Linear programming based heuristics for multi-project capacity planning

Citation for published version (APA):

Document status and date:
Published: 01/01/2002

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Linear Programming Based Heuristics for Multi-Project Capacity Planning

Noud Gademann & Marco Schutten

WP-79
Linear Programming Based Heuristics for Multi-Project Capacity Planning

Noud Gademann and Marco Schutten*

Center for Production, Logistics and Operations Management
University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

and

ORTEC Consultants bv
P.O. Box 490, 2800 AL Gouda, The Netherlands

email addresses authors
A.J.R.M.Gademann@cplom.utwente.nl, ngademann@ortec.nl
J.M.J.Schutten@cplom.utwente.nl, mschutten@ortec.nl

telephone number: +31 53 489 4676
fax number: +31 53 489 2159

May 17, 2002

Abstract

Many multi-project organizations are capacity driven, which means that their operations are constrained by various scarce resources. An important planning aspect in a capacity driven multi-project organization is capacity planning. By capacity planning, we mean the problem of matching demand for resources and availability of resources for the medium term. Capacity planning is a very useful method to support important tactical decisions such as due date quotation and price quotation for new projects, and to get insight in capacity requirements for the medium term. We present a capacity planning model in which aspects such as capacity flexibility, precedence relations between work packages, and maximum work content per period can be taken into account. For this model, we discuss several LP based heuristics. Using a large set of test instances, we compare these heuristics with some results from literature. It turns out that some of these heuristics are very powerful for solving capacity planning problems.

---

*Corresponding author
1 Introduction

Many organizations use project management to manage their activities. Most of these organizations manage multiple projects simultaneously. Since the projects often use common resources, a multi-project organization is usually capacity driven. Good management of the scarce resources is of crucial importance for, e.g., reliable due date and price quotation and good delivery performance. Time is more and more a competitive edge, and a good control over, e.g., lead times requires adequate capacity management.

Platje, Seidel, and Wadman [10] stress that capacity in a multi-project organization cannot be managed in a traditional single project oriented approach. They describe an organizational structure to manage a portfolio of projects in a multi-project organization, and call this project based management. Management, project leaders, and resource managers together form the portfolio management team. They must make important resource allocation decisions. The portfolio management team plays a central role in project based management. De Boer [1] has developed a prototype decision support tool to support the decisions of the portfolio management team quantitatively. De Boer distinguishes two planning levels for portfolio management. The first level is known as rough-cut capacity planning (RCCP); the second level addresses the so-called resource constrained project scheduling problem (RCPSP). RCCP addresses medium-term capacity planning problems. At this level, projects are split up in relatively large work packages, which are planned over time taking into account the availability of scarce resources. The RCPSP addresses the operational, short term, scheduling. To that extend, work packages are split up into smaller activities which are scheduled over time. The usual objective is makespan minimization, constrained by the finite resource availability.

These two planning levels play a significant role in the portfolio management of multi-project organizations. They support medium term and short term resource management in a multi-project environment. Moreover, using the concept of larger work packages for RCCP, a top-down planning is facilitated. It is not necessary to make a detailed project breakdown and process planning to get insight in capacity requirements and capacity availability. RCCP supports important aspects like capacity requirements, the use of nonregular capacity ( subcontracting, working overtime, hiring), and order acceptance (due date and price quotation). Only for the short term planning a detailed process planning is required.

There is a close relation between a project life cycle and planning issues at the RCCP and RCPSP level. In a project life cycle, in general five project phases can be distinguished: Order acceptance, process planning, scheduling, execution, and evaluation and service. In the order acceptance phase, prices and delivery times must be quoted to the customers. After a project has been accepted, the project activities are specified in the process planning phase. Using the specified activities, a detailed project schedule can be made, after which the execution of the activities takes place. Finally, a project is evaluated and serviced. During the order acceptance phase and during the project scheduling phase, capacity management is important.

A vast amount of literature has been dedicated to project scheduling (see, e.g., Demeulemeester and Herroelen [3], Kolisch and Drex [6], and Özdemar and Ulusoy [9]). The standard problem that is being studied is the RCPSP. The literature dealing with the order acceptance phase is very limited. The decisions that are made in this phase, however, are very important, because the quoted project price and delivery time determine whether the organization will be the order winner and whether a project will be profitable. To quote a realistic price and a realizable delivery time requires a good insight in the available capacity in the forthcoming
months and in the required capacity for a new project. This asks for capacity planning using aggregate data on available and required capacity. Moreover, details about activities to be performed and resource requirements of these activities, may only become available during the process planning phase. This process planning phase can take up to a few months time, whereas the customer does not want to wait that long. This also suggests to use aggregate data on available and required capacity.

We assume that a project is broken down into a number of rather large work packages, which we call jobs. A job can be seen as an aggregation of possibly yet unknown activities that are related. In the process planning phase, the jobs are divided in activities. We assume that for each job estimations of the required resource capacity are available and precedence relations may exist between the jobs. We refer to this problem as the rough-cut capacity planning (RCCP) problem. Analogously to the variants that Möhring [8] distinguishes for the RCPSP, we distinguish two variants of the RCCP problem:

**Time driven RCCP.** In the time driven variant, a desired project delivery time must be met, i.e., it is considered a deadline. This may imply that nonregular capacity must be used, for example by temporarily hiring extra personnel, subcontracting jobs, and working overtime. The objective is to minimize the cost of using nonregular capacity.

**Resource driven RCCP.** In this variant, we can only use the capacity that is regularly available to the organization. We will refer to this capacity as regular capacity. The objective is to minimize the project delivery time.

In this paper, we will focus on time driven RCCP.

## 2 Problem description

The problem that we consider can be formally described as follows. We are given a set of \( n \) jobs \( J_1, J_2, \ldots, J_n \) that must be planned on \( K \) resources \( R_1, R_2, \ldots, R_K \). The jobs \( J_1, J_2, \ldots, J_n \) may belong to different projects. We assume that the time horizon is divided in \( T \) buckets of one week, say. The regular capacity of resource \( R_k \) \((k = 1, 2, \ldots, K)\) in week \( t \) is equal to \( Q_{kt} \) hours. Job \( J_j \) requires \( q_{jk} \) hours of resource \( R_k \).

Let \( x_{jkt} \) denote the fraction of job \( J_j \) that is performed on resource \( R_k \) in week \( t \). This means that \( x_{jkt} \cdot q_{jk} \) hours are spent on job \( J_j \) in week \( t \) on resource \( R_k \). We assume that \( x_{jkt} = x_{jlt} \ (k, l = 1, 2, \ldots, K) \), i.e., that an equal fraction is spent on job \( J_j \) in week \( t \) on all resources. In the remainder of this paper, we therefore denote the fraction of a job \( J_j \) that is performed in week \( t \) by \( x_{jt} \). We also assume that for each job \( J_j \) we are given a maximum fraction \( \frac{1}{p_j} \) that can be done per week. This means that a job \( J_j \) has a minimum duration of \( p_j \) weeks. A job \( J_j \) is completely performed in time interval \([t_1, t_2]\) if \( \sum_{t=t_1}^{t_2} x_{jt} = 1 \).

