IIE Transactions (2005) **37**, 153–165
Copyright © "IIE"
ISSN: 0740-817X print / 1545-8830 online
DOI: 10.1080/07408170590885611

Linear-programming-based heuristics for project capacity planning

NOUD GADEMANN[1] and MARCO SCHUTTEN[1,2*]

[1] *ORTEC bv, P.O. Box 490, 2800 AL Gouda, The Netherlands*
E-mail: ngademann@ortec.nl
[2] *School of Business, Public Administration and Technology, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands*
E-mail: m.schutten@utwente.nl

Received June 2002 and accepted May 2004

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 gain an insight into 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 linear-programming-based heuristics. Using a large set of test instances, we compare these heuristics with some results from the literature. It turns out that some of these heuristics are very powerful for solving capacity planning problems.

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, e.g., for reliable due date and price quotation and a 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 *et al.* (1994) 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 Waard (1999) evaluates this portfolio-based management approach using a number of case studies. Most companies in these case studies indicate that this organizational structure would benefit the company performance.

*Corresponding author

De Boer (1998) has developed a prototype decision support tool to support the decisions of the portfolio management team quantitatively. He also tested this tool at the Royal Netherlands Navy Dockyard, which is one of the involved companies of the case studies of De Waard (1999). 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 into 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 extent, 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. Currently, commercial software is available that is based on the prototype decision support system of De Boer. This commercial software includes the models described in this paper. The Royal Netherlands Navy Dockyard uses this software to support their business processes.

It is important to understand the difference between the two levels RCCP and RCPSP. In RCCP the planning horizon is split up into time buckets (e.g., weeks) unlike a continuous time horizon in the RCPSP, and the work packages are defined in terms of aggregated work content per required resource (e.g., a total of 200 hours of work content for electricians) unlike a specific execution mode with corresponding duration in the RCPSP (e.g., the work must be performed by two electricians each for 100 hours). RCCP is a tactical planning level which is concerned with allocating portions of the work content to time buckets (e.g., 50 hours in week 3, 150 hours in week 4) in order to determine the required resource availability levels and reliable due dates. To that extent nonregular capacity may be used (e.g., outsourcing, working overtime, or hiring additional personnel), which will involve additional costs. The planning objective will be a combination of time-related (e.g., minimizing project lateness) and cost-related (minimizing costs for deploying nonregular capacity or for lateness penalties) factors. The RCCP is used to determine the constraints for the operational scheduling, the RCPSP, in terms of, e.g., project milestones and available resources. In the RCPSP the smaller activities are scheduled over time according to available execution modes (e.g., two electricians start on Thursday in week 3 at 2:00 PM), and there is often very little or even no capacity flexibility. The objective is therefore often time related (e.g., minimizing makespan or minimizing maximum lateness). Note that the scheduling of activities belonging to a certain job is not restricted to the planned time frame of this job.

RCCP and RCPSP 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 gain insights into the capacity requirements and capacity availability. RCCP supports important aspects such as capacity requirements, the use of nonregular capacity (sub-contracting, working overtime, hiring), and order acceptance (due date and price quotation). Only for short-term planning is a detailed process planning required. The goal of the RCCP is to roughly match available and required capacity.

There is a close relationship 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: (i) order acceptance; (ii) process planning; (iii) scheduling; (iv) execution; and (v) 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, for example, Demeulemeester and Herroelen, 1992; Özdamar and Ulusoy, 1995; Kolisch and Drexl, 1996). 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 into the available capacity in the forthcoming months and in the required capacity for a new project. This means 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 in time, whereas the customer does not want to wait that long. This also suggests the use of 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 into 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 RCCP problem. We model, without loss of generality, the multiple-project capacity planning problem as a single-project capacity planning problem. This single project contains all jobs from all projects. As a result, the corresponding network need not be connected. Each job may have its own release and due date. In this way, we can account for the project release and due dates. Analogously to the variants that Möhring (1984) 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 maximum job lateness.

