo-polynomial time algorithm (turned into a polynomial one in Franck, Neumann, and Schwindt (2001)) for computing earliest and latest start times for preemptable as well as non-preemptable jobs.

For a computational benchmarking of project scheduling problems, different problem sets are available in the PSPLib (Kolisch & Sprecher, 1996), where several variants of the RCPSP and of resource investment problems can be found. For the RCPSP single-mode case, test sets containing 60 jobs could not be solved in total by a vast number of researchers. In the multi-mode case, instances with 30 jobs are not solved yet. For the resource investment problem, test sets containing 10, 20, or 30 jobs are available, but they do not contain working shifts, are in single-mode or include generalized precedence constraints. On the other hand a job may need more than one resource type. Even though none of these problem variants is immediately suited for a direct comparison, they are similar to ours, and the mentioned instances inspired us when generating the test set used in this study (see Coughlan et al., 2010 and Section 5).

3. Mixed integer programming formulations

For solving large-scale scheduling problems, mixed integer programming is not considered as primary choice because of typically weak LP relaxations. Moreover, huge numbers of variables (in particular for time-indexed formulations) and constraints may result in high computation times and memory failures when solving even only the LP relaxation. The approach we propose demonstrates that more sophisticated algorithmic techniques can be a partial remedy for these issues. In the following, we assume the reader to be familiar with solving MIPs by branch-and-bound, see e.g., Achterberg (2009), and the basics of column generation and branch-and-price, see e.g., Desrosiers and Lübbecke (2005).

3.1. Shortcomings of time-indexed mixed integer programs for RCPSP

One of the most prominent models for the RCPSP is based on a time discretization and was introduced in Pritsker, Watters, and Wolfe (1969). Their formulation adapted to our resource leveling problem reads as follows:

$$\min \sum_k c_k \cdot R_k \tag{1}$$

$$\text{s.t.} \quad \sum_t \sum_{m\in\mathcal{M}_j} x_{jmt} = 1 \quad \forall j \in \mathcal{J} \tag{2}$$

$$\sum_t \sum_{m\in\mathcal{M}_j} t \cdot x_{jmt} = S_j \quad \forall j \in \mathcal{J} \tag{3}$$

$$\sum_t \sum_{m\in\mathcal{M}_i} (t+p_{im}) x_{imt} \le \sum_t \sum_{m\in\mathcal{M}_j} t x_{jmt} \quad \forall (i,j) \in E \tag{4}$$

$$\sum_{j\in\mathcal{J}} \sum_{m\in\mathcal{M}_j} \sum_{\substack{\tau=t-p_{jm}+1 \\ \tau\ge 0}}^{t} r_{jmk} \cdot x_{jm\tau} \le R_k \quad \forall k,t \tag{5}$$

$$S_j \ge 0, \quad \forall j \in \mathcal{J} \quad R_k \ge 0, \quad \forall k \tag{6}$$

$$x_{jmt} \in \{0,1\} \quad \forall t,j,m \in \mathcal{M}_j \tag{7}$$

Binary ("start-time") variables x_{jmt} model whether job j starts at time t in mode m or not. Each job j must start exactly once, and in exactly one mode (2). The start time S_j of job $j \in \mathcal{J}$ is linked to the binary variables x_{jmt} in Equation (3) and is therefore implicitly integral. This linking is redundant, but we keep it for instructional purposes. Also precedence constraints (4), and resource capacity constraints (5) can be expressed as linear inequalities. In this model we decide about the resource capacities $R_k \ge 0$ for each resource k, such that the total resource availability cost is minimized via the objective function (1). For notational convenience we do not restrict the (j,m,t) indices of x_{jmt} variables to their feasible combinations. In fact, certain start times and certain modes would violate the calendar boundaries. However, this can be efficiently taken care of in a preprocessing step, yielding a huge reduction in the number of variables.

Depending on several factors, such as network complexity (the density of G) or the time discretization considered, this formulation may give very weak lower bounds. Very often, we experience that in an optimal solution to the LP relaxation only few binary variables are fractional, but the points in time used in the convex combination (3) to yield the actual start time S_j of a job are far apart from one another. A simple example is $x_{jmt_1} = x_{jmt_2} = 0.5$ which results in $S_j = (t_1+t_2)/2$. When t_1 and t_2 lie in different shifts it may happen that S_j is *outside* any shift, resulting in an infeasible start time. Fractions of jobs may violate precedence constraints even though the aggregated constraint (4) is fulfilled for the fractional solution. This holds similarly for the resource constraint (5). Things get more involved in our setting as different modes can be fractionally used for each job, thus all job durations between smallest and largest may appear. We informally call this a "smearing" of start time variables. This smearing gives us irrelevant information about the schedule and we lose most if not all structure in the model. This happens often in MIP models, however, it hits us particularly hard for scheduling problems.

Finally, branching a binary variable x_{jmt} to zero is a very weak decision if the corresponding job can be scheduled one time unit earlier or later. That is, the decision essentially has no effect. Not so the upward branch, which fixes a start time and mode and thus imposes significant structure to the overall solution. It is therefore to be expected that branching on x_{jmt} variables gives an unbalanced search tree, which is not desired.

3.2. Master problem: A model based on shift configurations

In order to reduce the effects of "losing the timing information" caused by the smearing of variables, we propose a model which exploits the problem structure by decomposing the time horizon into the availability periods of resources. Based on the calendar for each resource type, every working shift represents a smaller subproblem for which sub-schedules, or *configurations*, are stated for one resource

type only. These configurations are linked by global constraints ensuring that exactly one configuration is chosen for each working shift, and that precedence constraints are respected.

Definition 1. A *configuration* ξ for an interval $I \in \mathcal{I}_k$ of resource type k is a schedule for a subset $J \subseteq \mathcal{J}$ of jobs that all require resource type k. Each job $j \in J$ is specified by its start time $S_{j\xi}$ and its completion time $C_{j\xi}$ (this uniquely determines its mode). The maximum simultaneous usage of resource type k by these jobs is denoted by $\bar{r}_\xi$.

For each configuration ξ for a particular shift $I \in \mathcal{I}_k$ we introduce a binary variable x_ξ which indicates whether configuration ξ is chosen or not. We use the short hand notation $j \in \xi$ to express that job j is executed in the shift corresponding to configuration ξ. Note that the mode of each job is determined by the respective start and completion times. The so-called *master problem* reads:

$$\min \sum_k c_k \cdot R_k \tag{8}$$

$$\text{s.t.} \quad C_i \le S_j \quad \forall (i,j) \in E \tag{9}$$

$$S_j = \sum_{\xi : j \in \xi} S_{j\xi} \cdot x_\xi \quad \forall j \in \mathcal{J} \tag{10}$$

$$C_j = \sum_{\xi : j \in \xi} C_{j\xi} \cdot x_\xi \quad \forall j \in \mathcal{J} \tag{11}$$

$$\sum_{\xi : \xi \in I} \bar{r}_\xi \cdot x_\xi \le R_k \quad \forall k \quad \forall I \in \mathcal{I}_k \tag{12}$$

$$\sum_{\xi : j \in \xi} x_\xi = 1 \quad \forall j \in \mathcal{J} \tag{13}$$

$$R_k, S_j, C_j \ge 0 \quad \forall k, j \in \mathcal{J} \tag{14}$$

$$x_\xi \in \{0,1\} \quad \forall \xi \tag{15}$$

Each job is executed exactly once by Equation (13). This constraint also ensures, that for each job exactly one mode (from exactly one configuration) can be chosen. The start and completion times for each job are computed from the chosen configurations via the linking constraints (10) and (11). Constraints (9) model the precedence relations between jobs. These could be directly expressed by substituting S_j and C_j from the linking constraints, but (10) and (11) are helpful in the pricing problem (Section 3.3) where they help penalizing or encouraging certain start or completion times of jobs. Constraints (12) link resource consumptions to the capacities and ensure that the total resource capacity available over the planning horizon is at least the maximum resource usage needed in each shift.