Job \( J_j \) must be performed in time window \([r_j, d_j]\), which results in \( x_{jt} = 0 \) for \( t < r_j \) and for \( t > d_j \). Precedence relations may exist between jobs. Suppose that a precedence relation exists between jobs \( J_i \) and \( J_j \), which we denote by \( J_i \rightarrow J_j \). This means that job \( J_i \) must be completely performed before we can start performing job \( J_j \), i.e., \( x_{j \tau} > 0 \Rightarrow \sum_{t=t_1}^{\tau-1} x_{ut} = 1 \) for \( 1 \leq \tau \leq T \). We assume that the time windows respect the precedence relations, i.e., \( r_j \geq r_i + p_i \) and \( d_i \leq d_j - p_j \) if \( J_i \rightarrow J_j \). If this is not the case, then we can alter the time windows to respect this via a simple forward-backward algorithm. Moreover, we assume that the (resulting) time windows are large enough to perform the jobs \((r_j + p_j - 1 \leq d_j \) for each
job $J_j$). If this is not the case for a job, then no solution exists in which this job is completely performed within its time window.

A solution $[x_{11}, x_{12}, \ldots, x_{1T}, x_{21}, x_{22}, \ldots, x_{2T}, \ldots, x_{n1}, x_{n2}, \ldots, x_{nT}]$ is feasible if each job is completely performed within its time window, respecting the precedence relations, and respecting the maximum fraction that can be done per week. A solution may imply that the required capacity of a resource $R_k$ in a week $t$ exceeds the regular availability $Q_{kt}$. In that case, we need $U_{kt} = \max\{0, \sum_{j=1}^{n} q_{jk} \cdot x_{jt} - Q_{kt}\}$ hours nonregular capacity of resource $R_k$ in week $t$. The cost of using $U_{kt}$ hours of nonregular capacity of resource $R_k$ in week $t$ is equal to $c_{kt} \cdot U_{kt}$. We assume that the availability of nonregular capacity is infinite and that $c_{kt} \geq 0, \forall k, t$. Our objective is to minimize the total cost of required nonregular capacity.

Figure 1 shows a feasible solution of the instance of the RCCP problem of which data can be found in Table 1. Jobs $J_2$ and $J_3$ can only start when job $J_1$ is completed and job

<table>
<thead>
<tr>
<th>Job</th>
<th>$J_r$</th>
<th>$r_j$</th>
<th>$d_j$</th>
<th>$p_j$</th>
<th>$q_{j1}$</th>
<th>$q_{j2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>60</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>$J_2$</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>80</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>$J_3$</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>70</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>$J_4$</td>
<td>4</td>
<td>8</td>
<td>1</td>
<td>60</td>
<td>150</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Data for example instance.
$J_4$ can only start when both job $J_2$ and job $J_3$ are completed. In this example, there are two resources. The regular availability of resource $R_1$ is 35 hours in every week, whereas the regular availability of resource $R_2$ is 60 hours in every week. In the solution in Figure 1, we use 5 hours of nonregular capacity on resource $R_1$ in weeks 3, 4, and 5, and 15 hours on resource $R_2$ in weeks 7 and 8. Note that some kind of preemption is allowed: 50% of job $J_2$ is done in weeks 3 and 5, whereas no work is done to this job in week 4. Also note that this solution is not an optimal solution (if $c_{14} > 0$), because a fraction of $5/70$ of job $J_3$ can be moved from week 4 to week 6. In the resulting solution, we would use less nonregular capacity on resource $R_1$ and the same amount of nonregular capacity on resource $R_2$.

If we ignore the precedence relations and assume that all costs $c_{kt}$ are non-negative, then the problem can be formulated as the following linear programming problem $(P)$:

$$\min \sum_{t=1}^{T} \sum_{k=1}^{K} c_{kt} \cdot U_{kt}$$

Subject to:

$$\sum_{t=r_j}^{d_j} x_{jt} = 1 \quad \forall j \quad (1)$$

$$x_{jt} \leq \frac{1}{p_j} \quad \forall j, t \quad (2)$$

$$U_{kt} \geq \sum_{j=1}^{n} q_{jk} \cdot x_{jt} - Q_{kt} \quad \forall k, t \quad (3)$$

$$x_{jt}, U_{kt} \geq 0 \quad \forall j, k, t \quad (4)$$

Equations (1) ensure that each job is performed completely within its time window and Equations (2) express that no more than the maximum fraction can be done in a week. Equations (3) guarantee the required amount of nonregular capacity.

If we solve this linear programming problem, then in general one or more precedence relations are violated, and thus the solution is not feasible. To control feasibility, we introduce an allowed to work (ATW) window for every job. An ATW window $[S_j, C_j]$ for job $J_j$ specifies the weeks in which we allow to work on job $J_j$. This means that we do not allow to work on job $J_j$ before week $S_j$ and also not later than week $C_j$. Associated with each set $S$ of ATW windows, we consider the linear programming problem $P_S$. $P_S$ is obtained from the linear programming problem $P$ by replacing constraints (2) with

$$x_{jt} \leq \frac{s_{jt}}{p_j} \quad \forall j, t, \quad (5)$$

where parameter $s_{jt}$ indicates whether processing of job $J_j$ is allowed in week $t$:

$$s_{jt} = \begin{cases} 1 & \text{if } S_j \leq t \leq C_j; \\ 0 & \text{otherwise}. \end{cases}$$

We call an ATW window $[S_j, C_j]$ for job $J_j$ feasible, if:

1. $S_j \geq r_j$ and $C_j \leq d_j$;
2. $C_j - S_j \geq p_j - 1$;
Let $S$ be a set of ATW windows, with one ATW window $[S_j, C_j]$ for every job $J_j$. We call $S$ feasible, if every ATW window in $S$ is feasible and moreover:

3. $S_j > C_i$ if $J_i \rightarrow J_j$ ($\forall i, j$).

The last condition ensures that the precedence relations are met. Clearly, if $S$ is feasible, then any feasible solution to $P_S$ is feasible for the RCCP problem.

In the remainder of this paper, we investigate heuristics based on the linear programming problem $P_S$. Note that a feasible solution of any heuristic implies a starting time $S_j$ for all jobs $J_j$. Based on these job starting times and the precedence relations between the jobs, we can compute a maximum allowed completion time $C_j$ for job $J_j$. These starting times and maximum allowed completion times induce a feasible set $S$ of ATW windows. By solving the associated linear programming problem $P_S$, we get a feasible solution which is at least as good as the heuristic solution. This way an improvement step can easily be added to any heuristic.

