

A Branch-Price-and-Cut Algorithm for Multi-Mode Resource Leveling*

Eamonn T. Coughlan · Marco E. Lübbecke · Jens Schulz

the date of receipt and acceptance should be inserted later

Abstract Resource leveling is a variant of resource-constrained project scheduling in which a non-regular objective function, the resource availability cost, is to be minimized. We present an exact branch-price-and-cut approach together with a new heuristic to optimally or near-optimally solve the more general turnaround scheduling problem. In addition to precedence and resource constraints, also resource availability periods and multiple modes per job have to be taken into account. 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. In particular for larger instances, our approach gives tighter bounds, enabling us to optimally solve instances with 50 multi-mode jobs.

1 Introduction

Motivated by an industrial application from chemical engineering, we study a resource leveling problem, which was recently introduced as *turnaround scheduling problem* by Megow et al (2011). In turnaround scheduling, for the inspection and renewal of parts, plants are shut down, disassembled, and rebuilt, so there is a partial ordering of jobs to be done. Jobs are *multi-mode*, that is, they can be sped-up by investing in more workers. The time horizon and the number of workers hired for each job determine production downtime and working cost, the two of which are conflicting in a time-cost tradeoff manner. Once a time horizon is fixed, the problem turns into a resource leveling problem, on which we focus in this paper.

We have workers of different specialization, resp. different renewable resources in general, each associated with *availability periods* that can be thought of as working shifts. The granularity of planning is so fine that each job needs (possibly several units of) exactly one

* A preliminary version appeared in Coughlan et al (2010)

Eamonn T. Coughlan · Jens Schulz
Technische Universität Berlin, Institut für Mathematik, MA 5–1, Straße d. 17. Juni 136, 10623 Berlin, Germany, E-mail: {coughlan,jschulz}@math.tu-berlin.de

Marco E. Lübbecke
RWTH Aachen University, Operations Research, Templergraben 64, 52056 Aachen, Germany, E-mail: marco.luebbecke@rwth-aachen.de

resource. This one-resource-per-job policy is because of the input format of our industrial partners who use MS Project™ software. It is not restrictive, since generalized precedence constraints can model the case where each job needs different types of resources. In addition to the actual scheduling of jobs, our task is to decide how much of the respective resource is allocated to each job, at a minimum total resource cost. *Balancing* the resource usage is not an issue at our higher-level planning stage.

Heuristics, rather than exact methods, are prominent for solving such complex scheduling problems. This is also because of the fact that mixed integer programming formulations for scheduling problems in general, and for ours in particular, often yield very weak bounds from the linear programming relaxation.

Our Contribution.

Besides presenting a heuristic that improves on the results recently reported in Megow et al (2011), we formulate a mixed integer program which is based on working shifts, and thus has an exponential number of variables. Our branch-price-and-cut algorithm to solve this model computes optimal schedules for instances with up to 50 jobs, which is a large number in this area of scheduling. In particular, the derived lower bounds demonstrate that our heuristic solutions are mostly near the optimum or at least near the best solution found by exact methods within half an hour of computation time. As a side effect we demonstrate how well-known valid inequalities from the literature can help strengthening bounds also in column generation approaches to scheduling problems.

2 Formal Problem Description

For a recent survey on resource-constrained project scheduling (RCPSP) we 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 between jobs are given as an acyclic digraph $G = (\mathcal{J}, E)$ with $ij \in E$ iff job i has to be finished before job j starts. Each job j may be run in exactly one out of a 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 $k \in \mathcal{R}$. Due to a fine granularity of planning, in our setting 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 finish before T , the time horizon. Each resource $k \in \mathcal{R}$ has a set $\mathcal{S}_k := \{[a_1, b_1], \dots, [a_{k_i}, b_{k_i}]\}$ of $k_i \in \mathbb{N}$ availability periods, also called *shifts*, where $a_1 < b_1 < \dots <$

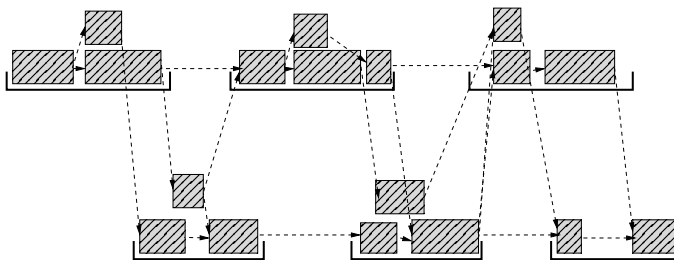


Figure 1 Schematic representation of turnaround scheduling with two resources.

$a_{k_i} < b_{k_i}$. A job requiring resource k can only be executed during a time interval $I \in \mathcal{I}_k$, see Fig. 1. 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 we have to determine the available capacity R_k such that at any time the total resource requirement of all the jobs does not exceed R_k .

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

Following the extended $\alpha|\beta|\gamma$ -classification scheme (Brucker et al, 1999), we consider $MPSm, \infty|prec, shifts|\sum c_k \cdot \max r_k(S, M, t)$ for multi-mode project scheduling with m renewable resources of unbounded capacity, with precedence constraints and working shifts, with the objective to minimize the total *resource availability cost*, i.e., minimizing $\sum_{k \in \mathcal{R}} c_k \cdot R_k$.

Related Work.

Turnaround scheduling comprises project scheduling with calendars, multi-mode scheduling, and resource leveling; see Megow et al (2011) for an industrial application. The zoo of 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 by 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, Bianco and Caramia (2011) derive lower bounds 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 et al (2003).

The resource leveling problem with single-modes per job, which is denoted by $PS|temp|\sum c_k \max r_k(S, t)$ with general temporal 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., by Demeulemeester (1995) and Möhring (1984). These authors competed on the same instance set which contained about 16 jobs and four resources, with a time horizon between 47 and 70. Further computational studies were done containing 15 to 20 jobs and four resources. In the same setting, Drexl and Kimms (2001) propose lower bound computations, one based on Lagrangian relaxation, and one based on a column generation procedure, where variables represent schedules as in our approach. 20 jobs of small sizes 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).

