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

Eamonn T. Coughlan[1], Marco E. Lübbecke[2], and Jens Schulz[1]

[1] Technische Universität Berlin, Institut für Mathematik, Straße d. 17. Juni 136, 10623 Berlin, Germany, {coughlan,jschulz}@math.tu-berlin.de

[2] Technische Universität Darmstadt, Fachbereich Mathematik, Dolivostr. 15, 64293 Darmstadt, Germany, luebbecke@mathematik.tu-darmstadt.de

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 a branch-and-price approach together with a new heuristic to solve the more general turnaround scheduling problem. Besides precedence and resource constraints, also 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* [MMS09]. In turnaround scheduling, for inspection and renewal of parts, plants are shut down, disassembled, and rebuilt, so there is a partial ordering of jobs to be done. 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.

Different types of renewable resources are given, each associated with *availability periods* which can be thought of as working shifts. Besides the actual scheduling of jobs, the task is to decide how many workers need to be assigned to each job such that working costs are minimized, that is, we must determine a minimum amount of resources needed. We break down the granularity of planning, so that each job needs exactly one resource, possibly several units of which.

Heuristics, rather than exact methods, are prominent for solving such complex scheduling problems. This is also due to 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. We approach the turnaround scheduling from both ends. Besides presenting a heuristic which improves on the results reported in [MMS09], we formulate a mixed integer program which is based on working shifts, and thus has an exponential number of variables. A branch-and-price 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.

2 Formal Problem Description

For a recent survey on resource-constrained project scheduling (RCPSP) we refer to [HB09]. 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 $\mathcal{I}_k := \{[a_1, b_1], \ldots, [a_{k_i}, b_{k_i}]\}$ of $k_i \in \mathbb{N}$ availability periods, also called *shifts*, where $a_1 < b_1 < \ldots < a_{k_i} < b_{k_i}$. A job requiring resource k can only be executed during a time interval $I \in \mathcal{I}_k$. 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 $S = (S_1, \ldots, S_n)$ 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 $A(S, M, t) := \left\{ j \in \mathcal{J} : S_j \leq t < S_j + p_{jm_j} \right\}$ 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. 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 $ij \in E$. The objective of the resource leveling problem is to minimize the *resource availability costs* $\sum_{k \in \mathcal{R}} c_k \cdot R_k$.

Following the extended $\alpha|\beta|\gamma$-classification scheme [BDM+99], 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 resource availability cost.

Related Work. Turnaround scheduling comprises project scheduling with calendars, multi-mode scheduling, and resource leveling; see [MMS09] for an industrial application. The zoo of scheduling problems is large, and we give an idea only of the most related problems. Makespan minimization is a classical scheduling goal. In contrast to this regular objective function, i.e., being non-decreasing in the completion time of the jobs, a measure of the variation of resource utilization, e.g., $f(r_k(S,t))$ is not regular; see e.g., [NSZ03].

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 [Dem95] and [Möh84]. They 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, [DK01] propose lower bound computations, one based on Lagrangean relaxation, and one based on a column generation procedure, where variables represent schedules as in our approach. Small job sizes with 20 jobs can be handled; but for 30 jobs the Lagrangean relaxation wins against the column generation approach.

Multi-mode (also known as *malleable*) 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., [DH02,Har01].

For benchmarking, different problem sets are available in the PSPLib [PSP], 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 and include time-lags. On the other hand a job may need more than one resource. Even though none of these problems is suited for a direct comparison, they are similar to ours, and the mentioned instances inspired us when generating our own test set (see Sect. 5).

Algorithms have also taken calendars into account. Scheduling problems with fixed processing times and calendars, but without resource capacities were considered by [Zha92] who provides an exact pseudo-polynomial time algorithm (turned into a polynomial one by [FNS01]) for computing earliest and latest start times for preemptable as well as non-preemptable jobs.

3 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 only the LP relaxation.