3.3. Column generation: Pricing problem

Since the number of feasible configurations is exponential in the number of jobs, we solve the LP relaxation of (8)–(15) by column generation. That is, we start with a very small (e.g., heuristically generated) subset of configuration variables, and dynamically add more variables to the model until it can be proven that no more promising variables exist. This optimality proof is provided—as in the standard simplex method—via non-negativity of reduced costs of all configuration variables. It is checked with the help of an auxiliary optimization problem, the *pricing subproblem*, which determines a negative reduced cost configuration variable, or proves that none exist.

We must solve a pricing problem for every shift $I \in \mathcal{I}_k$ of each resource type k. For reduced cost computations, we need the dual variables of constraints (10)–(13) which are denoted by s_j, c_j, ρ_{kI}, and π_j, respectively. For a fixed shift I only the subset $J \subseteq \mathcal{J}$ of jobs needs to be considered that can be scheduled in I. This subset is restricted to jobs using the proper resource type, but also—as we see later—e.g., branching and other decisions may render certain start times of certain jobs infeasible. The objective function (16) reflects minimizing the reduced cost.

$$\max \quad \sum_j \pi_j \cdot X_j - \sum_j c_j \cdot C_j - \sum_j s_j \cdot S_j - \rho_{kI} \cdot \bar{r}_k \tag{16}$$

$$\text{s.t.} \quad \sum_{t \in I} \sum_{m \in \mathcal{M}_j} x_{jmt} = X_j \quad \forall j \in J \tag{17}$$

$$\sum_{t \in I} \sum_{m \in \mathcal{M}_j} t \cdot x_{jmt} = S_j \quad \forall j \in J \tag{18}$$

$$\sum_{t \in I} \sum_{m \in \mathcal{M}_j} (t + p_{jm}) \cdot x_{jmt} = C_j \quad \forall j \in J \tag{19}$$

$$\sum_{j \in J} \sum_{m \in \mathcal{M}_j} \sum_{\substack{\tau = t - p_{jm} + 1 \\ \tau \in I}}^{t} r_{jmk} \cdot x_{jm\tau} \le \bar{r}_k \quad \forall t \in I \tag{20}$$

$$\bar{r}_k \ge 0 \tag{21}$$

$$S_j,\ C_j \ge 0 \quad \forall j \in J \tag{22}$$

$$x_{jmt} \in \{0,1\} \quad \forall j \in J, \quad m \in \mathcal{M}_j, \quad t \in I \tag{23}$$

$$X_j \in \{0,1\} \quad \forall j \in J \tag{24}$$

The X_j, S_j, and C_j variables are "reporting" variables, and again present for instructional purposes only (as they could be substituted). Binary variable X_j decides whether job $j \in J$ is processed, and if so, at what start time S_t and completion time C_j. Note once more, that this uniquely determines the job's mode. All three variables for each job $j \in J$ characterize a configuration ξ for the specific shift $I \in \mathcal{I}_k$ of resource type k. This pricing problem is again a scheduling problem with a non-regular objective function. By its mechanism, this is again a time-indexed formulation, with binary start-time variables x_{jmt}, as is MIP (1)–(7). However, it has a much smaller time horizon (only one shift), and it is not required that each possible job $j \in J$ is actually executed. Constraints (17) model the selection of job j and its mode; Constraints (18) and (19) encode the start and completion times of jobs according to the chosen mode assignment. Resource capacity constraints (20) have to be satisfied as before and the resource usage in this shift is again a decision variable, denoted by $\bar{r}_k$. The objective function value is rewarded by π_j if job j is selected into the configuration and by multiples of s_j and c_j according to the start and completion times. A unit increase of resource usage penalizes the objective value by ρ_{kI}.

The pricing problem (16)–(24) is NP-hard as it contains a resource leveling problem as special case. This can be seen as follows: Set all π_j to a value large enough to ensure that each job must be scheduled, and let s_j and c_j be zero for all j. Then, we need to schedule all jobs during I at a minimum resource cost $\rho_{kI} \cdot \bar{r}_k$. As we are not aware of other exact approaches to this kind of hard scheduling problem, we solve the pricing problem as a mixed integer program itself.

The master problem (8)–(15) and the pricing problem (16)–(24) are solved alternately, exchanging dual variable values in one, and configuration variables in the other direction, until optimality of the master problem is proven. Note that an optimal solution to the master problem can be fractional, and integrality of configuration variables x_ξ still needs to be ensured. This leads us to a *branch-and-price* algorithm, in which the LP relaxation in each node of the branch-and-bound tree is solved by column generation as above.

3.4. Interpretation as non-trivial Dantzig–Wolfe Reformulation

In a Dantzig–Wolfe reformulation of a MIP $\min\{c^T x \mid Ax \ge b,\ Dx \ge d,\ x \in \mathbb{Q}^{n_1} \times \mathbb{Z}^{n_2}\}$, which is called the *original* formulation in this context, we change the polyhedral representation for a subset of constraints, say $Dx \ge d$. The convex hull of all the integer points (resp. mixed integer sets) that are feasible for $Dx \ge d$ is expressed

as convex combination $x = \sum_q \lambda_q x_q$, $\sum_q \lambda_q = 1$, $\lambda_q \geq 0$ of its finitely many extreme points x_q (assuming that $Dx \geq d$ is bounded). This convex combination is substituted in the remaining constraints $Ax \geq b$, yielding the *master problem*. The extreme points are given implicitly as solutions to an optimization problem, the *pricing problem*. The reformulation, which results in a best possible strengthening of constraints $Dx \geq d$, is applied in order to obtain stronger bounds from the relaxation. In many practical examples, $Dx \geq d$ can be separated into K *blocks* of disjoint sets of variables and constraints, $D^\kappa x^\kappa \geq d^\kappa$, $\kappa = 1, \ldots, K$, where the dimensions of vectors and matrices are compatible, and each block gives rise to a reformulation. We refer to Desrosiers and Lübbecke (2005, 2011) for details.