Calendars have been taken into account in previous algorithms as well. Scheduling problems with fixed processing times and calendars, but without resource capacities were considered by Zhan (1992) who provides an exact pseudo-polynomial time algorithm (turned

into a polynomial one by Franck et al, 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 and 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 time-lags. On the other hand a job may need more than one resource. 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 our own test set (see Section 6).

3 Mixed Integer Programming Formulations

For solving large-scale scheduling problems, mixed integer programming (MIP) is not considered as primary choice since the linear programming (LP) relaxations may be weak. Huge numbers of variables and constraints may result in high computation times and memory failures for 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, see e.g., Desrosiers and Lübbecke (2005).

3.1 Obstacles of Mixed Integer Programming for RCPSP

One of the most prominent models for the RCPSP was introduced already by Pritsker et al (1969). Their formulation adapted to resource leveling looks as follows:

$$\begin{aligned}
 & \min \sum_k c_k \cdot \bar{R}_k & (1) \\
 \text{s.t.} \quad & \sum_t x_{jt} = 1 & \forall j \in \mathcal{J} & (2) \\
 & \sum_t t \cdot x_{jt} = S_j & \forall j \in \mathcal{J} & (3) \\
 & S_i + p_i \leq S_j & \forall ij \in E & (4) \\
 & \sum_{j \in \mathcal{J}} \sum_{\substack{\tau=t-p_j+1 \\ \tau \geq 0}}^t r_{jk} \cdot x_{j\tau} \leq \bar{R}_k & \forall k \forall t & (5) \\
 & x_{jt} \in \{0, 1\} & \forall j \forall t & (6)
 \end{aligned}$$

Binary variables x_{jt} model whether job j starts at time t or not. Each job j must start exactly once (2). The (continuous) start times $S_j \geq 0$ are linked to the binary variables x_{jt} in (3). Also precedence constraints (4), and resource capacity constraints (5) are linear. The mixed integer program decides on the (continuous) resource capacities $\bar{R}_k \geq 0$ for each resource k , such that the total resource availability cost is minimized.

Depending on several factors, such as network complexity (the density of G) or the time discretization considered, this formulation may give poor 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. We will informally call this a “smearing” of start time variables. This smearing gives us irrelevant information about the schedule and we lose all structure in the model.

Furthermore, branching on the binary variables leads to an unbalanced search tree, since branching to zero is a very weak decision if the job can be scheduled one time unit earlier or later (that is, the decision essentially has no effect). Thus, one needs a more sophisticated branching rule that is aware of the linking of continuous variables S_j and binary variables x_{jt} and that prefers branching on the start time variables S_j . Therefore, natural branching candidates are start time variables whose corresponding binary variables are fractional. It turns out that this intuition goes very well with our approach.

3.2 Master Problem: A Model based on Shift Configurations

In order to reduce the effects of “losing the timing information” because of 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 generated independently for each resource. These configurations are linked by constraints ensuring that exactly one is chosen for each working shift. For each such configurations we introduce a binary variable x_ξ which indicates whether configuration ξ is chosen. We abbreviate $j \in \xi$ to express that job j is executed in the shift corresponding to configuration ξ . Every ξ has an associated resource usage R_ξ and start times $S_{j\xi}$ and completion times $C_{j\xi}$ for each $j \in \xi$. Note that the mode of each job is determined by the respective start and completion times. The so-called master problem reads:

$$\min \sum_k c_k \cdot \bar{R}_k \quad (7)$$

$$\text{s.t.} \quad C_i \leq S_j \quad \forall ij \in E \quad (8)$$

$$S_j = \sum_{\xi: j \in \xi} S_{j\xi} \cdot x_\xi \quad \forall j \in \mathcal{J} \quad (9)$$

$$C_j = \sum_{\xi: j \in \xi} C_{j\xi} \cdot x_\xi \quad \forall j \in \mathcal{J} \quad (10)$$

$$\sum_{\xi: \xi \in I} R_\xi \cdot x_\xi \leq \bar{R}_k \quad \forall k \forall I \in \mathcal{I}_k \quad (11)$$

$$\sum_{\xi: j \in \xi} x_\xi = 1 \quad \forall j \in \mathcal{J} \quad (12)$$

$$x_\xi \in \{0, 1\} \quad \forall \xi \quad (13)$$

Each job is executed in exactly one configuration by (12). The start and completion times for each job are computed from the chosen configurations via the linking constraints (9) and (10). Constraints (8) model the precedence relations between jobs. These could be directly expressed by substituting S_j and C_j from the linking constraints, but (9) and (10) are

helpful in the pricing problem (Section 3.3) where they penalize or encourage certain start or completion times of jobs. Constraints (11) link resource consumptions to the capacities.

3.3 Column Generation: Pricing Problem

Since the number of feasible configurations is exponential in the number of jobs, we solve the LP relaxation by column generation. That is, we start with a very small (e.g., heuristically generated) subset of configuration variables, and dynamically add more to the model until one can prove that no more promising variables exist. This optimality proof is given—as in the standard simplex method—via non-negativity of reduced costs of all configuration variables. We now describe the *pricing subproblem* which is used to generate promising configuration variables if they still exist.