3 Related work

De Boer and Schutten [2] propose several heuristic algorithms for the RCCP problem in which they try to minimize the total number of hours of nonregular capacity that is used, i.e., $c_{kt} = 1$ ($\forall k, t$). Their heuristic ICPA (incremental capacity planning algorithm) is a constructive heuristic that plans jobs one by one in two phases. First, a maximum part of a job is planned in its time window, without using nonregular capacity for this job. If the job cannot be planned completely, capacity is increased in the second phase such that the remainder of the job fits in its time window. De Boer and Schutten test two variants of ICPA, which result from different criteria for the order in which the jobs are planned. De Boer and Schutten measure the performance of ICPA by the deviation of the solution value to a specific lower bound. If we apply the improvement step mentioned at the end of Section 2 to ICPA, on average the reduction in the deviation from that lower bound is about 15%.

Another type of heuristics proposed by De Boer and Schutten is based on problem $P$. After solving problem $P$, it appears in general that one or more precedence relations are violated. Suppose that relation $J_i \rightarrow J_j$ is violated. Hence, intervals $[r_i, d_i]$ and $[r_j, d_j]$ overlap. The precedence relation between job $J_i$ and job $J_j$ is 'repaired' by specifying a week $T_{ij}$ ($r_i + p_i \leq T_{ij} \leq d_j - p_j + 1$) before which job $J_i$ must be finished and before which job $J_j$ cannot start. The variants of this heuristic arise from different rules for specifying week $T_{ij}$. It appears that the rule in which the ratio $(T_{ij} - r_i)/(d_j - T_{ij} + 1)$ is as close as possible to the ratio $W_i/W_j$ results in the best heuristic, in which $W_i$ is the total work content of job $J_i$. After specifying a week $T_{ij}$, $d_i$ is set to $T_{ij} - 1$, $r_j$ is set to $T_{ij}$, and the time windows of the other jobs are updated. A successor $J_k$ of job $J_j$, for example, cannot start before $r_j + p_j$, and since $r_j$ is updated, $r_k$ may have to be updated as well. For more details on this heuristic, we refer to Section 4.2.

Hans et al. [4] consider the so-called resource loading problem. The resource loading problem can be seen as an RCCP problem with simple precedence constraints, where the project network is a chain. They present a hybrid model of the resource driven and the capacity driven variant. In this model, the total cost of both non-regular capacity usage (subcontracting, hiring, working overtime) and tardiness penalties is minimized. They formulate this problem as a mixed integer programming problem with an exponential number of variables.
and present an exact branch and price algorithm to solve it. They also suggest several primal and improvement heuristics for the problem. Test results show that small to moderate size instances can be solved to optimality in practical time. In combination with some of the heuristics, a truncated branch-and-price method gives good results for larger instances too.

In Hans et al. [5] this model is extended, by allowing generalized precedence relations as in our RCCP problem description. This in particular affects the pricing problem for the branch and price algorithm. Whereas the pricing problem for the resource loading problem can be solved by a polynomial time dynamic programming algorithm, the pricing problem for the RCCP problem becomes hard. This is in particular due to a multi-dimensional state space in case of generalized precedence relations. Computational experiments show that the RCCP problem can be solved to optimality for smaller instances, but for larger instances in particular the pricing problems become too large. Several other approaches are suggested and tested, like pricing heuristics and an integer programming formulation of the pricing problem that is solved directly by CPLEX.

4 Heuristics

In this paper we focus on heuristics for the RCCP problem. We distinguish three categories of solution approaches:

1. constructive heuristics;
2. heuristics that start with infeasible solutions and convert these to feasible solutions;
3. heuristics that improve feasible solutions.

De Boer and Schutten [2] propose heuristics such as ICPA that belong to the first two categories. We propose a heuristic that belongs to the second category and a number of heuristics that belong to the third category. The latter use dual information about solutions for problem $P_S$ to find better solutions. Since heuristics in the third category must start with a feasible solution, we also describe two basic primal heuristics (category 1).

4.1 Category 1 heuristics

A feasible set of ATW windows $S$ can easily be constructed by setting $S_j$ equal to $r_j$ and setting $C_j$ as large as possible: $C_j = \min \{d_j, \min_k | J_j \rightarrow J_k r_k - 1 \}$. A feasible solution is then found by solving $P_S$. We will refer to this basic primal heuristic as $H_{BASIC}$.

Another constructive heuristic is obtained by dividing the slack of jobs equally on the critical path. Define the slack $L_j$ of job $J_j$ as $L_j = C_j - (S_j + p_j)$. An ordered set of jobs $\{J_{i1}, J_{i2}, \ldots, J_{ik} \}$ is called a path if $J_{i1} \rightarrow J_{i2}, J_{i2} \rightarrow J_{i3}, \ldots, J_{ik-1} \rightarrow J_{ik}$. A critical path $\{J_{j1}, J_{j2}, \ldots, J_{jR} \}$ is a path for which holds that:

- $L_{j1} = L_{j2} = \ldots = L_{jR} = \min_{1 \leq j \leq n} L_j$;
- $S_{ji} = S_{ji-1} + p_{ji-1}$ for $2 \leq i \leq R$;
- $C_{ji} = C_{ji+1} - p_{ji+1}$ for $1 \leq i \leq R - 1$. 
A critical path \( \{J_1, J_2, \ldots, J_{JR}\} \) is a \textit{maximal critical path} if there is no job \( J_k \) such that either \( \{J_k, J_1, J_2, \ldots, J_{JR}\} \) or \( \{J_1, J_2, \ldots, J_{JR}, J_k\} \) is a critical path. We initialize the heuristic by setting \( S_j = r_j \) and \( C_j = d_j \) for every job \( J_j \). Note that this does not need to result in a feasible set of ATW windows. Next, we find a maximal critical path, \( \{J_{j1}, J_{j2}, \ldots, J_{jR}\} \), say. The \textit{total slack} \( \bar{L} \) of this path is defined as \( C_{jR} - (S_{j1} + \sum_{i=1}^{R} p_{ji}) \). The total slack is distributed over the maximal path by changing the ATW windows for these jobs by first setting \( S_{ji} = S_{j1} + \sum_{k=1}^{i-1} p_{jk} + |\bar{L} - \sum_{k=1}^{i-1} p_{jk} / \sum_{k=1}^{R} p_{jk}| \) for \( 1 < i \leq R \) and then \( C_{ji} = S_{ji+1} - 1 \) for \( 1 \leq i < R \), where \( |x| \) denotes the integer value we get by rounding \( x \). The ATW windows of the jobs that do not belong to the maximal critical path are changed (if necessary) so that they meet the precedence relations with jobs that belong to the maximal critical path, \( i.e., S_i + p_i \leq S_j \) and \( C_i \leq C_j - p_j \) if \( J_i \rightarrow J_j \). The jobs of the maximal critical path are then removed from the network and a new maximal critical path is found in the resulting network. The total slack of this path is distributed and so on, until we have no more jobs left. Note that the resulting set of ATW window is a feasible set of ATW windows. We will refer to this heuristic as \( H_{CPM} \).