Considering the original time-indexed formulation (1)–(7), a Dantzig–Wolfe reformulation is not obvious. Note, however, that variables can be grouped "into blocks" according to shifts for all resource types. That is, variables x_{jmt} form a group for each fixed resource type k and shift $I \in \mathcal{I}_k$, containing all jobs j executable during I in all modes m. For each such group of variables, only those linking constraints (3) and resource constraints (5) for k and all $t \in I$ are relevant. These $\kappa = \sum_{k=1}^{|\mathcal{R}|} |\mathcal{I}_k|$ groups of variables and constraints form independent subproblems $D^\kappa x^\kappa \geq d^\kappa$ which are reformulated as sketched above. All other constraints (2) and (4) are global and kept in the role of master constraints $Ax \geq b$. Given k and I, the solution vectors ("extreme points") of the resulting pricing problem (16)–(24), consisting of $(X_j, S_j, C_j)_{j \in J}$ and $\bar{r}_k$, characterize a configuration ξ. This information, except the resource usage, entirely encodes a feasible setting of the original x_{jmt} variables; or in other words, the original variable vector can be expressed as selecting exactly one solution vector $(X_j, S_j, C_j)_{j \in J}$ and $\bar{r}_k$. Fractionally, this is a convex combination with multipliers x_ξ, and the convexity constraint is stated in (13).

It is not necessary to understand the model (8)–(15) as a Dantzig–Wolfe reformulation of (1)–(7), as our model has a self-supported interpretation. However, this interpretation motivates the branching rules we propose as well as the way we make use of cutting planes.

4. Branch-price-and-cut algorithm

4.1. Branching scheme

A solution to the original problem (1)–(7) is given by the resource capacities R_k, and an assignment of start times S_j and completion times C_j for each job j. The mode is given by the unique mode m_j, such that $p_{jm_j} = C_j - S_j$. In a modern branch-and-price context one tries to branch on these *original variables* which are expressible, via the Dantzig–Wolfe reformulation, as a sum of master variables, compare (10)–(12), instead of on single master variables (Desrosiers & Lübbecke, 2005). A main reason is that branching decisions on single master variables are hard to respect in the pricing problems, e.g., a configuration ξ, forbidden via $x_\xi = 0$, must not be re-generated. Branching decisions on (combinations of) master variables represent additional constraints that give rise to additional dual variables that need to be respected in the pricing problem. This usually implies new variables and new constraints in the pricing problem. However, when branching constraints are formulated on the original variables, these dual variables only affect a subproblem's objective function, not its constraints, see e.g., Desrosiers and Lübbecke (2011).

We impose a priority on branching decisions according to the effect we observed in preliminary experiments (Coughlan et al., 2010). The order is R_k, S_j, and then C_j, from highest to lowest priority. Only when all higher priority variables assume integral values, we branch on variables of the next class. Let a superscript star indicate the variable's value in the current master LP solution. Branching on resource capacity values has the largest impact as these have non-zero objective coefficients. We first evaluate the down-branch $R_k \leq \lfloor R_k^\star \rfloor$, and then the up-branch $R_k \geq \lfloor R_k^\star \rfloor + 1$. Branching on R_k changes the feasible space of the pricing problems, so we branch on these variables first even if $R_k^\star$ is not fractional. When there is choice between multiple resource variables, we select a most fractional one and use the objective function coefficient as a tie-breaker (prefer larger values). Start and completion time variables are considered as branching candidates only if any corresponding binary configuration variable in (10) is fractional. An unfixed start time variable S_j is selected where the points in time corresponding to positive x_ξ variables differ most, counteracting the smearing of variables. The node is split into two child nodes with $S_j \leq \lfloor S_j^\star \rfloor$, and $S_j \geq \lfloor S_j^\star \rfloor + 1$, respectively. Completion times are handled accordingly. This scheme is used in conjunction with propagation rules (described next) and aims at creating a more balanced search tree.

4.2. Propagation

It is a fundamental principle of constraint programming to tighten variable domains due to logical implications given by the constraints, and/or already fixed variables. For example, the precedence and resource constraints bear the logical structure of the problem and can be used to detect infeasible start times of variables which can therefore be eliminated from the domains. This is called *propagation*. In branch-and-price, domain propagation can also be used to establish consistency of the LP relaxation after branching, i.e., already generated master variables which represent configurations that contradict any previous branching decision taken on the path that led to the current node need to be locally fixed to zero (variables cannot simply be eliminated as they are essential in other parts of the search tree). We mentioned earlier that we only use x_{jmt} variables in a pricing problem that correspond to feasible combinations of (j, m, t) indices. These combinations are restricted after branching and propagation, and maybe after cutting. The preprocessing to eliminate infeasible combinations is also done in propagation.

In the area of scheduling problems, a large variety of propagation algorithms is known to detect infeasible start times and perform variable bound adjustments. *Edge-finding* is a constraint programming technique concerned with deriving better bounds for earliest start and latest completion times of jobs using energy arguments. We adapt the algorithm proposed in Mercier and Van Hentenryck (2008) to the multi-mode case by using the minimum energy used by any mode for each job, which naturally seems to give weaker bounds. This is balanced by the fact that jobs are not preemptive, may not cross shift-bounds and obey precedence constraints which enables further propagation of start and completion times.

We use this algorithm in every node of the branch-and-price tree, prior to calling the pricing problem and cutting plane separation.

4.3. Cutting planes

State-of-the-art MIP solvers heavily rely on additional valid inequalities ("cutting planes") in order to improve the dual bound and by that prune unpromising nodes of the branch-and-bound tree. In branch-and-price, formulating cutting planes in the master problem variables is possible, but raises again (as with branching constraints) the issue of how to respect the additional dual variables in the pricing problem. It is technically easier to formulate valid inequalities on the *original* variables and add their Dantzig–Wolfe reformulation, i.e., in our case their translation to configuration variables, to the master problem. Again, this only changes the objective function of the pricing problem, see again Desrosiers and Lübbecke (2011) for details. Incidentally, this is a good situation for us as the literature knows several cutting planes for various scheduling problems, all of them formulated on variables with a meaning as in the standard MIP (1)–(7). Here, we generalize the precedence inequalities as introduced in Christofides, Alvarez-Valdes, and Tamarit (1987) where jobs have

a fixed processing time:

$$\forall (i,j) \in E, \tau: \quad \sum_{t \geq \tau} x_{it} + \sum_{t < \tau + p_i} x_{jt} \leq 1. \tag{25}$$

These cuts read as follows: For any point in time τ, in an end-to-start precedence relation between jobs i and j, either job i starts the latest in τ or job j starts before $\tau + p_i$. As multiple modes are present we compute the minimum processing time $p_{i,\min}$ a job i may have locally, i.e., the minimum over all of its modes. Building on these cuts, we obtain the following cuts in our master problem:

$$\forall (i,j) \in E, \tau: \quad \sum_{\xi : S_{i\xi} \geq \tau} x_\xi + \sum_{\xi : S_{j\xi} \leq \tau + p_{i,\min}} x_\xi \leq 1. \tag{26}$$

By construction, this does not entail any structural changes to the pricing problem, but only the coefficients of the objective function need to be updated. For each constraint (26) that is added to the relaxation, we introduce the dual variable $\mu_{\tau ij}$ which integrates in the reduced cost objective function of the pricing problem. If $S_i \geq \tau$ or $S_j < \tau + p_{i,\min}$, the cost coefficient of variable x_{imt} becomes

$$\sum_{(i,j) \in E} \sum_{\tau \geq t} \mu_{\tau ij} + \sum_{(k,i) \in E} \sum_{\tau < t + p_{im}} \mu_{\tau ki}. \tag{27}$$

The cutting planes are added to the master problem after new configuration variables have been generated and each configuration variable is introduced into the corresponding constraints. Overall, this gives a branch-price-and-cut algorithm. In Appendix A, we present these cuts in the context of generalized precedence constraints.