We denote the dual variables of constraints (9), (10), (11), and (12) by s_j, c_j, ρ, π_j , respectively. We formulate a pricing problem for each shift I in which only the subset $J \subseteq \mathcal{J}$ of the jobs that can be scheduled in I needs to be considered. The objective function (14) reflects minimizing the reduced cost.

$$\max \sum_j \pi_j \cdot X_j - \sum_j c_j \cdot C_j + \sum_j s_j \cdot S_j - \rho \cdot R \quad (14)$$

$$\text{s.t.} \quad X_j = \sum_{m,t} x_{jmt} \quad \forall j \in J \quad (15)$$

$$S_j = \sum_{m,t} t \cdot x_{jmt} \quad \forall j \in J \quad (16)$$

$$C_j = \sum_{m,t} (t + p_{jm}) \cdot x_{jmt} \quad \forall j \in J \quad (17)$$

$$\sum_{j \in \mathcal{J}} \sum_m \sum_{\substack{\tau=t-p_{jm}+1 \\ t \geq 0}}^t r_{jm} \cdot x_{jmt} \leq R \quad \forall t \in I \quad (18)$$

$$x_{jmt} \in \{0, 1\} \quad \forall j \in J, m, t \quad (19)$$

$$X_j \in \{0, 1\} \quad \forall j \in J \quad (20)$$

This is a scheduling problem with a non-regular objective function where a new configuration ξ for a specific shift I is generated. It must be decided, see Constraint (15), whether a job j corresponding to the binary decision variable X_j is running in this shift or not, and if so, which mode $m \in \mathcal{M}_j$ is used. Constraints (16) and (17) fix the start and completion times of jobs according to the chosen mode assignment. Resource capacity constraints (18) have to be satisfied such that the total profit is maximized. The objective value is increased by π_j if a job is taken into the configuration and by multiples of s_j and c_j if it has late start times and early completion times. With each unit increase of resource capacity the objective value decreases by a factor of ρ .

The pricing problem is NP-hard as it contains a 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 . This negative complexity result justifies solving the pricing problem as a mixed integer program. The interested reader will have noticed that our overall approach results from applying a Dantzig-Wolfe reformulation to the *original* mixed integer program (1)–(6). Note that integrality of the configuration variables x_ξ in the master problem (7)–(13) still needs to be ensured by branching, see Section 4.1. When the

LP relaxation in each node of the branch-and-bound tree is solved by column generation one speaks of *branch-and-price*.

4 Branch-Price-and-Cut Algorithm

A solution to the original problem 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 closest resource allocation, such that $p_{jm_j} \leq C_j - S_j$. In a modern branch-and-price context one tries to branch on these *original variables*, instead of on the master variables. A main reason is that branching decisions constitute additional constraints that give rise to additional dual variables that need to be respected in the pricing problem. Branching constraints formulated on the original variables only affect the subproblems' objective function, not their structure, which is very desirable (Desrosiers and Lübbecke, 2005, 2011).

4.1 Branching Scheme

Preliminary experiments revealed that not all branching decisions are of equal importance, so we choose to branch on the most important variables first. Only when these are already integer, we branch on the second-important class, and so forth. The order we choose is R_k , S_j , and then C_j . That is, branching on fractional resource capacity values has the largest impact. Start and completion time variables are considered as branching candidates only if any corresponding binary configuration variable is fractional. After the resource capacities are fixed in the search tree, a start time variable S_j with LP solution value S_j^* is selected. The node is split into two child nodes with $S_j \leq \lfloor S_j^* \rfloor$, and $S_j \geq \lceil S_j^* \rceil$, respectively. Completion times are handled accordingly. This scheme is used together with some propagation rules to overcome the “smeared” LP solutions and to create a more balanced search tree.

4.2 Propagation

Domains of variables can be tightened due to logical implications given by the constraints, and/or already fixed variables. For example, the precedence and resource constraints cover 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 the area of scheduling problems, a large variety of propagation algorithms is known that detect these 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. The first correct algorithm, proposed in Mercier and Van Hentenryck (2008) can be adapted to the multi-mode case, by using the minimum energy of all modes 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.

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. Adding cutting planes to the master problem is possible, but raises again (as with branching decisions) 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 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)–(6).

We start from the well-known precedence cuts first introduced by Christofides et al (1987) for the original formulation (1)–(6) and extend them to the multi-mode case:

$$\forall (i, j) \in E, \tau: \quad \sum_{t \geq \tau} x_{imt} + \sum_{t \leq \tau + p_{i,\min}} x_{jmt} \leq 1.$$

These cuts can be expressed in the master variables x_ξ as follows (again, a Dantzig-Wolfe reformulation is behind this):

$$\forall (i, j) \in E, \tau: \quad \sum_{\xi: S_{i\xi} \geq \tau} x_\xi + \sum_{\xi: C_{j\xi} \leq \tau + p_{i,\min}} x_\xi \leq 1. \quad (21)$$

As desired, the nature of the pricing problem (i.e., the constraints) does not change, only the coefficients of the objective function need to be updated. For each cut $(\tau, (i, j))$ we need to add the dual variable $\mu_{\tau ij}$ to the objective function if $S_{i\xi} \geq \tau$ or if $C_{j\xi} < \tau + p_{i,\min}$. The cost coefficient of variable x_{imt} becomes:

$$\sum_{ij \in E} \sum_{t \geq \tau} \mu_{\tau ij} + \sum_{ki \in E} \sum_{t < \tau + p_{im} - 1} \mu_{\tau ki} \quad (22)$$

5 Primal Heuristics

For the master problem, rounding heuristics for LP 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 extended a leveling heuristic from Megow et al (2011) and implemented a generic list scheduling algorithm which is used as a standalone heuristic during the branching process as well as in the leveling procedure.

5.1 Ready Scheduling Heuristic.

The general idea of *ready scheduling* is similar to that of list scheduling with jobs sorted by earliest start times. Jobs are divided according to the resource they need, and scheduled as soon as their predecessors are completed, if possible, thus increasing the chance to meet a given time horizon. We say a job is “ready” if all its predecessors are scheduled.

For each resource k we maintain a vector of jobs $J_k = (j_{k_1}, \dots, j_{k_{|J_k|}})$ that use this resource and have no unscheduled predecessors, together with a lower bound t_k on the next feasible start FS_j of any job $j \in J_k$. If J_k is empty, we set t_k to infinity.