4.2 Category 2 heuristics

De Boer and Schutten [2] suggest a number of heuristics based on problem \( P \). A solution to problem \( P \) may violate one or more precedence relations. The suggested heuristics are based on repairing violated precedence relations one by one. If the precedence relation \( J_i \rightarrow J_j \) is violated, then a week \( T_{ij} \) is specified to repair the relation as follows. \( J_i \) must complete before time period \( T_{ij} \); \( J_j \) cannot start before \( T_{ij} \). De Boer and Schutten suggest several rules to determine \( T_{ij} \). They first find the job \( J_i \) with lowest index \( i \) for which a precedence relation has been violated. Next, they find job \( J_j \) with the lowest index \( j \) for which precedence relation \( J_i \rightarrow J_j \) has been violated. Then, they repair \( J_i \rightarrow J_j \) by setting \( T_{ij} \). They use several rules to set \( T_{ij} \), of which a rule based on work content worked best. We suggest a similar approach. The main differences are the criterion to select a violated precedence relation \( J_i \rightarrow J_j \) and the rule to determine \( T_{ij} \). We first find the precedence relation \( J_k \rightarrow J_l \) that is violated and has the minimum slack \( S_{kl} \). \( S_{kl} \) is defined as \( d_l - r_k - (p_l + p_k) \). The idea is that we have little freedom for specifying \( T_{kl} \) for repairing the precedence relation \( J_k \rightarrow J_l \) and we should use the little freedom as good as possible. Therefore, instead of using a simple rule to determine \( T_{kl} \), we suggest to evaluate all possible values of \( T_{kl} \), that is, \( r_l \leq T_{kl} \leq d_k + 1 \), and keep the best one. Moreover, by decreasing \( d_k \) for job \( J_k \), it may be possible to decrease \( r_m \) for successors \( J_m \) of job \( J_k \) without violating more precedence constraints. If this is possible, then we do so. Analogously, we investigate whether we can increase \( d_m \) for predecessors of job \( J_i \). It appears that this strategy outperforms the heuristics of De Boer and Schutten, both in solution quality and required computation time (see Section 5). We denote this heuristic by \( H_{ENUM} \).

In order to validate our idea that the precedence relation with minimum slack should be repaired first, we also tested a variant of this heuristic that first repairs the precedence constraint which has the maximum slack. This heuristic appears to find slightly worse solutions and requires more computation time.
4.3 Category 3 heuristics

4.3.1 Local Search to improve a feasible solution

The LP based heuristics that we considered so far, start with a solution in which precedence relations may be violated. The precedence relations are then repaired, until a feasible solution results in which no precedence relation is violated.

Once a feasible solution is available, we may apply a local search heuristic that tries to improve the current feasible solution. Suppose that we have a feasible set of ATW windows $S$ and want to improve the corresponding solution, which is optimal for problem $P_S$. This can only be achieved by changing the ATW windows for the jobs. The basis for a local search heuristic is the neighborhood structure. Suppose that we want to change the ATW window $[S_j, C_j]$ for job $J_j$. We define a neighborhood based on increasing or decreasing either $S_j$ or $C_j$ for a job $J_j$ by 1. For a job $J_j$, this gives four possible changes. As an example, we discuss the case in which we want to enlarge ATW window $[S_j, C_j]$ by decreasing $S_j$ by one time period. The first condition that must hold is that $S_j > r_j$, since otherwise the resulting ATW window would not be feasible. Let job $J_k$ be a predecessor of job $J_j$ and suppose that $C_k = S_j - 1$. This means that the ATW windows of $J_j$ and $J_k$ are adjacent. So, if we decrease $S_j$, then, to maintain feasibility of $S$, we have to decrease $C_k$ as well. This is only possible if the ATW window $[S_k, C_k - 1]$ consists of at least $p_k$ time periods. Therefore, the second condition that must hold is that $C_k - S_k \geq p_k$ for all predecessors $J_k$ of job $J_j$ with $C_k = S_j - 1$.

If the two conditions hold, then it is possible to decrease $S_j$ by one time period. Let $S_j'$ represent the new value of $S_j$, i.e., $S_j' = S_j - 1$. To represent the new ATW window $[S_j', C_j]$ for job $J_j$ in problem $P_S$, the right-hand side of constraint (5) for $t = S_j'$ must be changed from 0 to $\frac{1}{P_j}$. If we have to decrease $C_k$ for a predecessor of job $J_j$ as well, it may be possible to enlarge the time windows for successors $J_l$ of job $J_k$ without losing feasibility of the ATW windows by decreasing $S_l$. If this is possible, we will do this. Denote the resulting set of ATW windows by $S'$.

Similar conditions and changes in problem $P_S$ can be derived for the other possible changes of the ATW window $[S_j, C_j]$.

For a feasible set $S$, we define the neighbors of $S$ as all feasible sets $S'$ that can be obtained from one of the four possible changes to an ATW window $[S_j, C_j]$ in $S$. Note that the resulting search space is connected.

Starting from any feasible set $S$, we can apply local search using this neighborhood to look for improvements to the corresponding solution. We will discuss several local search heuristics, in which we vary:

1. The initial feasible set $S$;
2. The neighbors of the current feasible set $S$ that are evaluated;
3. The acceptance criterion.