3.1 Obstacles of Integer Programming for RCPSP

One of the most prominent models for the RCPSP was introduced by [PWW69]. Their formulation adapted to resource leveling looks as follows:

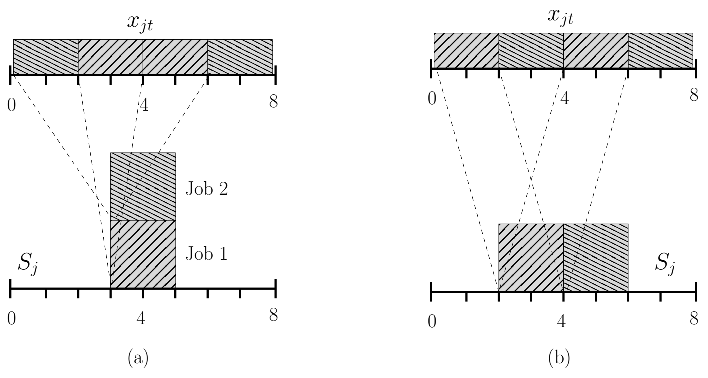

Figure 1. Example for a schedule where the resource demands are not counted correctly regarding the start time variables. In (a) a resource capacity of 4 is needed but the LP solution just yields a capacity bound of 1; the resource demand of job 2 is counted at points in time totally different from the start time S_2. In (b) an optimal schedule is given for the integer start time variables but the corresponding binary variables pretend that there is an optimality gap.

$$\min \sum_k c_k \cdot \bar{R}_k \tag{1}$$

$$\text{s.t.} \quad \sum_t x_{jt} \geq 1 \qquad \forall\, j \in \mathcal{J} \tag{2}$$

$$\sum_t t \cdot x_{jt} = S_j \qquad \forall\, j \in \mathcal{J} \tag{3}$$

$$S_i + p_i \leq S_j \qquad \forall\, ij \in E \tag{4}$$

$$\sum_{\tau=t-p_j+1}^{t} r_{jk} \cdot x_{j\tau} \leq \bar{R}_k \qquad \forall\, k\, \forall\, t \tag{5}$$

$$x_{jt} \in \{0,1\} \qquad \forall\, j\, \forall\, t \tag{6}$$

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

Depending on several factors, as network complexity (the density of G) or the time discretization considered, this formulation may yield good or poor lower bounds. We show an example LP solution, where even an optimal assignment of the start time variables S_j does not yield an optimum solution value.

Example. In Fig. 1 two jobs are given, each with processing time 2 and resource demand 2. We observe that the resource demand is easily underestimated in an arbitrary LP solution. In Fig. 1(a) the two jobs are running in parallel and need a resource capacity of 4, but an arbitrary LP solution with start times $S_1 = S_2 = 3$ is not necessarily integral and obtains a resource capacity of 1 as lower bound. This is achieved by setting the binary variables $x_{1,2} = x_{1,4} = x_{2,0} = x_{2,6} = 0.5$. Even if we are given an optimal solution $S_1 = 2$ and $S_2 = 4$, as in Fig. 1(b), the capacity is still underestimated. The convex combination (3) of far apart start times of a job to form S_j gives us irrelevant information about the schedule and we lose all structure in the model. Already the minimization of the makespan for a given capacity cannot be solved efficiently for job sets of size 60 [PSP].

Branching on the binary variables leads to a confusing result. In Fig. 1(b), when branching on $x_{1,0}$, job 1 which starts at time $S_1 = 2$ now would be scheduled at $t = 0$ or not. Thus, 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 is desirable.