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

2. Problem description

The problem that we consider can be formally described as follows (De Boer, 1998; De Boer and Schutten, 1999). 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 say one week. The regular capacity of resource R_k ($k = 1, 2, \dots, K$) in week t is equal to Q_{kt} hours. Job J_j requires q_{jk} hours of resource R_k. In general, the jobs have a work content of several weeks and will be subdivided into a number of activities for scheduling (at the RCPSP level). Therefore, we will allow that the fraction of a job that is planned in a certain week may vary over time.

Let x_{jkt} denote the fraction of job J_j that is performed on resource R_k in week t. This means that $x_{jkt}\, q_{jk}$ hours are spent on job J_j in week t on resource R_k. We assume that $x_{jkt} = x_{jlt}$ ($k, l = 1, 2, \dots, K$), i.e., that an equal fraction is spent on job J_j in week t on all resources. This assumption is realistic for capacity planning, since the underlying activities of a job are in general multi-resource activities. In the RCPSP, it is necessary that the required resources for each activity are available at the same time. Note that if spending equal fractions for a specific job is not realistic at the planning level, then this job can be split at the planning level into two or more smaller jobs (one for each required resource). 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 will allow the fractions to differ from week to week, implying that in a certain week no work is done on a job. We also assume that for each job J_j we are given a maximum fraction $1/p_j$ 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 $J_i \to J_j$. This means that job J_i must be completely performed before we can start performing job J_j, i.e., $x_{j\tau} > 0 \Longrightarrow \sum_{t=1}^{\tau-1} x_{it} = 1$ for $1 \le \tau \le T$. We assume that the time windows respect the precedence relations, i.e., $r_j \ge r_i + p_i$ and $d_i \le d_j - p_j$ if $J_i \to J_j$. 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 ($r_j + p_j - 1 \le d_j$ 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 $[x_{11}, x_{12}, \dots, x_{1T}, x_{21}, x_{22}, \dots, x_{2T}, \dots, x_{n1}, x_{n2}, \dots, x_{nT}]$ 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 $U_{kt} = \max\{0, \sum_{j=1}^{n} q_{jk}\, x_{jt} - Q_{kt}\}$ 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 $c_{kt} U_{kt}$. We assume that the availability of nonregular capacity is infinite and that $c_{kt} \ge 0, \forall k, t$. Our objective is to minimize the total cost of required nonregular capacity. The assumption on unlimited availability of the resources is not restrictive for the model and the approach. Additional constraints on maximum resource availability can be added.

Table 1. Data for example instance

J_j	r_j	d_j	p_j	q_{j1}	q_{j2}
J_1	1	4	1	60	120
J_2	3	6	2	80	120
J_3	3	6	1	70	105
J_4	4	8	1	60	150

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 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 Fig. 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 on 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):

$$(\mathrm{P}): \quad \min \sum_{t=1}^{T} \sum_{k=1}^{K} c_{kt}\, U_{kt},$$

subject to

$$\sum_{t=r_j}^{d_j} x_{jt} = 1 \qquad \forall j, \tag{1}$$

$$x_{jt} \le \frac{1}{p_j} \qquad \forall j, t, \tag{2}$$

$$U_{kt} \ge \sum_{j=1}^{n} q_{jk}\, x_{jt} - Q_{kt} \qquad \forall k, t, \tag{3}$$