In particular with respect to the selection of the neighbors to be evaluated, we use information from linear programming theory.
4.3.2 Linear programming theory

Every feasible solution is characterized by a feasible set $S$ of ATW windows. The corresponding solution is found by solving the associated LP problem $P_S$. As described in Section 4.3.1, a neighbor $S'$ of $S$ is obtained by changing the upper bound or lower bound of one of the ATW windows by one, and making the necessary changes to related ATW bounds. The corresponding LP problem $P_{S'}$ differs only slightly from $P_S$, namely for the ATW bounds that are changed, the right hand sides for the corresponding constraints in the LP problem also change. From standard sensitivity analysis in linear programming theory, it is known that a change to the right hand sides has no consequences for the optimality of the current basis, provided that the current basis remains feasible. If $B_S$ denotes the basis matrix and $b_S$ the right hand side for LP problem $P_S$, then the same basis remains feasible (and thus optimal) for $S'$ if:

$$B_S^{-1}b_{S'} \geq 0$$

For the moment, we assume that the same basis remains optimal for neighbor $S$ of $S'$. Let $\Pi_S$ denote the optimal dual solution and $z_S$ the optimal objective function value for $P_S$, then:

$$z_{S'} = \Pi_S b_{S'}$$

Let $\pi_j(t)$ be the value of the dual variable associated with constraint (5) for job $J_j$ in week $t$ in the optimal solution for problem $P_S$. Furthermore, let $\Delta(S, S')$ denote the set of indices of the constraints for which the right hand sides for $S$ and $S'$ differ, and, for $i \in \Delta(S, S')$ let $\Delta b_i$ denote the corresponding difference in the right hand side, then:

$$z_{S'} = z_S + \Delta z(S, S')$$

with

$$\Delta z(S, S') = \sum_{i \in \Delta(S, S')} \pi_i \Delta b_i$$

Thus, under the assumption that the same basis remains optimal, we can use the optimal dual solution $\Pi_S$ for $P_S$ to compute the change in objective function if we move from $S$ to $S'$.

Moving to a neighbor $S'$ of $S$ will often bring about changes in more than one right hand side. This makes it hard to predict whether the same basis will remain optimal. On the other hand, $\Delta z(S, S')$ may give a good indication of the impact on the optimal objective function. Since $\Delta z(S, S')$ can easily be computed, we may use it to decide which neighbors to evaluate in the local search.

In the example above, where we changed $S_j$ to $S_j - 1$, to get a feasible set of ATW windows we have to change $s_{kC_k}$ from 1 to 0 for predecessors $J_k$ of job $J_j$ for which holds that $C_k = S_j - 1 = S_j'$. Therefore in this example the (expected) total change is given by

$$\Delta z(S, S') = \pi_j(S_j) - \sum_{J_k \in \mathcal{P}'_j} \pi_k(S_j),$$

in which $\mathcal{P}'_j$ is the set of predecessors $J_k$ of job $J_j$ with $C_k = S_j - 1$.

The other three options for changing an ATW window (increasing $S_j$, increasing $C_j$, or decreasing $C_j$) are evaluated analogously.
4.4 $H_{FEAS()}$

Empirically, it has been shown that a steepest descent step in the Simplex Method for linear programming works very well. We adopt this result in evaluating the neighbors of a set $S$. In heuristic $H_{FEAS(BASIC)}$, we generate an initial feasible set $S$ by heuristic $H_{BASIC}$. Next, we order all neighbors of $S$ by increasing value of $\Delta_x(S, S')$. We try the neighbors in this order and accept the first neighbor that leads to an improved schedule. Then the local search is continued for the neighbors of $S'$. The heuristics stops when no more improvement is found.

Since we accept only improvements, we limit our search space significantly. We may expect that the final solution in that case is sensitive to the initial feasible solution. To test the influence of the starting solution to the final solution, we also implemented heuristic $H_{CPM}$ for finding a starting solution. Apart from the starting solution, heuristic $H_{FEAS(CPM)}$ is identical to heuristic $H_{FEAS(BASIC)}$.

In the heuristics $H_{FEAS(BASIC)}$ and $H_{FEAS(CPM)}$, we use the dual values of constraint $(5)$ to get insight in whether enlarging or shrinking the ATW window for a job may be advantageous. If, for example, we want to enlarge the ATW window for job $J_j$ by allowing processing at time $C_j + 1$, then we expect a change in the optimal solution value equal to $\pi_j(C_j + 1)$. We know, however, that $x_{ij,C_j+1} \leq \frac{1}{p_j}$, due to constraint $(2)$. We also used $\frac{1}{p_j} \cdot \pi_j(C_j + 1)$ as the expected change in the optimal solution value by allowing processing of job $J_j$ at time $C_j + 1$. In the same way, we use $-\frac{1}{p_j} \cdot \pi_j(T)$ as the expected change in the optimal solution value for disallowing processing of job $J_j$ at time $T$. If we use this method for the expected change, then we will refer to it by saying that we use ‘detailed shadow prices’.

5 Computational experiments

5.1 Instance generation

To test the quality of our heuristics, we used the set of 450 test instances of De Boer and Schutten [2]. An instance in this set is characterized by the three parameters $n$ (the number of jobs), $K$ (the number of resources), and $\phi$ (average float). The average float is defined as $\phi = \frac{\sum_{j=1}^{n} f_j}{n}$, where $f_j = d_j - (r_j + p_j) + 1$ is the float of job $J_j$. The used values for parameter $n$ were 10, 20, and 50; for parameter $K$ 3, 10, and 20; for parameter $\phi$ 2, 5, 10, 15, and 20. For all possible combinations of $n$, $K$, and $\phi$, 10 instances have been generated, which results in a total of 450 instances.

To generate precedence relations between the jobs, the network generation procedure of Kolisch et al. [7] has been used. Assume that we already have a set of $n$ jobs $J_1, J_2, \ldots, J_n$. The first step of the network generation procedure is to determine start jobs (jobs without predecessors) and end jobs (jobs without successors). The second step assigns to each job that is not a start job a predecessor; the third step assigns a successor to each job that is not an end job and that has no successor yet. In the last step, the procedure randomly assigns predecessors to jobs until a given average number of precedence relations per job is reached. Each instance in our test set has three start jobs, three end jobs, and the average number of precedence relations per job is equal to two.

The minimum duration $p_j$ of job $J_j$ is a randomly drawn integer from the interval $[1, 5]$. De Boer and Schutten developed a procedure to set $r_j$ and $d_j$ in such a way that $0 \leq f_j \leq 2\phi$ and that $\phi$ is equal to a specified value. For further details about this procedure, we refer to
De Boer and Schutten [2]. The time horizon $T$ is set to the maximum possible completion time, i.e., $T = \max_{1 \leq j \leq n} d_j$.

The available capacity $Q_{kt}$ for resource $R_k$ in week $t$ has been randomly drawn between 0 and 20. Each job requires at least one and at most 5 resources (randomly drawn). The values for the resource requirements $q_{jk}$ are randomly drawn such that the expected workload of a resource is about 80%.

De Boer and Schutten try to minimize the total number of hours of nonregular capacity. Therefore, $c_{kt} = 1$ for every resource $R_k$ in every week $t$.