3.2 Master Problem: A Model based on Shift Configurations

In order to reduce the effects of "losing the timing information" just described, we propose a model which exploits the problem structure by decomposing the time horizon. For each schedule, or *configuration*, of a single resource for a single shift, 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 ξ. For each ξ we know its resource capacity R_ξ and start time $S_{j\xi}$ and completion time $C_{j\xi}$ of each $j \in \xi$. Note that the mode of each job is determined by the start and completion times.

$$\min \sum_k c_k \cdot \bar{R}_k \tag{7}$$

$$\text{s.t.} \quad C_i \leq S_j \qquad \forall\, ij \in E \tag{8}$$

$$S_j = \sum_{\xi : j \in \xi} S_{j\xi} x_\xi \qquad \forall j \in \mathcal{J} \tag{9}$$

$$C_j = \sum_{\xi : j \in \xi} C_{j\xi} x_\xi \qquad \forall j \in \mathcal{J} \tag{10}$$

$$\sum_{\xi : \xi \in I} R_\xi x_\xi \leq \bar{R}_k \qquad \forall k\ \forall\, I \in \mathcal{I}_k \tag{11}$$

$$\sum_{\xi : j \in \xi} x_\xi \geq 1 \qquad \forall j \in \mathcal{J} \tag{12}$$

$$x_\xi \in \{0, 1\} \qquad \forall \xi \tag{13}$$

Each job is executed in at least 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 precedences between jobs which could be directly expressed by substituting S_j and C_j from the linking constraints, but (9) and (10) are helpful in the upcoming pricing problem 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

There are too many configurations to explicitly include them in the master problem, therefore we solve the relaxation by column generation embedded into a branch-and-bound tree. By defining dual variables s_j, c_j, ρ, π_j for constraints (9), (10), (11), and (12), respectively, we obtain a pricing problem for each shift I.

$$\max \quad \sum_j \pi_j X_j - \sum_j c_j C_j + \sum_j s_j S_j - \rho R$$

$$\begin{aligned}
s.t. \qquad X_j &= \sum_{m,t} x_{jmt} && \forall j \in J && (14)\\
S_j &= \sum_{m,t} t x_{jmt} && \forall j \in J && (15)\\
C_j &= \sum_{m,t} (t + p_{jm}) x_{jmt} && \forall j \in J && (16)\\
\sum_j \sum_{m,\tau : t \in [x_{jm\tau} + p_{jm}]} r_{jm} x_{jm\tau} &\le R && \forall t \in I && (17)\\
x_{jmt} &\in \{0,1\} && \forall j \in J, m, t && (18)\\
X_j &\in \{0,1\} && \forall j \in J && (19)
\end{aligned}$$

This is a scheduling problem with non-regular objective function where a new configuration ξ for a specific shift I is generated. It must be decided, see constraint (14), whether a job 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 (15) and (16) fix the start and completion times of jobs according to the chosen mode assignment. Resource capacity constraints (17) 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. This can be seen when all π_j are set to a value large enough to ensure that each job must be scheduled, and by setting s_j and c_j to zero for all j.

4 Branch-and-price 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 $p_{jm_j} \le C_j - S_j$. We refer to R_k, S_j, C_j as original or *problem variables*.

The variables of the pricing problem x_ξ that symbolize configurations of different shifts are generated during the branch-and-price loop and whenever a heuristic finds a feasible solution. We refer to these variables as *pricing variables*.

4.1 Branching scheme

Experiments revealed as branching order R_k, S_j, and then C_j. First, variables R_k are branched on until they are fixed even if they are integral, then the start time variables S_j are branched on and lastly the completion time variables C_j. A variable with largest domain is selected in each rule. Branching on configuration variables is not an option since this interferes with the pricing problem and would unduly complicate it.

4.2 Propagation

For scheduling problems a large variety of propagation algorithms is known from constraint programming (CP). *Edge-finding* is a CP technique concerned with deriving better bounds for earliest start and latest end times of jobs using energy arguments. The first correct algorithm, proposed in [MVH08] can be adapted to the multi-mode case, by using the minimum energy of all modes for each job. While this naturally seems to give weaker bounds, the fact that jobs are not preemptive and may not cross shift-bounds, makes this a powerful propagator during branch-and-bound.

Propagation also serves the technical purpose to communicate the branching decisions to the pricing problem in the beginning of node processing and whenever local bounds of the problem variables change during node processing.