4.4. Primal heuristics

For the master problem, rounding heuristics for fractional solutions are not promising, since values of binary variables may be smeared over the time horizon and precedence constraints are likely to be violated in a rounded solution. To improve our upper bounds we extend a leveling heuristic from Megow et al. (2011) that produces initial solutions prior to search. We implement a generic list scheduling algorithm (a serial schedule generation scheme) and a ready scheduling heuristic (a parallel schedule generation scheme, see Coughlan et al., 2010) which are used as standalone heuristics during the branching process as well as in the initial leveling procedure.

The two heuristics are called in each node of the search tree. To this end, we set the capacity of each resource to the fractional variable value rounded up, and fix the earliest and latest start and completion times for each job to the local bounds of the corresponding variables. We perform list scheduling using the (fractional) LP solution with jobs sorted by earliest completion times, where each job's mode is chosen according to the largest processing time of at most $C_j - S_j$. If no feasible solution is obtained we try ready scheduling. Both of these heuristics produce solutions that are not necessarily feasible w.r.t. the current primal bound, since resource capacities are rounded up. Regardless of this, if a feasible schedule is found new columns representing that schedule are added to the master problem, in order to reduce the total number of pricing iterations.

5. Computational study

In this section we compare the outcome of our branch-price-and-cut algorithm against a preprocessed time-indexed formulation (1)–(7) solved by the MIP solver CPLEX. In particular, we are interested in the dual bounds that are obtained in the root node by either algorithm since these bounds indicate the strength of the polyhedral description. Furthermore, we give experimental evidence that the branching strategy and on some instances a careful use of precedence inequalities play an important role to efficiently solve a large number of the benchmark problems.

5.1. Benchmark instances

There is no publicly available test set of benchmark instances reflecting the setup of our problem. The existing instances in the PSPLib (Kolisch & Sprecher, 1996) guided the design of our test set compilation. We use the same generation scheme as before in Coughlan et al. (2010), now for larger instances. Our set is composed of two sets of job scenarios, with 50 instances each. Each job can run in three different modes, using one to three units of its resource, with durations ranging from 5 to 12. The first set, denoted by N50E70, contains 50 jobs and 70 precedence constraints, whereas the second set N50E100 contains 50 jobs and 100 precedence constraints. The maximal width W of the precedence graph is six, which is achieved by constructing W chains of length $|\mathcal{J}|/W$, and randomly choosing the remaining edges.

There are two different resources which run in five calendar configurations, called C1–C5. These calendars are described schematically in Fig. 2. In the top row, calendars C1–C3 are shown. In each of these, the length of the shifts is 60. In C1 and C3 shift breaks are 60 units long, in C2 only 20 units long. Both resources are available at the same time in C1, while in C3 availability periods are complementary. In C2 the second resource is offset at 40 units. Calendars C4 and C5 show shifts with length 20 and breaks having length five. In C5 one of the resources is offset by 10. All scenarios are tested with each of the five different calendars. Time horizons were chosen by computing a minimal and maximal makespan heuristically using an earliest start list scheduling policy, and averaging these. The two makespans are computed by either scheduling all jobs at their highest resource usage (fastest mode) or by assigning the fewest possible resource usage to each job (slowest mode).

5.2. Experimental setup

All experiments were done on Intel Core™ i7-870 PCs (2.93 gigahertz, 8 megabytes cache, 8gigabytes memory) running Linux 2.6.34 (single thread). Each test run had a time limit of 1800 seconds. Our C++ implementation is based on SCIP 2.0.1 (SCIP, 2011) to perform the branch-price-and-cut process, with custom plug-ins for our heuristics, propagation, branching rules, cutting plane separation, and column generation. For the standard MIP (1)–(7) we used CPLEX 12.2 on the same machine, with default parameter settings, again single thread. Up to two threads were run in parallel on not entirely idle machines, so run time differences of 5 percent are regarded as "noise."

Fig. 2. Calendar configurations C1–C3 (top) and C4 and C5 (bottom) used in our test set. Black bars symbolize the temporal location of shifts.

5.2.1. Usefulness of the heuristics

Previous results showed that it is important for CPLEX and our branch-price-and-cut framework to have a good initial solution, whereas the execution time of the heuristic throughout search is negligible, see Coughlan et al. (2010).

5.2.2. Separation of cuts

There are $m \cdot T$, i.e., a pseudo-polynomial number of precedence cuts (26); these are too many to explicitly add them all to the master problem, as this would have a negative impact on LP solving times. For the identification of violated cutting planes, which is called *separation*, we pursue two approaches. In our first approach we only check for a precedence pair (i, j) at point in time $\tau_{\text{greedy}} = \lceil (C_i + S_i)/2 \rceil$ whether Equation (26) holds or is violated by more than a threshold ϵ. In the second approach, we find an optimum point τ_{best} such that the left-hand side of Equation (26) is maximally violated. This can be done by sorting the summands of Equation (26) for each point in time τ such that for each value τ the left-hand side can be computed in linear time (after sorting). This procedure is pseudo-polynomial in the number of points in time.

5.2.3. Settings

We compare the outcome of a CPLEX run on the standard MIP model (1)–(7) to different settings of our tailored branch-price-and-cut approach. In a first experiment, we do not separate cuts, denoted by "nocuts." Then, we evaluate different separation strategies. First, we separate cuts already in the root node, and secondly, after resource capacity variables are fixed (by branching or propagation). Checking for violation only at a promising guess as point in time is denoted by "τ_{greedy}," searching for the largest violation cuts is denoted by "τ_{best}." Furthermore, these cuts are only separated if a certain threshold "tol." is exceeded, i.e., if in Equations (26) the sum of the variables exceeds $1 + \text{tol}$. We use thresholds 10^{-4}, 0.1, 0.3, 0.5 and 0.8, which range from a very small violation (gives more cuts) to a very high violation (gives less cuts).

5.3. Results

We confirm that a Dantzig–Wolfe reformulation can considerably improve the dual bound as compared to the standard LP relaxation. As an even stronger statement, we also obtain a dual bound improvement over a state-of-the-art branch-and-cut algorithm. In Fig. 3 we see that the average improvement per calendar of our configuration based formulation (8)–(15) over the LP relaxation of (1)–(7) lies between 20 percent and 65 percent throughout all instances. The displayed average deviations and the minimum and maximum values of that improvement show the strength of our relaxation.