The heuristic loops over t , which increases to the minimal t_k in each iteration. A subset $I \subset J_k$ of constant size s (we chose $s = 6$ in our computational studies) is scheduled so that the overall completion time is kept small (line 8). This is accomplished by trying all mode combinations for I recursively, and bounding recursion using the currently shortest feasible solution found in this way. If s is small, this can be done quickly. Finally, for each scheduled job in I , those successors which become ready, are added to the corresponding set J_k , each t_k is updated, and the next iteration begins. This process continues until all jobs are scheduled, or a makespan violation occurs.

Algorithm 1: Ready Scheduling

Input: Set of jobs \mathcal{J} to be scheduled and max. total duration T
Output: Job start times and modes, or that no solution was found

- 1 **forall the** $k \in \mathcal{R}$ **do**
- 2 Let $J_k \subset \mathcal{J}$ be the set of jobs that use resource k , and are ready;
- 3 $t_k := \min_{j \in J_k} FS_j$;
- 4 $t := 0$;
- 5 **while** $t < T$ **do**
- 6 $\ell := \operatorname{argmin}_k t_k$, and $t := t_\ell$;
- 7 $m := \min(s, |J_\ell|)$ with s a small constant and $I := \{j_{\ell_1}, \dots, j_{\ell_m}\}$;
- 8 Schedule jobs in I such that $\max_{i \in I} C_i$ is minimal ;
- 9 Add all successors of jobs to I that become ready to their respective J_k ;
- 10 $J_\ell := J_\ell \setminus I$;
- 11 Update t_k for all changed J_k as in Step 3 ;

5.2 Resource Leveling Heuristic.

We now describe the resource leveling heuristic by Megow et al (2011), and how the ready scheduler ties into the framework of the leveling procedure. This heuristic uses a binary search on the capacity bounds of the resources, greedily selecting the resource whose upper bound is to be improved in each iteration. This selection is based on a parameter μ_k , measuring how badly a resource k is leveled.

One iteration of the binary search consists of trying to find a feasible schedule for the current bounds. These bounds for the selected resource k^* are set to $(\text{UB}_{k^*} + \text{LB}_{k^*})/2$, UB_{k^*} and LB_{k^*} being the upper and lower bounds on the capacity of resource k^* , while all other resource bounds remain fixed. We try list scheduling, and on failure fall back to ready scheduling to prove the bounds feasible. If neither ready scheduling nor list scheduling yield a feasible schedule, we consider the current upper bound for the selected resource as a new lower bound, and the next iteration begins.

Algorithm 2: Resource Leveling**Input:** Set \mathcal{R} of resource types to be leveled, project duration T **Output:** Leveled resource utilization R_k for each resource type $k \in \mathcal{R}$

- 1 Set LB_k and UB_k to initial values for each resource type $k \in \mathcal{R}$;
- 2 **while** $\exists k \in \mathcal{R} : LB_k < UB_k$ **do**
- 3 Choose resource type $k^* \in \mathcal{R}$ with $LB_{k^*} < UB_{k^*}$ and μ_{k^*} maximum;
- 4 Perform binary search using list scheduling and ready scheduling in order to decrease the capacity bounds of k^* ;

5.3 LP Solution and Ready Scheduling Heuristics.

We finally present heuristics based on the LP relaxation of each node of the branch-and-bound tree. These heuristics set the maximal capacity of each resource to the LP solution value rounded up, and fix the earliest and latest start and completion times for each job to the global bounds of the corresponding variables. We perform list scheduling using the LP solution with jobs sorted by earliest completion times, and each job's mode is chosen as the one matching $C_i - S_i$ best. 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 steps.

6 Computational Study

6.1 Benchmark Instances

There is no publicly available test set of instances reflecting the setup of our problem. When compiling our own test set, the existing instances in the PSPLib (Kolisch and Sprecher, 1996) guided our design. 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 five to twelve. 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 $\lfloor \mathcal{J} \rfloor / W$, and randomly choosing the remaining edges.

There are two different resources which come in five calendar configurations, called C1 to C5. These calendars are described schematically in Fig. 2. In the top row, calendars C1 to C3 are shown. In each of these, the length of the shifts is 60. In C1 and C3 shift breaks

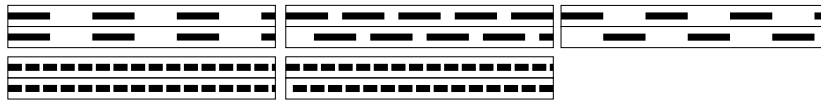


Figure 2 Calendar configurations C1–C3 (top) and C4 and C5 (bottom) used in our test set. Black bars symbolize the temporal location of shifts.

are 60 units long, in C2 only 20. 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 ten. 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 first makespan comes by scheduling all jobs at their highest resource usage and the second run is carried out by assigning the fewest number of resources to each job.

6.2 Experimental Setup

All experiments were done on Intel Core™ i7-870 PCs (2.93 GHz, 8MB cache, 8GB memory) running Linux 2.6.34 (single thread). Each test run had a time-limit of 30 minutes. 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, branching rules, cutting plane separation, and column generation. For the standard MIP (1)–(6) 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% are probably “noise.”

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 use of the heuristic throughout search is negligible, see Coughlan et al (2010).

Separation Procedure. Since there are $m \cdot T$ many precedence cuts (21), we need a good separation oracle. We pursue two approaches in order to add these cuts into the model. In our first approach we only check for a precedence pair (i, j) at time point $\tau_{greedy} = \lceil (C_i + S_j)/2 \rceil$ where equation (21) holds or is violated by more than some ε . In the second approach, we find the optimum point τ_{best} such that the left-hand side of equation (21) is maximally violated. This can be done by sorting the summands of equation (21) for each time point τ such that for each value τ the left-hand side can be computed in linear time (after sorting). This procedure is pseudopolynomial in the number of time points.