$$x_{jt}, U_{kt} \ge 0 \qquad \forall j, k, t, \tag{4}$$

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

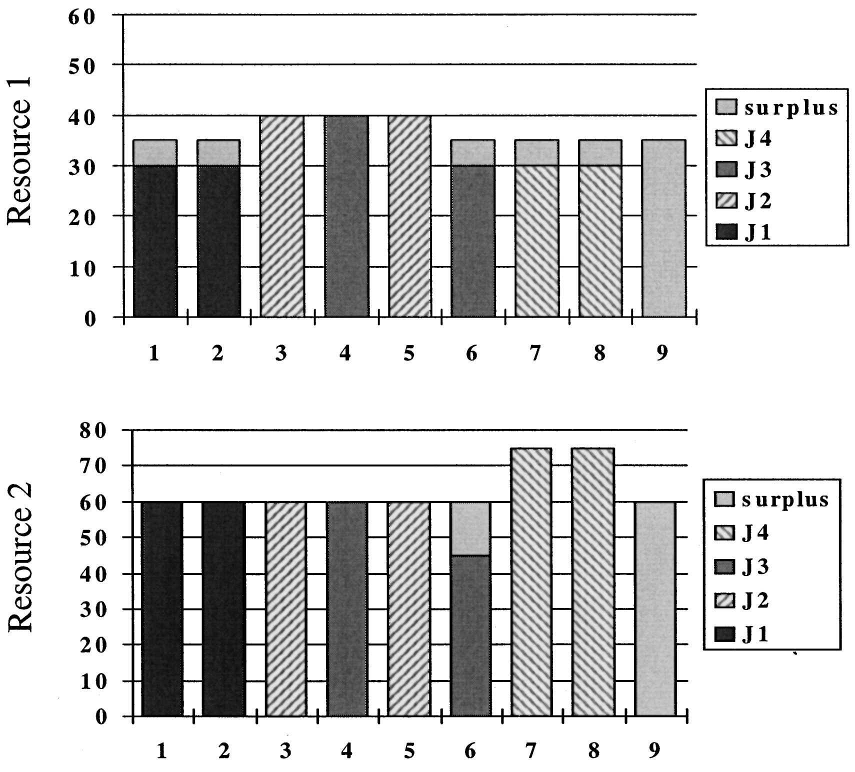

Fig. 1. A feasible solution for the example instance.

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 work on job J_j. This means that we do not allow work on job J_j before week S_j and also not later than week C_j. Associated with each set $\mathcal{S}$ of ATW windows, we consider the linear programming problem (P$_\mathcal{S}$). Problem (P$_\mathcal{S}$) is obtained from the linear programming problem (P) by replacing constraints (2) with:

$$x_{jt} \leq \frac{s_{jt}}{p_j} \qquad \forall j, t, \tag{5}$$

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 $\mathcal{S}$ be a set of ATW windows, with one ATW window $[S_j, C_j]$ for every job J_j. We call $\mathcal{S}$ feasible, if every ATW window in $\mathcal{S}$ is feasible and moreover:

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

The last condition ensures that the precedence relations are met. Clearly, if $\mathcal{S}$ is feasible, then any feasible solution to problem (P$_\mathcal{S}$) is feasible for the RCCP problem.

In the remainder of this paper, we investigate heuristics based on problem (P$_\mathcal{S}$). 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 $\mathcal{S}$ of ATW windows. By solving the associated linear programming problem, (P$_\mathcal{S}$), we obtain 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 (1999) 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 into 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 also have to be updated. For more details on this heuristic, we refer to Section 4.2.

Hans (2001) and Hans *et al.* (2002a) 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 variants. In this model, the total cost of both nonregular capacity usage (subcontracting, hiring, working overtime) and tardiness penalties is minimized. They formulate this problem as a mixed-integer programming problem with an exponentional number of variables and present an exact branch-and-price algorithm for its solution. They also suggest several primal and improvement heuristics for the problem. Test results show that small to moderate sized instances can be solved to optimality in a practical time. In combination with some of the heuristics, a truncated branch-and-price method also gives good results for larger instances.