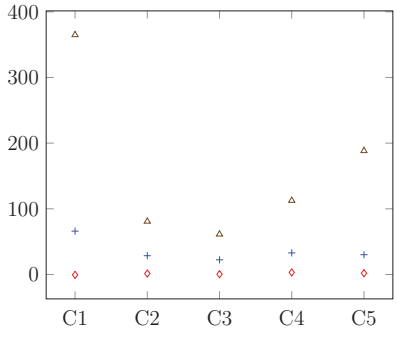


Fig. 3. Percentage of improvement of the lower bound obtained from our configuration based formulation (8)–(15) (no additional cutting planes) over the standard LP relaxation of (1)–(7) as computed by the state-of-the-art MIP solver CPLEX. The minimum (diamond), maximum (triangle), mean, and standard deviation of the improvement in percent of setting "no cuts" is shown.

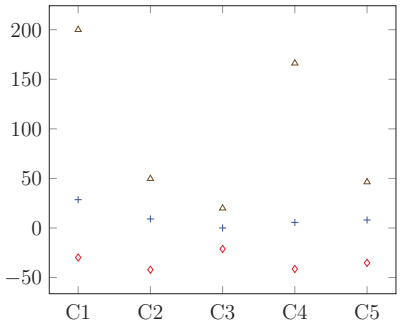


Fig. 4. Average mean and standard deviation of root dual bound improvement of our branch-price-and-cut approach as compared to CPLEX in percent. The best bound over all tolerances has been used. Triangles show the best obtained improvement (e.g., 200 percent in C1) and diamonds show the worst lower bound (e.g., 44 percent worse than CPLEX in C2).

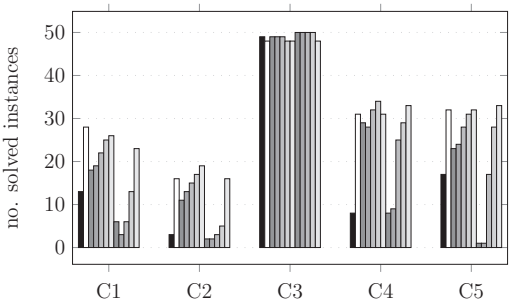


Fig. 5. Number of solved instances for N50E70 with cuts already separated in the root node; each set of bars from left to right corresponds to rows in Table 1. A CPLEX run on the standard MIP is black; our branch-and-price algorithm without cuts is white; and the two groups of different shades of grey show runs of the full branch-price-and-cut algorithm with precedence cuts enabled, with settings τ_{greedy} and τ_{best}, respectively, with increasing tolerances (lighter grey is larger tolerance).

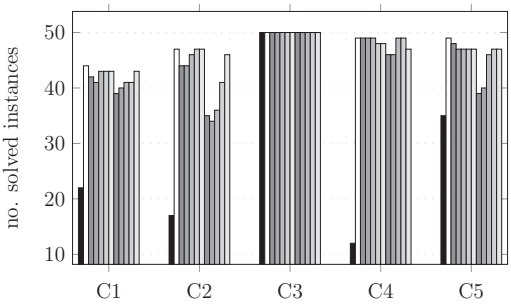


Fig. 6. Number of solved instances for N50E100 with cuts already separated in the root node. See Fig. 5 for the meaning of the bars.

To evaluate the strength of generic cutting planes added in the root node of the commercial MIP solver, we compare the root dual bounds in Fig. 4 after all standard cutting planes have been added by CPLEX versus the root node dual bound in our approach. Our improvement is no longer that dominating as in Fig. 3 but still for several of the hard instances in calendar setting C1, there is a 28 percent improvement on the average of the root dual bound compared to CPLEX.

Next, we compare our branch-price-and-cut framework to a standard MIP approach in terms of absolute number of solved instances and afterward, we discuss the effect of precedence cuts in our model by evaluating the number of branch-and-bound nodes and the running time. In Fig. 5 and 6 for each calendar, the first bar (black) gives the number of solved instances obtained by CPLEX, the second (white) bar represents this number for our branch-and-price approach without precedence cuts. The first five grey bars symbolize the results for setting "τ_{greedy}" in increasing order of tolerances and the last five bars stand for settings "τ_{best}" in increasing order of tolerances.

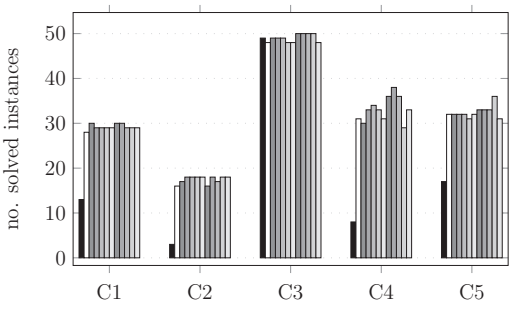


Fig. 7. Number of solved instances of test set N50E70 if cuts are separated after resource capacity variables are fixed. See Fig. 5 for the meaning of the bars.

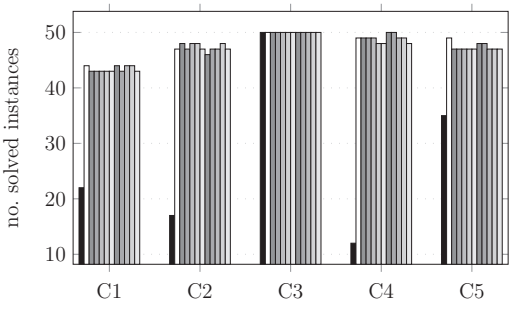


Fig. 8. Number of solved instances of test set N50E100 if cuts are separated after resource capacity variables are fixed. See Fig. 5 for the meaning of the bars.

Fig. 5 shows for the instance set N50E70 that the pure branch-and-price algorithm without precedence cuts outperforms CPLEX, and adding precedence cuts seems to be a bad idea. Comparing Fig. 5 and 6, the set N50E70 with fewer precedence constraints than N50E100 is the harder one, as expected. In some cases the pure MIP approach is even better than the branch-price-and-cut approach with additional precedence cuts. The reason is that the time spent for separating new cuts, pricing new variables, and the additional LP iterations lead to too many timeouts and is therefore not competitive. Hence, not using any precedence cuts at all is, at first sight, a good decision.

Nevertheless, it is possible to increase the root dual bound on several instances by using precedence cuts (26). During root solving most jobs are able to slide in their time window and the precedence cuts result in more binary configuration variables that are smeared over the time horizon. Hence, these cuts should not be used in the root node but still might be beneficial to prune certain nodes of the branch-and-bound tree. Recall that our branching scheme first branches on the resource capacities R_k and afterwardon the start and completion time variables. Fixed resource capacities in a node already decide on a lot of structure for the scheduling problem, since several modes of a job may no longer be valid. Thus, this seems to be a good point to separate precedence cuts (26). Figs. 7 and 8 show that more instances than before can be solved using the precedence cuts. In several cases a tolerance of 10^{-4} belongs to the best choices for the separation procedure. Especially, for the harder instances N50E70 some more instances of each calendar test set can be solved in the time limit to proven optimality in contrast to CPLEX or a setting without additional precedence cuts.