### 5.2 Lower bounds

To get an impression how good the heuristics perform, we can compare them to each other and to a lower bound to the optimal solution value. De Boer and Schutten [2] use the solution value for problem $P$ in which all precedence relations are ignored. We denote this lower bound as $lb_1$. We improve this lower bound in the following way. Suppose that there is a precedence relation $J_i \rightarrow J_j$ and that the intervals $[r_i, d_i]$ and $[r_j, d_j]$ overlap. Let $z_{ij}(T_{ij})$ be the optimal solution value of problem $P$ in which we replace $d_i$ by $T_{ij} - 1$ and $r_j$ by $T_{ij}$. In every feasible solution (including optimal solutions), we can identify a week $T_{ij}$ ($r_i + p_i \leq T_{ij} \leq d_j - p_j + 1$) before which job $J_i$ is completely performed and before which no work is done to job $J_j$. Therefore, $LB_{ij} = \min\{z_{ij}(T_{ij})| r_i + p_i \leq T_{ij} \leq d_j - p_j + 1\}$ is a lower bound to the optimal solution value. We obtain a lower bound $lb_2 = \max\{LB_{ij}|(i,j) \in P\}$, with $(i,j) \in P$ if and only if there is a precedence relation $J_i \rightarrow J_j$ between job $J_i$ and job $J_j$. Note that, by definition, $lb_2 \geq lb_1$.

### 5.3 Computational results

The average value of the lower bound $lb_1$ used by De Boer and Schutten [2] is 939.7. The lower bound $lb_2$ that we developed has an average value of 981.0. This means an improvement of about 4.5% of the average value. The average relative improvement per instance, on the other hand, is more than 22%. Table 2 shows the results of the different heuristics for the 450 test instances.

<table>
<thead>
<tr>
<th></th>
<th>$H_{WC}$</th>
<th>$H_{ENUM}$</th>
<th>$H_{FEAS(BASIC)}$</th>
<th>$H_{FEAS(CPM)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min slack</td>
<td>Max slack</td>
<td>No detail</td>
<td>Detail</td>
</tr>
<tr>
<td>value</td>
<td>1470.4</td>
<td>1307.3</td>
<td>1314.4</td>
<td>1402.5</td>
</tr>
<tr>
<td>dev. (%)</td>
<td>8.80</td>
<td>5.24</td>
<td>5.39</td>
<td>7.88</td>
</tr>
<tr>
<td>max dev (%)</td>
<td>27.63</td>
<td>15.71</td>
<td>16.13</td>
<td>25.80</td>
</tr>
<tr>
<td>time (s)</td>
<td>20¹</td>
<td>13</td>
<td>21</td>
<td>4</td>
</tr>
<tr>
<td>max time (s)</td>
<td>193</td>
<td>237</td>
<td>459</td>
<td>113</td>
</tr>
<tr>
<td># best</td>
<td>6</td>
<td>215</td>
<td>143</td>
<td>56</td>
</tr>
<tr>
<td># unique best</td>
<td>0</td>
<td>151</td>
<td>92</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2: Computational results for test instances.

¹The experiments for heuristic $H_{WC}$ are performed on a slower computer, using older versions of both the source code compiler and the CPLEX optimizer than the other heuristics. The computation time of $H_{WC}$ on the one hand and the other heuristics on the other hand are therefore not directly comparable.
instances. The row ‘value’ denotes the average value of the objective function. Analogously to De Boer and Schutten, we use

\[ \text{dev} = \frac{\sum_{t=1}^{T} \sum_{k=1}^{K} U_{kt} + \sum_{t=1}^{T} \sum_{k=1}^{K} Q_{kt} - LB}{LB} \]

as an indication of the quality of a heuristic. It measures the deviation of the total capacity used \((\sum_{t=1}^{T} \sum_{k=1}^{K} U_{kt} + \sum_{t=1}^{T} \sum_{k=1}^{K} Q_{kt})\) from a lower bound \(LB\) to the total capacity required. For lower bound \(LB\), we use \(LB = lb_2 + \sum_{t=1}^{T} \sum_{k=1}^{K} Q_{kt}\) (recall that \(lb_2\) is a lower bound to the total number of hours of nonregular capacity that must be used; \(\sum_{t=1}^{T} \sum_{k=1}^{K} Q_{kt}\) is the total number of hours of regularly available capacity). The row ‘dev’ shows the average deviation, whereas row ‘max dev’ shows the maximum deviation. In the same way, ‘time’ and ‘max time’ indicate the average and the maximum required computation time (in seconds). The row ‘# best’ shows the number of instances for which a heuristic found the best solution value. The row ‘# unique best’ indicates the number of instances for which a heuristic found the best solution value and all other heuristics found a worse solution value. The column ‘HWOC’ shows the results for the best heuristic of De Boer and Schutten [2]. In this heuristic, the precedence relations are repaired using the work content ratio (see Section 4). The columns ‘Min slack’ and ‘Max slack’ indicate the order in which the precedence relations are repaired in heuristic \(H_{ENUM}\). The columns ‘Detail’ and ‘No detail’ refer to the use of detailed shadow prices or not.

In terms of solution values, all new heuristics appear to perform better than \(H_{WC}\). Heuristic \(H_{ENUM}\) using the minimum slack repairing rule results in the best average solution value using on average 13 seconds of computation time. We conclude that the order in which precedence relations are repaired influences the solution value for \(H_{ENUM}\). First repairing the precedence relation that has the minimum slack results in better results than first repairing the precedence relation that has the maximum slack. Moreover, using the maximum slack order instead of the minimum slack order results in larger computation times (both on average and in the maximum). We also see that, when \(H_{ENUM}\) results in the best solution value found, it is in about 70% of the cases the only algorithm that finds this solution value.

The results indicate that the starting solution for heuristic \(H_{FEASC}\) strongly affects the quality of the final solution. In general, heuristic \(H_{FEAS(CPM)}\) produces solutions that are slightly worse than \(H_{ENUM}\) does. The average solution value for heuristic \(H_{FEAS(BASIC)}\), however, is substantially larger. The influence of using detailed shadow prices is only marginal on the average solution value and on the average computation time. The maximum of the required computation times, however, differs significantly as a result of using detailed shadow prices. The average values of the starting solutions are 1645.5 for heuristic \(H_{BASIC}\) and 1479.4 for heuristic \(H_{CPM}\).

Since Table 2 suggests that the starting solution affects the quality of heuristic \(H_{FEASC}\), we tried \(H_{ENUM}\) (with the minimum slack ordering) as a starting solution for \(H_{FEASC}\), where we did not use detailed shadow prices. The average solution value of this combination is 1293.3, requiring on average a computation time of 16 seconds. In 308 (out of 450) instances, the solution of \(H_{ENUM}\) was improved by the combination. In 313 instances, the combination resulted in a solution which was at least as good as the best found solution so far, of which in 194 cases the best found solution value was improved.