Settings. We compare the outcome of a CPLEX run on MIP model (1)–(6) to different settings of our branch-price-and-cut approach. Initially, we do not separate cuts, denoted by “nocuts.” Then, we evaluate how to separate the cuts. First, we separate them already in the root node, and second as it will turn out to be profitable, after resource capacity variables are fixed. Since it may not always be beneficial to check for the largest violation, we use a promising guess as timepoint: “ τ_{greedy} ” for each precedence pair (i, j) . Searching for the highest violation is denoted by “ τ_{best} ” which is computed as described above. Furthermore, these cuts are only separated if a certain threshold is exceeded. We call this the tolerance “tol.” and try tolerances 10^{-4} , 0.1, 0.3, 0.5 and 0.8, which range from a very small violation at which the cut is separated to a very high violation.

6.3 Results

Very often a Dantzig-Wolfe decomposition can improve the lower bound obtained from the standard linear relaxation by far. Nevertheless, the cutting plane algorithms of commercial

MIP solvers are sophisticated alternatives to improve the dual bounds. In Fig. 3 we see that the average improvement per calendar of the Dantzig-Wolfe decomposition over the LP relaxation of (1)–(6) lies between 20% to 65% throughout all instances. The displayed average deviations and the minimum and maximum values of that improvement show the strength of our relaxation.

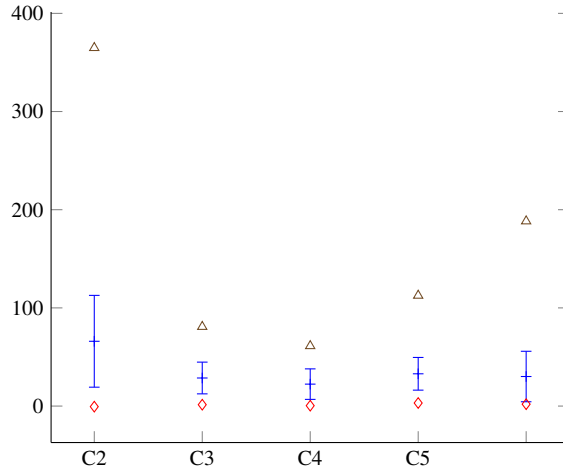


Figure 3 Percentage of improvement of the lower bound obtained from our branch-and-price formulation (7)–(13) over the standard LP relaxation of (1)–(6) as computed by CPLEX. The minimum (diamond), maximum (triangle), mean, and standard deviation of the improvement in percent of setting “no cuts” is shown.

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 cutting planes have been added by CPLEX and our solver. 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% improvement on the average of the root dual bound compared to CPLEX.

Now, we compare our branch-price-and-cut framework to a standard MIP approach in terms of absolute number of solved instances and afterwards, 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. 3 the first bar of each calendar (black) gives the number of solved instances obtained by CPLEX, the second (white) bar represents this number for the branch-price-and-cut 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. 3 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. 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. Fig. 3 clearly shows that a high tolerance (tol.), so that fewer (and stronger) precedence cuts are added, leads to better results. Hence, not using any precedence cuts at all is, at first sight, a good decision.

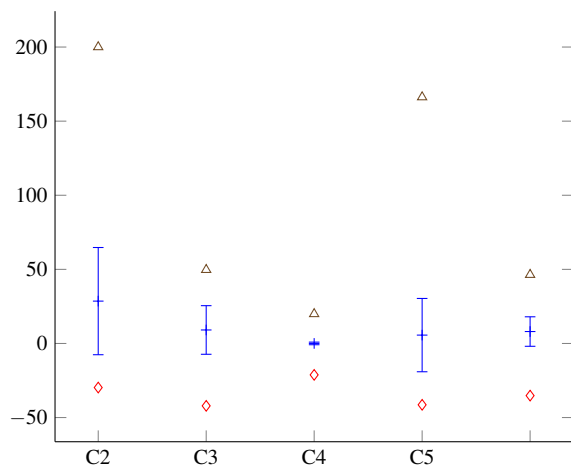


Figure 4 Average mean and standard deviation of root dual bound improvement of our branch-price-and-cut approach as compared to CPLEX in percent. The best bound over all tolerances has been used. Triangles show the best obtained improvement (e.g., 200% in C1) and diamonds show the worst lower bound (e.g.,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

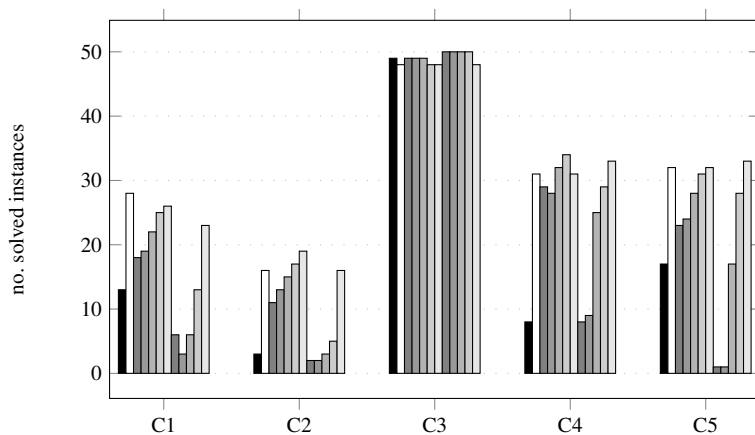


Figure 5 Number of solved instances for N50E70 with cuts already separated in the root node; each set of bars from left to right corresponds to rows in Table 1. A CPLEX run on the standard MIP is black; our branch-and-price algorithm without cuts is white; and the two groups of different shades of grey 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).