Next, we elaborate on the solution time and on the number of nodes needed to find an optimal solution and prove its optimality. The results on the total number of solved instances showed that it is beneficial to separate precedence cuts after resource variables are fixed. Therefore, we will only present the numbers for that case.

Tables 1 and 2 reveal that using additional precedence cuts enables us to decrease the number of tree nodes by 10 percent to 20 percent on average. For several hard instances, e.g., in calendar C5 in test set N50E70 a decrease by even 50 percent is possible. Best results in terms of nodes are obtained when τ_{best} is used for the separation of cuts. Nevertheless, this does not carry over to a reduced running time. For C1 the running times increase (τ_{best} vs. τ_{greedy}), whereas for C5 it decreases and the running time is about 10 percent faster than if no precedence cuts are separated. That is, there is the usual trade-off between quality and time, and cutting planes may be most interesting in memory critical applications.

A tolerance between 0.1 and 0.3 gives the overall best results as higher and lower tolerances usually increase the running times.

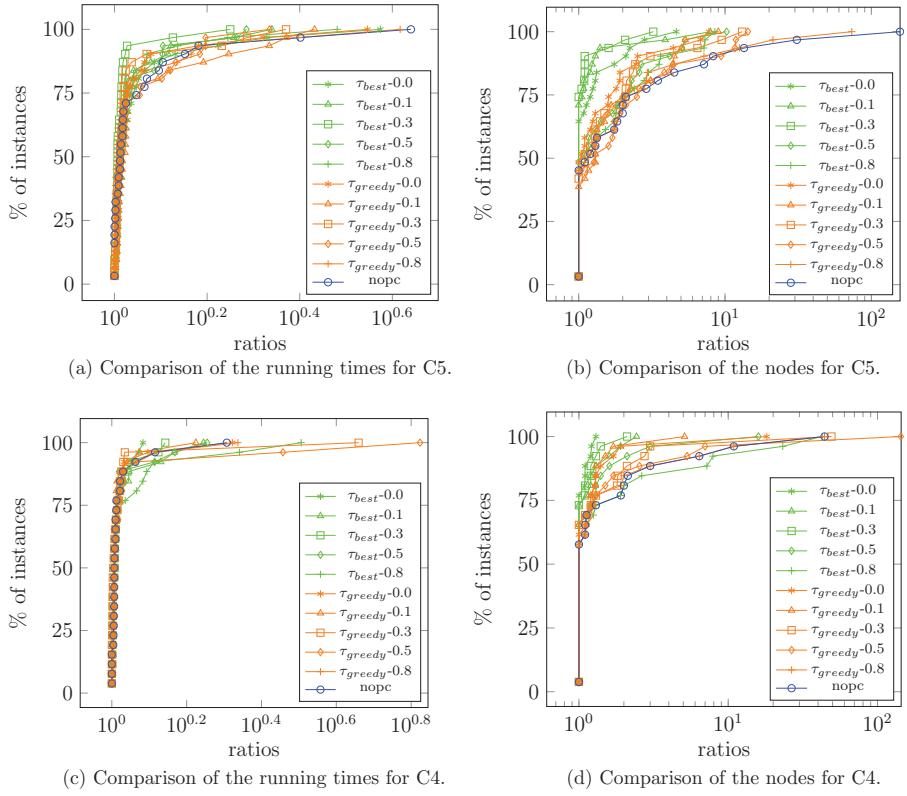
Tables 1 and 2 only show a slight improvement considering the average running time and the number of nodes needed (in the shifted geometric mean) if precedence inequalities are separated after resource capacities are fixed. The performance profiles (Dolan & Moré, 2002) in Fig. 9 give a more detailed comparison by comparing the ratios of the number of nodes (running time) needed per instance versus the best running time by any of the settings listed in the tables. We see that, e.g., for calendar C5 on more than 50 percent of the instances the precedence inequalities remarkably reduce the number of nodes needed. On several instances, the setting without precedence inequalities needs between 10 and 100 times more nodes than the best setting with precedence inequalities. But we also see that the reduction in terms of running times is much smaller as separating these cuts and computing the new objective coefficients in the pricing problems is costly, too. Similarly, for calendar C4 we observe a decrease in the number of nodes needed, whereas the reduction in terms of running time is rather modest and can only be seen on less than 20 instances. We do not elaborate on the results for calendars C1–C3 here. On calendars C1 and C2, results are similar to C5, whereas on C3 not much changes, as on these instances the dual bounds have not been a bottleneck.

Table 1
Comparison of tree nodes and running time in seconds for N50E70—Cuts are only separated after resources are fixed. Means are computed over the instances solved by all settings, of which there are 27, 14, 48, 26, and 31 instances for calendars C1–C5.

		C1		C2		C3		C4		C5	
	Tol.	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time
No cuts		12.0	660.7	10.7	546.3	1.0	6.4	11.4	200.5	21.3	356.2
τ_{greedy}	0.0	10.0	667.9	9.5	553.4	1.0	6.4	8.9	201.0	14.3	346.3
	0.1	10.2	662.7	8.8	518.7	1.0	6.4	8.3	197.2	16.7	361.8
	0.3	10.6	632.6	9.0	523.4	1.0	6.4	10.0	204.2	15.9	338.9
	0.5	11.4	661.0	10.5	538.9	1.0	6.4	11.2	216.9	19.1	347.2
	0.8	11.4	611.5	10.7	560.7	1.0	6.3	11.1	200.4	19.7	351.5
τ_{best}	0.0	9.2	704.6	8.3	536.0	1.0	6.4	7.4	196.4	12.0	353.6
	0.1	8.9	673.1	8.4	559.8	1.0	6.4	7.9	202.8	11.6	341.3
	0.3	9.2	653.6	8.6	542.7	1.0	6.4	7.7	197.6	10.9	323.3
	0.5	10.1	660.7	9.8	560.6	1.0	6.4	9.1	201.1	15.6	337.6
	0.8	11.1	608.4	10.3	528.2	1.0	6.3	12.4	212.4	16.6	338.9

Table 2
Comparison of tree nodes and running time in seconds for N50E100—Cuts are only separated after resources are fixed. Means are computed over the instances solved by all settings, of which there are 42, 44, 50, 48, and 47 instances for calendars C1–C5 respectively.