4.3 Primal bounds

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 therefore solution values of the start and completion time variables are very unlikely to fit to any configuration variable priced so far. It could happen, that a starting time falls between two availability periods but the non-zero configuration variables use completely different shifts, see Fig. 1.

To improve upper bounds we extended a leveling heuristic from [MMS09] and implemented a generic list scheduling algorithm which is used as a stand alone heuristic during the branching process as well as in the leveling procedure.

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 set of jobs $J_k = (j_{k_1}, .., j_{k_{|J_k|}})$ 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 ∞.

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  for $k \in \mathcal{R}$ do
2      Let $J_k \subset \mathcal{J}$ be the set of jobs that use resource $k$, and are ready.
3      Set $t_k$ to $\min_{j \in J_k} FS_j$ .
4  Set $t := 0$
5  while $t < T$ do
6      $\ell := \operatorname{argmin}_k t_k$, and $t := t_l$
7      $m := \min(s, |J_\ell|)$ with $s$ a small constant and $I := \{j_{\ell_1}, .., j_{\ell_m}\}$
8      Schedule jobs in $I$ such that $\max_{i \in I} C_i$ is minimal.
9      Add all successors of jobs in $I$ 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
```

Resource Leveling heuristic. We now describe the resource leveling heuristic by [MMS09], 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, while 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 $k^\star$ are set to $(\mathrm{UB}_{k^\star} + \mathrm{LB}_{k^\star})/2$, $\mathrm{UB}_{k^\star}$ and $\mathrm{LB}_{k^\star}$ being the upper and lower bounds on the capacity of resource $k^\star$, 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 $\mathrm{LB}_k$ and $\mathrm{UB}_k$ to initial values for each resource type $k \in \mathcal{R}$.
2 while $\exists\, k \in \mathcal{R} : \mathrm{LB}_k < \mathrm{UB}_k$ do
3     Choose resource type $k^\star \in \mathcal{R}$ with $\mathrm{LB}_{k^\star} < \mathrm{UB}_{k^\star}$ and $\mu_{k^\star}$ maximum.
4     Perform binary search using list scheduling and ready scheduling in
      order to decrease the capacity bounds of $k^\star$.
```

LP solution and ready scheduling heuristics. In 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 with jobs sorted by start completion times in the LP solution, 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.

5 Computational Study

5.1 Experimental Setup

As there is no set of instances which really suits our problem, we needed to compile our own. It is composed of three sets of job scenarios, with 10 instances each. Each job can run in 3 different modes, using 1 to 3 resources, with durations ranging from 5 to 12. In the first scenario instances have 30 jobs and 60 edges, and in the second and third scenario instances contain 50 jobs with 70 and 100 edges respectively. The maximal precedence graph width is 5 for scenario one, and 6 for the other two. These width bounds are achieved by constructing W chains of length $|\mathcal{J}|/N$, and randomly choosing the remaining edges.

There are two different resources which come in 5 calendar configurations, called C1 to C5. These calendars are described schematically in Fig. 2. In the

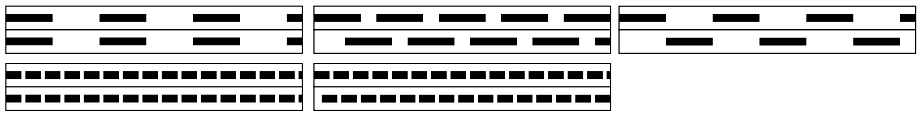
Figure 2. Calendar configurations C1–C3 (top) and C4 and C5 (bottom) used in our test set. Black bars symbolize the temporal location of shifts.

Table 1. Comparison of the BP approach with the standard MIP ($|\mathcal{J}| = 30$).