Nevertheless, it is possible to increase the root dual bound on several instances by using precedence cuts (21). 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 afterwards on 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

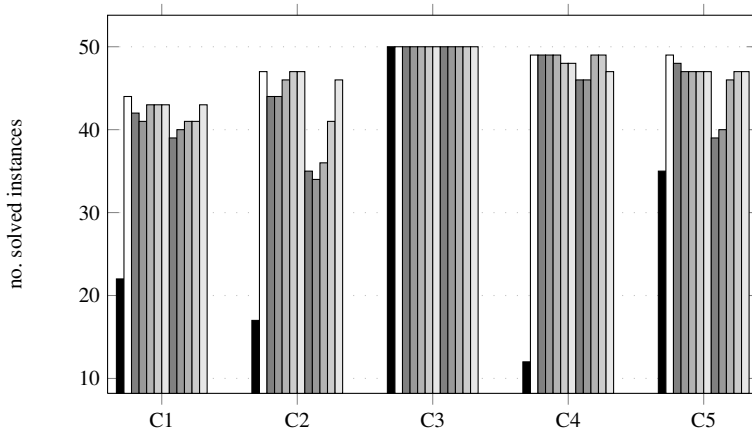


Figure 6 Number of solved instances for N50E100 with cuts already separated in the root node.

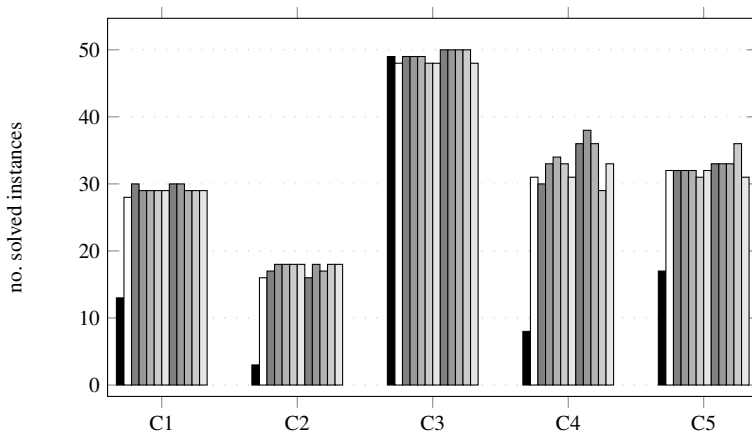


Figure 7 Number of solved instances of test set N50E70 if cuts are separated after resource capacity variables are fixed.

seems to be a good point to separate precedence cuts (21). 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. A complete list of the following numbers can be found in the appendix.

Tables 1 and 2 reveal that using additional precedence cuts enables us to decrease the number of tree nodes by 10%–20% on average. For several hard instances, e.g., in calendar C5 in test set N50E70 a decrease by even 50% is possible. Best results in terms of nodes

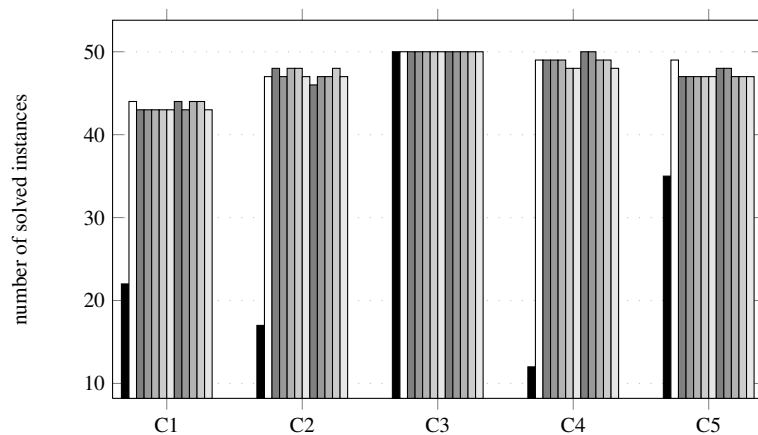


Figure 8 Number of solved instances of test set N50E100 if cuts are separated after resource capacity variables are fixed.

Table 1 Comparison of tree nodes and time 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 for Calendars C1 to C5.

		C1		C2		C3		C4		C5		
	tol.	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	
τ_{greedy}	no cuts		12.0	660.7	10.7	546.3	1.0	6.4	11.4	200.5	21.3	356.2
	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

are obtained when τ_{best} is computed. 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% 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. Furthermore, Table 5 in the appendix shows that the best lower bounds are obtained using precedence cuts.

7 Conclusions

We studied a complex multi-mode resource leveling problem, and our aim was to obtain optimal solutions. The literature on related problems and in particular the PSPLib with up-to-date benchmark results suggest that even today instances with only 30 jobs are hard to solve to optimality. Our study demonstrates that it helps to decompose the problem resource-wise. Technically, this means applying a Dantzig-Wolfe reformulation to a standard formulation

Table 2 Comparison of tree nodes and time for N50.E100 – 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 for Calendars C1 to C5 respectively.

	tol.	C1		C2		C3		C4		C5	
		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

as a mixed integer program. The resulting model is based on variables that represent entire schedules for each resource, of which there are exponentially many. This necessitates a solution via a column generation and branch-and-price algorithm. A usually complicating feature, the resource availability periods (or shift calendars), is even of help in this context, as this reduces the size of the variable generating subproblems to be solved.

The linear programming relaxations of the reformulated model are much stronger than that of the standard approach. This can even be strengthened by applying additional valid inequalities which seamlessly integrate into 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 of the 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,” and we plan to investigate this further.