		C1		C2		C3		C4		C5	
	Tol.	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time
No cuts		5.3	31.0	9.8	88.8	1.1	1.0	8.4	18.1	8.2	23.8
τ_{greedy}	0.0	5.4	32.6	8.0	83.6	1.1	1.0	6.3	16.4	7.7	25.5
	0.1	4.8	31.1	8.2	86.9	1.1	1.0	6.4	16.5	7.3	24.3
	0.3	5.4	30.5	9.1	87.3	1.1	1.0	6.5	16.3	7.7	24.1
	0.5	5.1	30.7	9.1	85.5	1.1	1.0	6.8	16.5	8.2	24.0
	0.8	5.4	29.7	10.3	88.1	1.1	0.9	7.4	16.4	9.0	23.6
τ_{best}	0.0	4.8	32.3	7.5	88.8	1.1	1.0	5.9	16.5	6.3	23.5
	0.1	4.8	32.5	7.2	86.8	1.1	1.0	5.7	16.4	6.2	23.0
	0.3	4.7	30.5	7.6	87.3	1.1	1.0	5.8	16.3	6.5	22.9
	0.5	5.1	31.4	7.7	83.2	1.1	1.0	6.4	16.5	7.8	24.0
	0.8	5.4	29.8	10.0	92.2	1.1	0.9	7.2	16.2	8.5	23.3



(a) Comparison of the running times for C5.

(b) Comparison of the nodes for C5.

(c) Comparison of the running times for C4.

(d) Comparison of the nodes for C4.

Fig. 9. Performance profiles: Ratios of nodes and running times for all optimally solved instances from N50E70 for calendar C5 on top and for calendar C4 below.

Table 3 indicates the computational efforts spent in the pricing problems. For each set, N50E70 and N50E100, and per calendar C1–C5, we list the average running time (avtime) and number of nodes (avnodes) per instance. Column "ptime/time" shows that solving the pricing problem (setting up, solving the MIP, and generating new columns) takes from 58 percent up to 99 percent of the total solving time. Hence, the pricing routine determines the overall execution time. Recall that in each pricing round, a MIP for each shift needs to be solved. Column "# calls per node" shows how often the pricer is called on average per node, while the number of variables generated per call is indicated in the last column. The number of shifts per instance (which is equal to the number of MIPs to be solved with each call of the pricing problem) and the number of pricing rounds when solving a node influence this number.

Table 4 shows for each set and per calendar how often per instance that was not solved to optimality ("nsubopt") CPLEX hit the best known primal ("nbestprimal") and dual ("nbestdual") bounds. The succeeding columns show the average gap of the best primal (dual) solution versus the best known solution (dual bound) from all our branch-price-and-cut variants, denoted by "avpgap" ("avdgap"). The last column presents the average gap that CPLEX was able to arrive at. From that table we can see that for set N50E70, the primal gap is the hardest for CPLEX, 13 to 23 percent. In contrast, for set N50E100 the average dual gap is much bigger than the average primal gap. Here, the average dual gap compared to all settings using branch-price-and-cut ranges from 13 percent to 37 percent, while the primal gap falls between 13 and 17 percent.

6. Summary and conclusions

We studied a practically motivated, complex multi-mode project scheduling problem with the objective to minimize the resource availability cost. Our aim was to demonstrate how far one can get when insisting on optimal solutions. The literature on related problems and in

Table 3
Share of time spent in the pricing problem (ptime/time), number of calls per node and number of variables generated per call on average over all instances per set. A call to the pricer triggers the pricing for each shift.

Set	Calendar	avtime	avnodes	ptime/time	# calls/node	# vars/call
N50E70	C1	1230.69	14.46	0.99	8.67	18.44
	C2	1424.49	40.77	0.97	10.08	10.72
	C3	51.72	29.76	0.97	5.46	9.01
	C4	843.63	313.36	0.88	5.71	6.63
	C5	922.76	255.53	0.92	4.30	7.76
N50E100	C1	312.61	276.70	0.76	3.76	5.73
	C2	273.46	76.16	0.93	6.65	8.35
	C3	3.06	1.13	0.99	6.68	22.84
	C4	74.46	491.68	0.64	2.33	7.13
	C5	152.70	519.20	0.56	2.70	3.86

Table 4
Evaluation of primal and dual bounds for all instances that could not be solved to optimality by CPLEX.

Set	Calendar	nsubopt	nbestprimal	nbestdual	avpgap	avdgap	avcplexgap
N50E70	C1	37	14	17	0.18	0.12	0.34
	C2	47	23	28	0.13	0.09	0.31
	C3	1	0	0	0.16	0.50	0.56
	C4	42	13	16	0.23	0.17	0.34
	C5	33	8	19	0.22	0.06	0.27
N50E100	C1	28	14	3	0.13	0.37	0.47
	C2	33	14	6	0.17	0.23	0.34
	C3	0	0	0	–	–	–
	C4	38	25	2	0.10	0.36	0.42
	C5	15	6	6	0.16	0.13	0.24

particular the PSPLib with up-to-date benchmark results suggest that even today instances with only 30 jobs are hard to solve to optimality when a "standard" model is used; in our case we compare against an adapted classical MIP formulation (Pritsker et al., 1969). Our study demonstrates that it helps to decompose the problem resource-wise per availability period. Technically, this means applying a Dantzig–Wolfe reformulation to the standard formulation (in a non-obvious way), even though this is not necessary to understand our approach. The resulting model is based on variables that represent entire schedules for each resource and shift, of which there are exponentially many. This necessitates a solution approach via a column generation and branch-and-price algorithm. A usually complicating feature in project scheduling, the resource availability periods (or shift calendars), can even be turned into an advantage in this context, as this reduces the size of the variable-generating subproblems in column generation.

In our computational study we showed that the LP relaxations of the reformulated model are much stronger than that of the standard approach. The dual bound can even be strengthened by applying additional valid inequalities which are known for the standard formulation, and which we translate to our model. We believe that this is prototypical, and that more inequalities known from the scheduling literature could be integrated in our approach. The resulting branch-price-and-cut algorithm enables us to solve a large amount of instances with 50 jobs to proven optimality, where state-of-the-art MIP solvers suffer from the size and weak relaxation of the standard model.

Our algorithm is generic and components like the pricing problem may also be solved via constraint programming algorithms or partially with a heuristic. We believe that the general approach is well-suited for similar problems, in particular, when the objective function is "complicated." We finally remark that we believe that our ability to provide solution algorithms to complex models as ours changes our way we model with integer variables. While classical IP modeling used variables with "little meaning" our study supports the advise to impose more structure on a single variable's meaning.

Appendix A. Cuts for generalized precedence inequalities

We generalize the precedence inequalities as introduced in Christofides et al. (1987) where jobs have a fixed processing time to our master problem also for the case of generalized precedence relations. This is interesting for modeling multi-resource jobs, among others. The cuts of Christofides et al. are given as follows:

$$\forall (i,j) \in E, \tau: \quad \sum_{t \geq \tau} x_{it} + \sum_{t < \tau + p_i} x_{jt} \leq 1. \tag{A.1}$$

These cuts read as follows: For any point in time τ, in an end-to-start precedence relation between jobs i and j, either job i starts the latest in τ or job j starts before $\tau + p_i$. Building on these cuts, we study generalized precedence constraints that are of the form start-to-start (GP_1), end-to-start (GP_2), start-to-end (GP_3) and end-to-end (GP_4). Let $\delta_{ij} \in \mathbb{Z}$, then we can include these generalized precedence relations in our master problem by adding the following inequalities based on start and completion time variables (S_j, C_j), and also separate strengthened inequalities, similar to (A.1), based on the configuration variables (x_ξ).