In Hans *et al.* (2002b) 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 the 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, such as pricing heuristics and an integer programming formulation of the pricing problem that is solved directly using 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 (1999) 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 $(\mathrm{P}_{\mathcal{S}})$ 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). All heuristics that we present proceed as follows. Iteratively, an ATW window is set for every job and the associated Linear Programming (LP) problem of Section 2 is solved. The differences in the heuristics are the way the ATW windows are set and how they are updated as a result of solving the LP. Schematically, the heuristics proceed as follows:

Category 1 heuristics:

Step 1. Construct a feasible set $\mathcal{S}$ of ATW windows.
Step 2. Solve problem $(\mathrm{P}_{\mathcal{S}})$.

Category 2 heuristics:

Step 1. Initialize with some set $\mathcal{S}$ of ATW windows.
Step 2. Solve problem $(\mathrm{P}_{\mathcal{S}})$.
Step 3. If no violated precedence relation exists, then stop (solution is feasible).
Step 4. Choose a violated precedence relation and repair it by changing set $\mathcal{S}$.
Step 5. Go to Step 2.

Category 3 heuristics:

Step 1. Initialize with a feasible set $\mathcal{S}$ of ATW windows.
Step 2. Solve problem $(\mathrm{P}_{\mathcal{S}})$.
Step 3. If some termination criterion is met, then stop.
Step 4. Change set $\mathcal{S}$ by using dual LP information, but maintain the feasibility of $\mathcal{S}$.
Step 5. Go to Step 2.

4.1. *Category 1 heuristics*

A feasible set of ATW windows $\mathcal{S}$ can easily be constructed (Step 1) 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 problem ($P_{\mathcal{S}}$) (Step 2). 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 $\{J_{i_1}, J_{i_2}, \ldots, J_{i_k}\}$ is called a *path* if $J_{i_1} \rightarrow J_{i_2}$, $J_{i_2} \rightarrow J_{i_3}, \ldots, J_{i_{k-1}} \rightarrow J_{i_k}$. A *critical path* $\{J_{j_1}, J_{j_2}, \ldots, J_{j_R}\}$ is a path for which it holds that:

- $L_{j_1} = L_{j_2} = \cdots = L_{j_R} = \min_{1 \le j \le n} L_j$;
- $S_{j_i} = S_{j_{i-1}} + p_{j_{i-1}}$ for $2 \le i \le R$;
- $C_{j_i} = C_{j_{i+1}} - p_{j_{i+1}}$ for $1 \le i \le R - 1$.

A critical path $\{J_{j_1}, J_{j_2}, \ldots, J_{j_R}\}$ is a *maximal critical path* if there is no job J_k such that either $\{J_k, J_{j_1}, J_{j_2}, \ldots, J_{j_R}\}$ or $\{J_{j_1}, J_{j_2}, \ldots, J_{j_R}, J_k\}$ 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 necessarily result in a feasible set of ATW windows. Next, we find a maximal critical path, say $\{J_{j_1}, J_{j_2}, \ldots, J_{j_R}\}$. The *total slack* $\bar{L}$ of this path is defined as $C_{j_R} - (S_{j_1} + \sum_{i=1}^{R} p_{j_i})$. The total slack is distributed over the maximal path by changing the ATW windows for these jobs by first setting $S_{j_i} = S_{j_1} + \sum_{k=1}^{i-1} p_{j_k} + \lfloor \bar{L} \sum_{k=1}^{i-1} p_{j_k} / \sum_{k=1}^{R} p_{j_k} \rfloor$ for $1 < i \le R$ and then $C_{j_i} = S_{j_{i+1}} - 1$ for $1 \le i < R$, where $\lfloor x \rfloor$ 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., $S_i + p_i \le S_j$ and $C_i \le C_j - p_j$ 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 windows is a feasible set of ATW windows (Step 1). We will refer to this heuristic (including Step 2) as H_{CPM}.