References

- Achterberg T (2009) SCIP: solving constraint integer programs. *Math 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 J Oper Res* 127(2):355–362
- Brucker P, Drexel A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: Notation, classification, models, and methods. *European J Oper Res* 112:3–41
- Christofides N, Alvarez-Valdes R, Tamarit JM (1987) Project scheduling with resource constraints: A branch and bound approach. *European J Oper Res* 29(3):262 – 273
- Coughlan ET, Lübbecke ME, Schulz J (2010) A branch-and-price algorithm for multi-mode resource leveling. In: Festa P (ed) SEA, Springer, Lecture Notes in Computer Science, vol 6049, pp 226–238
- Demeulemeester E (1995) Minimizing resource availability costs in time-limited project networks. *Management Sci* 41:1590–1598

- Demeulemeester E, Herroelen W (2002) Project Scheduling: A Research Handbook. Kluwer
- Desrosiers J, Lübbecke M (2005) A primer in column generation. In: Desaulniers G, Desrosiers J, Solomon M (eds) Column Generation, Springer-Verlag, Berlin, pp 1–32
- Desrosiers J, Lübbecke M (2011) Branch-price-and-cut algorithms. In: Cochran J (ed) Encyclopedia of Operations Research and Management Science, John Wiley & Sons, Chichester
- 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. Ann Oper Res 102(1-4):111–135
- Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. European J Oper Res 207(1):1–14
- Kolisch R, Sprecher A (1996) PSPLIB - A project scheduling problem library. European J Oper Res 96:205–216, <http://129.187.106.231/psplib/>, last accessed 2011/05/26
- Megow N, Möhring R, Schulz J (2011) Decision support and optimization in shutdown and turnaround scheduling. INFORMS J Computing 23(2):189–204
- Mercier L, Van Hentenryck P (2008) Edge finding for cumulative scheduling. INFORMS J Computing 20(1):143–153
- Möhring R (1984) Minimizing costs of resource requirements in project networks subject to a fixed completion time. Oper Res 32(1):89–120
- Neumann K, Schwindt C, Zimmermann J (2003) Project scheduling with time windows and scarce resources. Springer
- Pritsker A, Watters L, Wolfe P (1969) Multi project scheduling with limited resources: A zero-one programming approach. Management Sci 16:93–108
- 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 Oper Res 36(5):423–438

Table 3 Comparison of the number of solved instances for test set N50E70 and below N50E100. In the settings with precedence cuts, these are separated already in the root node.

	tol.	C1	C2	C3	C4	C5
CPLEX		13	3	49	8	17
no cuts		28	16	48	31	32
τ_{greedy}	0.0001	18	11	49	29	23
	0.1	19	13	49	28	24
	0.3	22	15	49	32	28
	0.5	25	17	48	34	31
	0.8	26	19	48	31	32
τ_{best}	0.0001	6	2	50	8	1
	0.1	3	2	50	9	1
	0.3	6	3	50	25	17
	0.5	13	5	50	29	28
	0.8	23	16	48	33	33
	tol.	C1	C2	C3	C4	C5
CPLEX		22	17	50	12	35
no cuts		44	47	50	49	49
τ_{greedy}	0.0001	42	44	50	49	48
	0.1	41	44	50	49	47
	0.3	43	46	50	49	47
	0.5	43	47	50	48	47
	0.8	43	47	50	48	47
τ_{best}	0.0001	39	35	50	46	39
	0.1	40	34	50	46	40
	0.3	41	36	50	49	46
	0.5	41	41	50	49	47
	0.8	43	46	50	47	47

Table 4 Comparison of the number of solved instances for test set N50E70 and below N50E100 where precedence cuts are separated after all variables for resource capacities are fixed.

	tol.	C1	C2	C3	C4	C5
CPLEX		13	3	49	8	17
no cuts		28	16	48	31	32
τ_{greedy}	0.0001	30	17	49	30	32
	0.1	29	18	49	33	32
	0.3	29	18	49	34	32
	0.5	29	18	48	33	31
	0.8	29	18	48	31	32
τ_{best}	0.0001	30	16	50	36	33
	0.1	30	18	50	38	33
	0.3	29	17	50	36	33
	0.5	29	18	50	29	36
	0.8	29	18	48	33	31
	tol.	C1	C2	C3	C4	C5
CPLEX		22	17	50	12	35
no cuts		44	47	50	49	49
τ_{greedy}	0.0001	43	48	50	49	47
	0.1	43	47	50	49	47
	0.3	43	48	50	49	47
	0.5	43	48	50	48	47
	0.8	43	47	50	48	47
τ_{best}	0.0001	44	46	50	50	48
	0.1	43	47	50	50	48
	0.3	44	47	50	49	47
	0.5	44	48	50	49	47
	0.8	43	47	50	48	47

Table 5 The number of best lower and best upper bounds that were found throughout search are displayed. for N50.E70 and below N50.E100. Cuts are separated after resource variables are fixed. ‘nS’ is the number of optimally solved instances, ‘bLB’ (‘bUB’) denotes how often the best lower (upper) bound was found.

tol.	C1			C2			C3			C4			C5		
	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB
no cuts	28	45	48	16	48	39	48	48	50	31	48	39	32	47	43
τ_{greedy}	30	45	49	17	48	41	49	49	50	30	49	37	32	47	41
	29	46	49	18	49	43	49	49	50	33	49	40	32	46	42
	29	46	49	18	49	40	49	49	50	34	49	42	32	46	43
	29	46	49	18	49	42	48	48	50	33	49	40	31	46	44
	29	47	49	18	49	42	48	48	50	31	48	39	32	48	43
	30	45	50	16	48	40	50	50	50	36	50	43	33	49	43
	30	47	48	18	47	41	50	50	50	38	49	44	33	49	44
τ_{best}	29	47	49	17	49	40	50	50	50	36	49	45	33	49	44
	29	47	49	18	49	42	50	50	50	29	50	38	36	49	46
	29	47	49	18	48	40	48	48	50	33	48	41	31	48	43

tol.	C1			C2			C3			C4			C5		
	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB
no cuts	44	48	50	47	49	50	50	50	50	49	49	50	49	49	50
	43	47	48	48	50	50	50	50	50	49	49	50	47	47	50
	43	46	49	47	50	49	50	50	50	49	49	50	47	47	50
τ_{greedy}	43	46	49	48	50	50	50	50	50	49	49	50	47	47	50
	43	46	50	48	50	50	50	50	50	48	48	50	47	47	49
	43	46	50	47	49	50	50	50	50	48	48	50	47	47	49
	44	48	48	46	49	49	50	50	50	50	50	50	48	48	50
τ_{best}	43	47	48	47	50	49	50	50	50	50	50	50	48	48	50
	44	47	50	47	50	48	50	50	50	49	49	50	47	47	50
	44	47	49	48	50	50	50	50	50	49	49	50	47	47	50
	43	46	50	47	49	50	50	50	50	48	48	50	47	47	49