$(i,j) \in GP_1 \Leftrightarrow S_i + \delta_{ij} \leq S_j$ or:

$$\sum_{\xi: S_{i\xi} \geq \tau} x_\xi + \sum_{\xi: S_{j\xi} < \tau + \delta_{ij}} x_\xi \leq 1 \quad \forall\, \tau$$

$(i,j) \in GP_2 \Leftrightarrow C_i + \delta_{ij} \leq S_j$ or:

$$\sum_{\xi: C_{i\xi} \geq \tau} x_\xi + \sum_{\xi: S_{j\xi} < \tau + \delta_{ij}} x_\xi \leq 1 \quad \forall\, \tau$$

$(i,j) \in GP_3 \Leftrightarrow S_i + \delta_{ij} \leq C_j$ or:

$$\sum_{\xi: S_{i\xi} \geq \tau} x_\xi + \sum_{\xi: C_{j\xi} < \tau + \delta_{ij}} x_\xi \leq 1 \quad \forall\, \tau$$

$(i,j) \in GP_4 \Leftrightarrow C_i + \delta_{ij} \leq C_j$ or:

$$\sum_{\xi: C_{i\xi} \geq \tau} x_\xi + \sum_{\xi: C_{j\xi} < \tau + \delta_{ij}} x_\xi \leq 1 \quad \forall\, \tau$$

Using these cuts in our master problem, the nature of the pricing problem (i.e., its constraints) does not change, only the coefficients of the objective function need to be updated. For each cut encoded by $(\tau, (i,j)) \in \mathrm{GP}_\ell$ we need to add the value of the dual variable $\mu^\ell_{\tau ij}$ to the objective function, for all types of generalized precedence relations GP_ℓ, $\ell = 1, \ldots, 4$ according to the following rules. The cost coefficient of variable x_{imt} in the pricing problem is increased by:

GP_1:

$$\sum_{(i,j)\in \mathrm{GP}_1} \sum_{\tau : t \geq \tau} \mu^1_{\tau ij} + \sum_{(k,i)\in \mathrm{GP}_1} \sum_{\tau : t < \tau + \delta_{ki}} \mu^1_{\tau ki}$$

GP_2:

$$\sum_{(i,j)\in \mathrm{GP}_2} \sum_{\tau : t + p_{im} \geq \tau} \mu^2_{\tau ij} + \sum_{(k,i)\in \mathrm{GP}_2} \sum_{\tau : t < \tau + \delta_{ki}} \mu^2_{\tau ki}$$

GP_3:

$$\sum_{(i,j)\in \mathrm{GP}_3} \sum_{\tau : t \geq \tau} \mu^3_{\tau ij} + \sum_{(k,i)\in \mathrm{GP}_3} \sum_{\tau : t + p_{im} < \tau + \delta_{ki}} \mu^3_{\tau ki}$$

GP_4:

$$\sum_{(i,j)\in \mathrm{GP}_4} \sum_{\tau : t + p_{im} \geq \tau} \mu^4_{\tau ij} + \sum_{(k,i)\in \mathrm{GP}_4} \sum_{\tau : t + p_{im} < \tau + \delta_{ki}} \mu^4_{\tau ki}$$

References

Achterberg, T. (2009). SCIP: Solving constraint integer programs. *Mathematical Programming Computation, 1*(1), 1–41.

Bianco, L., & Caramia, M. (2011). A new lower bound for the resource-constrained project scheduling problem with generalized precedence relations. *Computers and Operations Research, 38*(1), 14–20.

Brucker, P., & Knust, S. (2000). A linear programming and constraint propagation-based lower bound for the RCPSP. *European Journal of Operational Research, 127*(2), 355–362.

Christofides, N., Alvarez-Valdes, R., & Tamarit, J. M. (1987). Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research, 29*(3), 262–273.

Coughlan, E. T., Lübbecke, M. E., & Schulz, J. (2010). A branch-and-price algorithm for multi-mode resource leveling. In P. Festa (Ed.), *Experimental algorithms*. In *Lecture Notes in Computer Science: 6049* (pp. 226–238). Lecture Notes in Computer Science, vol. 6049. Springer.

Demeulemeester, E. (1995). Minimizing resource availability costs in time-limited project networks. *Management Science, 41*, 1590–1598.

Demeulemeester, E. L., & Herroelen, W. S. (2002). *Project scheduling: A research handbook*. Kluwer.

Desrosiers, J., & Lübbecke, M. E. (2005). A primer in column generation. In G. Desaulniers, J. Desrosiers, & M. M. Solomon (Eds.), *Column generation* (pp. 1–32). Berlin: Springer-Verlag.

Desrosiers, J., & Lübbecke, M. E. (2011). Branch-price-and-cut algorithms. In J. J. Cochran (Ed.), *Encyclopedia of operations research and management science*. Chichester: John Wiley & Sons. http://onlinelibrary.wiley.com/doi/10.1002/9780470400531.eorms0118/abstract.

Dolan, E. D., & Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming, 91*, 201–213.

Drexl, A., & Kimms, A. (2001). Optimization guided lower and upper bounds for the resource investment problem. *The Journal of the Operational Research Society, 52*(3), 340–351.

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

Hartmann, S. (2001). Project scheduling with multiple modes: A genetic algorithm. *Annals of Operations Research, 102*(1–4), 111–135.

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research, 207*(1), 1–14.

Kolisch, R., & Sprecher, A. (1996). PSPLIB—A project scheduling problem library. *European Journal of Operational Research, 96*, 205–216. http://129.187.106.231/psplib/, last accessed 2011/05/26

Megow, N., Möhring, R. H., & Schulz, J. (2011). Decision support and optimization in shutdown and turnaround scheduling. *INFORMS Journal of Computing, 23*(2), 189–204.

Mercier, L., & Van Hentenryck, P. (2008). Edge finding for cumulative scheduling. *INFORMS Journal of Computing, 20*(1), 143–153.

Möhring, R. H. (1984). Minimizing costs of resource requirements in project networks subject to a fixed completion time. *Operations Research, 32*(1), 89–120.

Neumann, K., Schwindt, C., & Zimmermann, J. (2003). *Project scheduling with time windows and scarce resources*. Springer.

Pritsker, A. A. B., Watters, L. J., & Wolfe, P. M. (1969). Multi project scheduling with limited resources: A zero-one programming approach. *Management Science, 16*, 93–108.

Santos, M. A., & Tereso, A. P. (2011). On the multi-mode, multi-skill resource constrained project scheduling problem a software application. In A. Gaspar-Cunha, R. Takahashi, G. Schaefer, & L. Costa (Eds.), *Advances in Intelligent and Soft Computing: 96. Soft computing in industrial applications* (pp. 239–248). Berlin/Heidelberg: Springer. doi:10.1007/978-3-642-20505-7_21.

SCIP (2011). Solving constraint integer programs. http://scip.zib.de/. Version 2.0.1.

Zhan, J. (1992). Calendarization of time planning in MPM networks. *ZOR – Methods and Models for Operations Research, 36*(5), 423–438.