To further increase the quality of the solutions found by heuristic \(H_{FEASC}\), we explored two more options. The first option is by randomly disturbing the starting solution. The second option we tried is randomly disturbing the end solution found by \(H_{FEASC}\) and using
that disturbed solution as a starting solution for the next iteration. To randomly disturb a (start or end) solution, we used the following procedure. First, we enumerate all possibilities to change the ATW window of one job \( J_j \) by increasing or decreasing \( S_j \) or \( C_j \) one time unit. Of all those possibilities, we randomly pick one. A next question is how many of such disturbances should be applied to a solution. On the one hand, the number of disturbances should not be that large, since the quality of the solution generally decreases as a result of disturbing the ATW windows and the quality of \( H_{FEAS(\cdot)} \) appears to depend on the quality of the starting solution. On the other hand, the number of disturbances should be large enough to be able to escape from the current local optimum. Experimentally, we found that the number of such disturbances equal to \( n \) (the number of jobs) divided by 4 works good.

Figure 2 displays the results we found for randomly disturbing either the initial solution from \( H_{FEAS(\cdot)} \) or the final solution of the improvement heuristics. The line for ‘CPM(30,1)’ gives the results for the 30 experiments we did by randomly disturbing \( H_{CPM} \). Experiment 1 uses \( H_{CPM} \) without disturbing it. Experiments 2 up to 30 use randomly disturbed versions of \( H_{CPM} \) as a starting solution. The \( i \)th data point on the line displays the total computation time for the first \( i \) experiments and the best solution value found in these experiments. The line for ‘CPM(30,5)’ displays results for the experiments in which we additionally disturb the end solution of an experiment 4 times (resulting in 5 solutions per experiment). The lines for ‘Enum(30,1)’ and ‘Enum(30,5)’ display the results for analogous experiments using \( H_{ENUM} \) as a starting solution. We see that this approach improves the solution quality considerably and that disturbing an end solution is more effective than disturbing the starting solution: one experiment for ‘CPM(30,5)’ and ‘Enum(30,5)’ requires less computation time and produces better solutions than five experiments for ‘CPM(30,1)’ and ‘Enum(30,1)’, respectively.

Hans et al. [5] test their exact algorithms on the same 450 instances. By using exact methods, they were able to find optimal solutions for a subset of the instances. Table 3 shows
detailed results on the number of times a solution was proven optimal, depending on \( n \) (the number of jobs), \( K \) (the number of resources), and \( \phi \) (the average float). The rows with label ‘Best HGVZ’ contain the number of times that at least one of the algorithms of Hans et al. found an optimal solution. Rows with label ‘CPM(30,5)’ indicate how many times heuristic \( H_{FEAS(CPM)} \) starting 30 times with a randomly disturbed version of \( H_{CPM} \) and disturbing the end solution 4 times. The number presented in those rows cannot be larger than those in ‘Best HGVZ’, since we do not know the optimal solution for those instances that were not solved exactly by Hans et al. In the same way, rows with label ‘Enum(30,5)’ present results for the same heuristic that only uses \( H_{ENUM} \) as a starting solution. Rows ‘Best GS’ present the numbers for the best of the latter two heuristics. We see that Hans et al. were particularly able to solve instances with a small number of jobs or with a small average float. In total, they were able to solve 126 case optimally. The heuristic based on \( H_{ENUM} \) found in 113 cases also an optimal solution, whereas the best algorithm found an optimal solution in 124 cases.

Table 3 shows detailed results on the average solution values, depending on \( n \), \( K \), and \( \phi \). Again we see that the best solution of the algorithms of Hans et al. produce solutions comparable to our solutions for instances with a small number of jobs and a small average float. For instances with a large number of jobs of with a large average float, our solutions are generally better (up to 35% for instances with an average float of 20).
Table 4: Detailed solution values.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>( n )</th>
<th>3</th>
<th>10</th>
<th>20</th>
<th>3</th>
<th>10</th>
<th>20</th>
<th>3</th>
<th>10</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPM(30,5)</td>
<td>2</td>
<td>108.26</td>
<td>770.94</td>
<td>2262.05</td>
<td>137.09</td>
<td>905.13</td>
<td>2479.08</td>
<td>216.70</td>
<td>945.43</td>
<td>3082.76</td>
</tr>
<tr>
<td>Enum(30,5)</td>
<td>108.26</td>
<td>773.48</td>
<td>2262.61</td>
<td>137.09</td>
<td>905.13</td>
<td>2479.08</td>
<td>216.70</td>
<td>943.35</td>
<td>3082.52</td>
<td></td>
</tr>
<tr>
<td>Best GS</td>
<td>108.26</td>
<td>770.46</td>
<td>2262.05</td>
<td>137.09</td>
<td>905.13</td>
<td>2479.08</td>
<td>216.70</td>
<td>943.19</td>
<td>3082.52</td>
<td></td>
</tr>
<tr>
<td>Best HGVZ</td>
<td>108.26</td>
<td>770.46</td>
<td>2262.05</td>
<td>137.09</td>
<td>905.13</td>
<td>2479.08</td>
<td>216.70</td>
<td>943.26</td>
<td>3085.68</td>
<td></td>
</tr>
<tr>
<td>CPM(30,5)</td>
<td>5</td>
<td>102.22</td>
<td>709.11</td>
<td>1942.90</td>
<td>124.65</td>
<td>814.89</td>
<td>2730.01</td>
<td>180.68</td>
<td>667.31</td>
<td>2814.62</td>
</tr>
<tr>
<td>Enum(30,5)</td>
<td>102.08</td>
<td>709.72</td>
<td>1945.62</td>
<td>123.91</td>
<td>812.60</td>
<td>2726.93</td>
<td>179.54</td>
<td>662.80</td>
<td>2811.54</td>
<td></td>
</tr>
<tr>
<td>Best GS</td>
<td>102.04</td>
<td>708.81</td>
<td>1941.73</td>
<td>123.91</td>
<td>811.34</td>
<td>2726.13</td>
<td>179.54</td>
<td>662.70</td>
<td>2810.01</td>
<td></td>
</tr>
<tr>
<td>Best HGVZ</td>
<td>102.04</td>
<td>708.81</td>
<td>1939.49</td>
<td>123.03</td>
<td>816.45</td>
<td>2725.55</td>
<td>188.11</td>
<td>691.27</td>
<td>2832.19</td>
<td></td>
</tr>
<tr>
<td>CPM(30,5)</td>
<td>10</td>
<td>108.97</td>
<td>930.44</td>
<td>2617.07</td>
<td>106.03</td>
<td>1031.75</td>
<td>2936.80</td>
<td>87.93</td>
<td>743.64</td>
<td>2371.26</td>
</tr>
<tr>
<td>Enum(30,5)</td>
<td>108.29</td>
<td>927.23</td>
<td>2619.31</td>
<td>102.91</td>
<td>1028.49</td>
<td>2945.38</td>
<td>82.97</td>
<td>730.96</td>
<td>2359.40</td>
<td></td>
</tr>
<tr>
<td>Best GS</td>
<td>108.13</td>
<td>927.23</td>
<td>2614.28</td>
<td>102.84</td>
<td>1026.07</td>
<td>2936.25</td>
<td>82.70</td>
<td>729.82</td>
<td>2355.92</td>
<td></td>
</tr>
<tr>
<td>Best HGVZ</td>
<td>108.13</td>
<td>927.23</td>
<td>2610.72</td>
<td>111.86</td>
<td>1061.48</td>
<td>2979.33</td>
<td>106.69</td>
<td>774.68</td>
<td>2436.17</td>
<td></td>
</tr>
<tr>
<td>CPM(30,5)</td>
<td>15</td>
<td>133.68</td>
<td>1029.58</td>
<td>3226.05</td>
<td>135.41</td>
<td>1055.06</td>
<td>3567.12</td>
<td>87.22</td>
<td>847.77</td>
<td>2282.97</td>
</tr>
<tr>
<td>Enum(30,5)</td>
<td>121.21</td>
<td>1030.71</td>
<td>3220.65</td>
<td>130.78</td>
<td>1032.84</td>
<td>3576.60</td>
<td>79.37</td>
<td>839.37</td>
<td>2267.21</td>
<td></td>
</tr>
<tr>
<td>Best GS</td>
<td>121.18</td>
<td>1029.57</td>
<td>3194.74</td>
<td>130.73</td>
<td>1029.89</td>
<td>3564.09</td>
<td>79.36</td>
<td>835.37</td>
<td>2262.02</td>
<td></td>
</tr>
<tr>
<td>Best HGVZ</td>
<td>135.39</td>
<td>1040.60</td>
<td>3231.63</td>
<td>175.48</td>
<td>1082.40</td>
<td>3637.79</td>
<td>122.78</td>
<td>939.84</td>
<td>2399.95</td>
<td></td>
</tr>
<tr>
<td>CPM(30,5)</td>
<td>20</td>
<td>194.25</td>
<td>1395.83</td>
<td>4116.86</td>
<td>174.60</td>
<td>1052.83</td>
<td>3295.04</td>
<td>83.27</td>
<td>861.17</td>
<td>2493.62</td>
</tr>
<tr>
<td>Enum(30,5)</td>
<td>147.80</td>
<td>1376.12</td>
<td>4131.67</td>
<td>154.52</td>
<td>1036.81</td>
<td>3260.21</td>
<td>70.98</td>
<td>828.16</td>
<td>2411.29</td>
<td></td>
</tr>
<tr>
<td>Best GS</td>
<td>147.56</td>
<td>1376.12</td>
<td>4101.54</td>
<td>154.52</td>
<td>1032.64</td>
<td>3253.02</td>
<td>70.89</td>
<td>828.02</td>
<td>2411.29</td>
<td></td>
</tr>
<tr>
<td>Best HGVZ</td>
<td>160.78</td>
<td>1395.06</td>
<td>4120.06</td>
<td>208.95</td>
<td>1129.25</td>
<td>3363.47</td>
<td>146.12</td>
<td>1026.19</td>
<td>3231.32</td>
<td></td>
</tr>
</tbody>
</table>