Table 6 N50E100 - AR: 0

Cat.	N	tol.	$\tau = C_i + S_i/2$						T_{best}						no cuts			alls
			solved	nodes	time	rbfgap	rbfi	solved	nodes	time	rbfgap	rbfi	solved	nodes	time			
1h2	50	0.0	42	4.3	28.7	103.2	9	39	3.2	84.1	114.3	25	44	4.2	15.5	38		
		0.1	41	3.4	28.7	104.2	10	40	3.1	71.8	114.3	26						
		0.3	43	3.3	20.7	102.3	5	41	3.1	51.3	113.1	25						
		0.5	43	4.5	18.3	101.5	3	41	4.3	33.0	106.4	19						
		0.8	43	4.4	16.2	100.4	1	43	4.3	16.8	103.1	4	22	1690	37.3			
1hC2	50	0.0	44	4.7	27.4	100.9	24	35	4.8	209.3	107.3	27	47	4.7	14.4	31		
		0.1	44	4.9	28.8	100.7	24	34	5.0	213.9	107.7	24						
		0.3	46	4.5	20.6	100.6	21	36	4.6	121.5	108.1	27						
		0.5	47	4.4	16.4	100.2	15	41	5.0	43.5	103.3	28						
		0.8	47	4.4	14.2	100.0	2	46	4.3	16.7	100.5	12	17	3890	67.9			
1hXC2	50	0.0	50	1.1	1.0	0.0	0	50	1.1	1.1	103.9	2	50	1.1	1.0	50		
		0.1	50	1.1	1.0	0.0	0	50	1.1	1.1	110.0	2						
		0.3	50	1.1	1.0	0.0	0	50	1.1	1.1	108.6	0						
		0.5	50	1.1	0.9	0.0	0	50	1.1	1.0	0.0	0						
		0.8	50	1.1	0.9	0.0	0	50	1.1	0.9	0.0	0	50	1	1.2			
20m5hC2	50	0.0	49	5.0	16.1	100.6	21	46	4.8	69.5	106.0	23	49	5.3	10.0	43		
		0.1	49	4.8	14.9	100.5	19	46	4.8	69.1	106.5	23						
		0.3	49	5.1	12.0	100.2	17	49	4.6	21.8	102.9	23						
		0.5	48	5.2	10.9	100.2	6	49	4.9	12.6	100.7	20						
		0.8	48	5.6	10.3	0.0	0	47	5.8	10.9	100.3	1	12	3449	129.8			
20m5hX	50	0.0	48	4.5	21.8	101.2	41	39	5.0	119.0	109.0	34	49	5.1	8.5	38		
		0.1	47	5.2	20.7	101.1	40	40	5.2	103.9	108.6	35						
		0.3	47	5.1	13.1	100.6	29	46	4.5	28.3	103.1	41						
		0.5	47	5.9	10.3	100.3	5	47	4.8	13.1	100.6	32						
		0.8	47	5.4	9.3	100.1	1	47	6.4	9.5	100.2	3	35	3072	70.8			

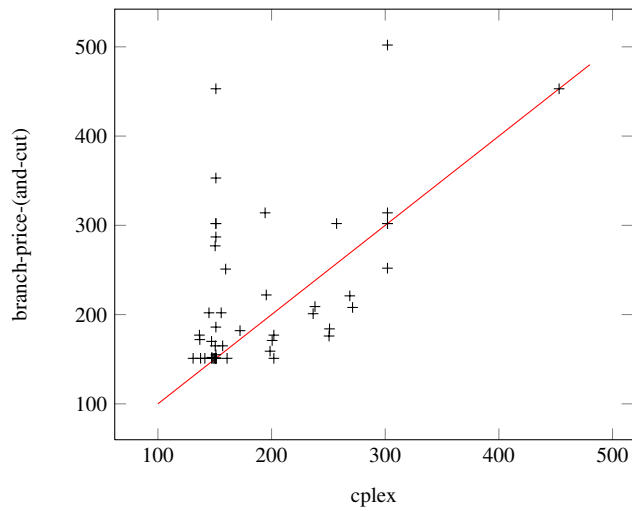


Figure 9 Comparison of lower bounds obtained by CPLEX and our branch-and-price algorithm without separation of additional precedence cuts. All points above the red line indicate that the lower bound obtained by the branch-and-price routine are stronger.

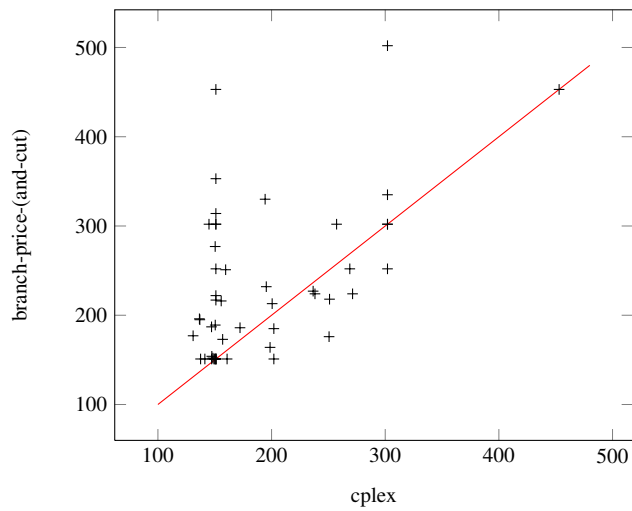


Figure 10 Comparison of lower bounds obtained by CPLEX and our branch-and-price algorithm using τ_{best} with tolerance 0.001. All points above the red line indicate that the lower bound obtained by the branch-and-price routine are stronger.