	Branch-and-Price				B&P w/o heur.				CPLEX				RL
Calendar	time	LB	UB	†	time	LB	UB	†	time	LB	UB	†	UB
C1	446	9	9	1	435	10	10	0	1350	6	3	7	6
C2	326	9	10	1	360	9	9	1	435	9	9	2	5
C3	9	10	10	0	11	10	10	0	1	10	10	0	9
C4	81	10	10	0	81	10	10	0	494	9	10	1	8
C5	72	10	10	0	146	10	10	0	1631	3	4	7	6
Total	187	48	49	2	207	49	49	1	782	37	36	17	34

Table 2. Comparison of the BP approach with the standard MIP ($|\mathcal{J}| = 50$).

	$|\mathcal{J}| = 50, |E| = 70$								
	Branch-and-Price				CPLEX				RL
Calendar	time	LB	UB	†	time	LB	UB	†	UB
C1	1660	10	9	7	1800	6	1	10	7
C2	1662	4	7	8	1361	7	6	7	5
C3	27	10	10	0	4	10	10	0	10
C4	719	10	10	2	1800	6	0	10	5
C5	1293	8	8	5	1335	7	6	7	5
Total	1072	42	44	22	1260	36	23	34	32
	$|\mathcal{J}| = 50, |E| = 100$								
Calendar	time	LB	UB	†	time	LB	UB	†	UB
C1	1137	9	10	3	1800	3	0	10	7
C2	1224	6	9	5	1622	5	4	9	6
C3	9	10	10	0	2	10	10	0	9
C4	254	10	10	0	1800	1	0	10	8
C5	365	10	10	0	1451	5	5	6	6
Total	598	45	49	8	1335	24	19	35	36

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 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, and shift breaks have length 20. Calendars C4 and C5 show shifts with length 20 and breaks having length 5. In C5 one of the resources is offset by 10. All scenarios are tested with each of the 5 different calendars. Time horizons were chosen by computing a minimal and maximal makespan heuristically using simple list scheduling, and averaging these.

Our algorithm was tested on an Intel 2.66 GHz processor, and each test run had a time-limit of 30 minutes. The implementation uses SCIP 1.2.0.6 [SCI], to perform the branch-and-bound process, with custom plug-ins for the heuristics, branching rules, and the pricer. For the standard MIP (1)–(6) we used CPLEX 12 with quad-core parallelization on a stronger machine.

Our empirical results can be seen in Tabs. 1 and 2. These tables show four columns per algorithm. The first column contains the average time in seconds to solve the instance with a limit of 1800. The second and third columns represent the number of times the algorithm reached the best known lower and upper bounds over all algorithms, denoted by "LB" and "UB." The fourth column is the number of timeouts, marked by "†." An additional last column "RL" per table shows the number of times the resource leveling heuristic reached the best known upper bound for each set of instances. The rows in each table represent the calendar configurations used.

In the first table, three different algorithms are compared on the first scenario. "Branch-and-price" refers to the general scheme presented here, while "B&P w/o heur." does not use the LP solution or ready scheduling heuristics during the branch-and-bound process. However, it does start with the solution found by the resource leveling heuristic. The "CPLEX" columns correspond to the results of the standard MIP (1)–(6). The "Branch-and-price" algorithm, as well as CPLEX and the resource leveling heuristic, are tested on the second and third scenarios in Tab. 2 respectively.

Impact of calendars. Obviously, the calendar choice has a big influence on running time. For 30 jobs we observe that with the one hour shifts the makespan generally increases as the resources overlap less. Interestingly, for the short shifts the makespan increases with more overlap (C4 instances generally have larger makespan than those in C5), yet these instances can nevertheless be solved faster. The "hardness" of a problem does not solely depend on the makespan though, as CPLEX shows the worst behavior on instances with 50 jobs and calendars C1 and C4, where the shifts are overlapping completely. This is not the case for the branch-and-price algorithm, which actually performs much better on C4 and C1, see also Tab. 2. All approaches easily solve instances using C3, because many jobs have a successor of a different resource, introducing long waiting times.