4.2. *Category 2 heuristics*

De Boer and Schutten (1999) suggest a number of heuristics based on problem (P), i.e., based on problem ($P_{\mathcal{S}}$) with set $\mathcal{S}$ such that $S_j = r_j$ and $C_j = d_j$ (Step 1). A solution to problem (P) (Step 2) may violate one or more precedence relations. The suggested heuristics are based on repairing violated precedence relations one by one (Step 4). 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 was the most successful. 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} (Step 4). 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 in specifying a T_{kl} to repair the precedence relation $J_k \rightarrow J_l$ and we should use this little freedom we have as well as possible. Therefore, instead of using a simple rule to determine T_{kl}, we suggest the evaluation of all possible values of T_{kl}, that is, $r_l \le T_{kl} \le d_k + 1$, 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 it will be done. Analogously, we investigate whether we can increase d_m for predecessors of job J_l. 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 have 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 $\mathcal{S}$ and want to improve the corresponding solution, which is optimal for problem ($P_{\mathcal{S}}$). 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 one. 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 $\mathcal{S}$, we also have to decrease C_k. 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 $C_k - S_k \ge p_k$ 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 ($P_{\mathcal{S}}$), the right-hand side of constraint (5) for $t = S_j^{'}$ must be changed from 0 to $1/p_j$. If we also have to decrease C_k for a predecessor of job J_j 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 it will be done. Denote the resulting set of ATW windows by $\mathcal{S}'$.

Similar conditions and changes in problem ($P_{\mathcal{S}}$) can be derived for the other possible changes of the ATW window $[S_j, C_j]$.

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

Starting from any feasible set $\mathcal{S}$, 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 $\mathcal{S}$; (Step 1).
2. The acceptance criterion; (Step 3).
3. The neighbors of the current feasible set $\mathcal{S}$ that are evaluated (Step 4).

In particular with respect to the selection of the neighbors to be evaluated, we use information from LP theory.

4.3.2. *LP theory*

Every feasible solution is characterized by a feasible set $\mathcal{S}$ of ATW windows. The corresponding solution is found by solving the associated LP problem, problem ($P_{\mathcal{S}}$). As described in Section 4.3.1, a neighbor $\mathcal{S}'$ of $\mathcal{S}$ 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, problem ($P_{\mathcal{S}'}$) differs only slightly from problem ($P_{\mathcal{S}}$), 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 LP 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 $\mathbf{B}_{\mathcal{S}}$ denotes the basis matrix and $b_{\mathcal{S}}$ the right-hand side for the LP problem ($P_{\mathcal{S}}$) then the same basis remains feasible (and thus optimal) for $\mathcal{S}'$ if:

$$\mathbf{B}_{\mathcal{S}}^{-1} b_{\mathcal{S}'} \geq 0.$$

For the moment, we assume that the same basis remains optimal for a neighbor $\mathcal{S}'$ of $\mathcal{S}$. Let $\Pi_{\mathcal{S}}$ denote the optimal dual solution and $z_{\mathcal{S}}$ the optimal objective function value for problem ($P_{\mathcal{S}}$), then:

$$z_{\mathcal{S}'} = \Pi_{\mathcal{S}} b_{\mathcal{S}'}.$$

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 ($P_{\mathcal{S}}$). Furthermore, let $\Delta(\mathcal{S}, \mathcal{S}')$ denote the set of indices of the constraints for which the right-hand sides for $\mathcal{S}$ and $\mathcal{S}'$ differ, and, for $i \in \Delta(\mathcal{S}, \mathcal{S}')$ let Δb_i denote the corresponding difference in the right-hand side, then:

$$z_{\mathcal{S}'} = z_{\mathcal{S}} + \Delta_z(\mathcal{S}, \mathcal{S}'),$$

with

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

Thus, under the assumption that the same basis remains optimal, we can use the optimal dual solution $\Pi_{\mathcal{S}}$ for problem ($P_{\mathcal{S}}$) to compute the change in objective function if we move from $\mathcal{S}$ to $\mathcal{S}'$.