6 Conclusions

We have presented several heuristic methods to solve capacity planning problems in which important practical issues such as capacity flexibility, precedence relations, and maximum work content per period can be taken into account. Such problems arise for instance in capacity driven multi-project organizations. In particular the precedence relations make the problems hard to solve. The main idea of our solution approach is to reduce the problem to a simple linear programming problem by introducing allowed to work windows. These ATW windows guarantee that precedence relations are met. The LP based heuristics try to find good ATW windows such that the solution for the corresponding LP is a good solution for the original capacity planning problem. We have discussed three categories of heuristics. The first category contains constructive heuristics. A constructive heuristic constructs only one feasible set of ATW windows. We presented the straightforward heuristics \( H_{\text{BASIC}} \) and \( H_{\text{CPM}} \). These are not really good heuristics, but they serve merely to provide an initial solution for other heuristics. The second category contains heuristics that start from an infeasible solution and convert this to a feasible solution. We presented the heuristic \( H_{\text{ENUM}} \) that repairs violated precedence constraints one by one. According to some criterion, the next precedence relation to be repaired is selected. For this relation, we evaluate all possibilities to repair its violation and then select the best one. The third category contains heuristics that start from a feasible solution and try to improve that solution by changing the ATW windows. Which ATW window will be changed is based on information about the dual solution to the corresponding LP. These heuristics are denoted by \( H_{\text{FEAS(BASIC)}} \) and \( H_{\text{FEAS(CPM)}} \), where
Basic and CPM refer to the constructive heuristics $H_{BASIC}$ and $H_{CPM}$ that were used to obtain an initial feasible solution.

Computational results show that $H_{FEAS}(CPM)$ and $H_{ENUM}$ clearly outperform our constructive heuristics and constructive heuristics from literature. Moreover, it turns out that the solutions still improve significantly if the starting solution and in particular the final solution is randomly disturbed and the improvement heuristic is restarted. The dual information gives good directions on which ATW should be changed. The combination of these ideas results in the best heuristic, in which $H_{FEAS(ENUM)}$ is used as initial solution for the improvement heuristic $H_{FEAS(\cdot)}$ and in which we randomly disturb both the initial solution and the final solution. In the best heuristic we tested, the initial solution is disturbed 29 times giving 30 initial solutions, and for each of these initial solutions the final solution is disturbed 4 times, giving 5 final solutions.

We compared our results to the solutions from Hans et al. They were able to prove optimality for 126 out of 450 test instances. Optimality could be proven in particular for the smaller instances. For 124 out of those 126 instances, we also found the optimal solution with one of our heuristics and for 113 instances the optimal solution was found by the best heuristic. For the larger instances, the solution of the truncated exact algorithm of Hans et al. was outperformed up to 35%. Therefore, we may conclude that our heuristics are very good heuristics for these types of capacity planning problems.

Both from the theoretical and the practical point of view, it would be very interesting to extend the model with several aspects. In particular, we mention the following. First of all, in the current model, preemption is allowed. For capacity planning, this is often not a problem, but from the practical point of view it may be desirable to limit the possibilities for preemption. For example, there should be a minimum work content in every time period in which a job is performed. Second, it is interesting to distinguish several types of nonregular capacity, such as working overtime, hiring, and subcontracting. Each of these types has its own availability and cost structure. Third, new personnel leads to additional costs, for example for training. Therefore, it is profitable to hire labor for a longer time period. Also, the management of additional work force is easier for smaller groups. Therefore, besides minimizing the costs for nonregular capacity, a practical objective is to balance the demand for, e.g., temporary labor. Fourth, other types of resources may require extensions to the current approach. Examples are spatial resources, such as docks and quays at a shipyard, and a hangar for aircraft maintenance. These aspects are subject to further research.

References