Usefulness of the heuristics. Our heuristics exploit problem specific knowledge during the branch-and-bound process, however, this does not appear to have a big impact on the number of problems solved, as can be seen in Tab. 1. Still, the average solving time decreases noticeably across all instances with the heuristics enabled. The running times of all our heuristics are negligible, and in contrast to the exact methods, the resource leveling heuristic is able to handle instances of practically relevant size, while still achieving the best found upper bound in 68% of all instances.

Influence of the network complexity. When the number $|E|$ of edges in G is increased, the makespan typically increases as well, so that the time-indexed MIP formulation grows fairly large and CPLEX generally fares better on the instances with less edges. For 50 jobs and 70 edges CPLEX finds the best lower bound 36 times and the best upper bound 23 times, in contrast to 24 best lower and 19 best upper bounds for 100 edges (Tab. 2). In contrast the branch-and-price does not suffer from this, and the computation times actually decrease on the instances with more edges.

As expected, CPLEX has considerable problems as the instances get larger. One of the main features of the turnaround scheduling problem is the presence of availability periods, which motivated this branch-and-price approach. It achieves the best results across all shift configurations, in the worst case 88% of the best upper bounds are found. In most cases lower bounds computed by CPLEX are of poor quality. Proving optimality succeeded in 80% of cases for Branch-and-

Price, whereas CPLEX only manages 43%. Summarizing, our branch-and-price algorithm significantly outperforms the standard time-indexed MIP formulation solved by the state-of-the-art solver CPLEX on large instances.

6 Summary

Our own experiments and instance sizes as reported e.g., in [DK01] or [PSP] for scheduling problems of comparable complexity give good reasons to believe that even today instances with only 30 jobs are hard to solve to optimality. We are encouraged to further investigate branch-and-price algorithms for this type of problem, as we do not only report better results on similarly sized instances, but are also able to optimally solve some instances with up to 50 jobs.

References

[BDM$^+$99] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European J. Oper. Res.*, 112:3–41, 1999.

[Dem95] E. Demeulemeester. Minimizing resource availability costs in time-limited project networks. *Management Sci.*, 41:1590–1598, 1995.

[DH02] E.L. Demeulemeester and W.S. Herroelen. *Project Scheduling: A Research Handbook*. Kluwer, 2002.

[DK01] A. Drexl and A. Kimms. Optimization guided lower and upper bounds for the resource investment problem. *The Journal of the Operational Research Society*, 52(3):340–351, 2001.

[FNS01] B. Franck, K. Neumann, and C. Schwindt. Project scheduling with calendars. *OR Spektrum*, 23:325–334, 2001.

[Har01] S. Hartmann. Project scheduling with multiple modes: A genetic algorithm. *Ann. Oper. Res.*, 102(1-4):111–135, 2001.

[HB09] S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European J. Oper. Res.*, In Press, 2009.

[MMS09] N. Megow, R.H. Möhring, and J. Schulz. Decision support and optimization in shutdown and turnaround scheduling. Preprint 009-2009, Institut für Mathematik, Technische Universität Berlin, 2009.

[Möh84] R.H. Möhring. Minimizing costs of resource requirements in project networks subject to a fixed completion time. *Oper. Res.*, 32(1):89–120, 1984.

[MVH08] L. Mercier and P. Van Hentenryck. Edge finding for cumulative scheduling. *INFORMS J. Computing*, 20(1):143–153, 2008.

[NSZ03] K. Neumann, C. Schwindt, and J. Zimmermann. *Project scheduling with time windows and scarce resources*. Springer, 2003.

[PSP] PSPLib. Project Scheduling Problem LIBrary. `http://129.187.106.231/psplib/`. Last accessed 2010/02/01.

[PWW69] A.A.B. Pritsker, L.J. Watters, and P.M. Wolfe. Multi project scheduling with limited resources: A zero-one programming approach. *Management Sci.*, 16:93–108, 1969.

[SCI] SCIP. Solving Constraint Integer Programs. `http://scip.zib.de/`.

[Zha92] J. Zhan. Calendarization of time planning in MPM networks. *ZOR – Methods and Models for Oper. Res.*, 36(5):423–438, 1992.