Moving to a neighbor $\mathcal{S}'$ of $\mathcal{S}$ 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, $\Delta_z(\mathcal{S}, \mathcal{S}')$ may give a good indication of the impact on the optimal objective function. Since $\Delta_z(\mathcal{S}, \mathcal{S}')$ can be easily 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 obtain a feasible set of ATW windows we have to change s_{kC_k} from one to zero 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:

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

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. $H_{FEAS(\cdot)}$

Empirically, it has been shown that a steepest-descent step in the Simplex method for LP works very well. We adopt this result in evaluating the neighbors of a set $\mathcal{S}$. In heuristic $H_{\text{FEAS(BASIC)}}$, we generate an initial feasible set $\mathcal{S}$ by heuristic H_{BASIC}. Next, we order all neighbors of $\mathcal{S}$ by increasing value of $\Delta_z(\mathcal{S}, \mathcal{S}')$. 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 $\mathcal{S}'$. The heuristic 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 on the initial feasible solution. To test the influence of the starting solution on the final solution, we also implemented heuristic H_{CPM} to find a starting solution. Apart from the starting solution, heuristic $H_{\text{FEAS(CPM)}}$ is identical to heuristic $H_{\text{FEAS(BASIC)}}$.

In the heuristics $H_{\text{FEAS(BASIC)}}$ and $H_{\text{FEAS(CPM)}}$, we use the dual values of constraint (5) to obtain an insight into 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 $x_{j,C_j+1} \leq 1/p_j$, due to constraint (2). We also used $(1/p_j)\pi_j(C_j + 1)$ 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 $-(1/p_j)\pi_j(T)$ 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 (1999). An instance in this set is characterized by the three parameters n (the number of jobs), K (the number of resources), and ϕ (average float). The average float is defined as $\phi = (\sum_{j=1}^{n} f_j)/n$, where $f_j = d_j - (r_j + p_j) + 1$ is the float of job J_j. The used values for parameter n were $n = 10$, 20 and 50; for parameter K, $K = 3$, 10, and 20; and for parameter ϕ, $\phi = 2$, 5, 10, 15 and 20. For all possible combinations of n, K and ϕ, 10 instances were generated, which results in a total of 450 instances.

To generate precedence relations between the jobs, the network generation procedure of Kolisch *et al.* (1995) 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 has as yet no successor. 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 ϕ is equal to a specified value. For further details about this procedure, we refer to De Boer and Schutten (1999). 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 zero and 20. Each job requires at least one and at most five 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 obtain an impression on how good the heuristics perform, we can compare them with one another and also to a lower bound to the optimal solution value. De Boer and Schutten (1999) 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 on 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 $lb_2 = \max\{LB_{ij} | (i,j) \in \mathcal{P}\}$, with $(i,j) \in \mathcal{P}$ 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, $lb_2 \geq lb_1$.

Problem (P) ignores the precedence constraints. By adding the following constraints, we can model the precedence relations explicitly in the model:

$$\sum_{\tau=r_j}^{t-1} x_{i\tau} \geq y_{jt} \quad \forall (i,j) \in \mathcal{P}, \quad r_j \leq t \leq d_i, \tag{6}$$

$$\sum_{\tau=r_j}^{t} x_{j\tau} \leq y_{jt} \quad \forall j, t, \tag{7}$$

$$y_{jt} \in \{0, 1\}. \tag{8}$$

The binary variable y_{jt} indicates whether all predecessors of job J_j have been completely performed before week t. This means that we can only work on job J_j in week t if the variable y_{jt} is equal to one. Constraints (6) ensure that y_{jt} is equal to zero if at least one of the predecessors of job J_j has not been completely performed before week t. Constraints (7) guarantee that we do not work on job J_j in week t (and before) unless all its predecessors have been completely performed, i.e., variable y_{jt} is equal to one. We will refer to the problem that we get from adding constraints (6)–(8) to problem (P) as problem (P_E). The solution value of the LP relaxation of problem (P_E) gives another lower bound (lb_3). It holds that $lb_3 \geq lb_1$.

5.3. *Computational results*

Most of the computational experiments were performed on a computer with a Pentium II processor running at

Table 2. Computational results for test instances

		H_{ENUM}		$H_{FEAS(BASIC)}$		$H_{FEAS(CPM)}$	
	H_{WC}	*Min slack*	*Max slack*	*No detail*	*Detail*	*No detail*	*Detail*
Value	1470.4	1307.3	1314.4	1408.2	1410.9	1324.2	1325.0
Dev. (%)	8.80	5.24	5.39	7.88	7.91	5.84	5.90
Max dev (%)	27.63	15.71	16.13	25.80	25.80	24.93	24.93
Time (s)	20	13	21	4	4	6	6
Max time (s)	193	237	459	113	219	200	130
Number best	6	215	143	56	56	133	137
Number unique best	0	151	92	3	1	23	30

500 MHz. The exception being the experiments for heuristic H_{WC} which were 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} in Table 2 is therefore not directly comparable with the computation times of the other heuristics. The average value of the lower bound lb_1 used by De Boer and Schutten (1999) is 939.7. The lower bound lb_2 that we developed has an average value of 984.1. The lower bound lb_3 has an average value of 948.5. In 400 cases (out of 450), lb_2 is strictly larger than lb_3, whereas in 34 cases lb_3 is strictly larger than lb_2. Table 2 shows the results of the different heuristics for the 450 test instances. The row "Value" denotes the average value of the objective function. Analogously to De Boer and Schutten, we use:

$$Dev = \frac{\sum_{t=1}^{T}\sum_{k=1}^{K} U_{kt} + \sum_{t=1}^{T}\sum_{k=1}^{K} Q_{kt} - LB}{LB},$$

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 $lb2$ 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 "Number best" shows the number of instances for which a heuristic found the best solution value. The row "Number unique best" indicates the number of instances for which a heuristic found a better solution than all other heuristics. The column "H_{WC}" shows the results for the best heuristic of De Boer and Schutten (1999). 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 whether or not detailed shadow prices are used.

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 $H_{FEAS(\cdot)}$ 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}.

Note that all heuristics are local search heuristics. The number of solutions in the search space is exponential and the number of iterations may therefore be large. The experiments show, however, that convergence to a local optimum is quite fast on the average, with some extreme exceptions (for the instances with a large number of jobs).

Since Table 2 suggests that the starting solution affects the quality of heuristic $H_{FEAS(\cdot)}$, we tried H_{ENUM} (with the minimum slack ordering) as a starting solution for $H_{FEAS(\cdot)}$, 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 $H_{FEAS(\cdot)}$, 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 $H_{FEAS(\cdot)}$ 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 by one time unit. Of all those possibilities, we randomly pick one. The next question is how many of these 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 a number of such disturbances equal to $n/4$ where n is the number of jobs produces good solutions.

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. Figure 2 compares the quality and the required computational effort of four 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 2–30 use randomly disturbed versions of H_{CPM} as a starting solution. The ith 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 four times (resulting in five 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 considerably improves the solution quality 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. We see that "Enum(30,5)" after 1 minute of computation time (i.e., after three experiments out of 30) outperforms the best results of the other heuristics after 30 experiments. Both the solution quality and the required computation time suggest that this heuristic is the best choice. In practice, the computation times are important, e.g., when the model is used for scenario analysis in a meeting of the portfolio management team (see Section 1). The available computation time determines the number of experiments that should be performed.

Hans *et al.* (2002b) 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 ϕ (the average float). The rows labeled as "Best HGVZ" contain the number of times that at least one of the algorithms of Hans *et al.* (2002b) found an optimal solution. Rows with the label "MIP" contain the results for optimizing problem (P_E). For each instance, we stopped the calculations after 30 minutes of computation time. Note, however, that problem (P_E) was solved using CPLEX 8.1 on a computer with a Pentium 4 processor running at 2.5 GHz. Experimentally, we found that this computer is more than four times faster than the computer

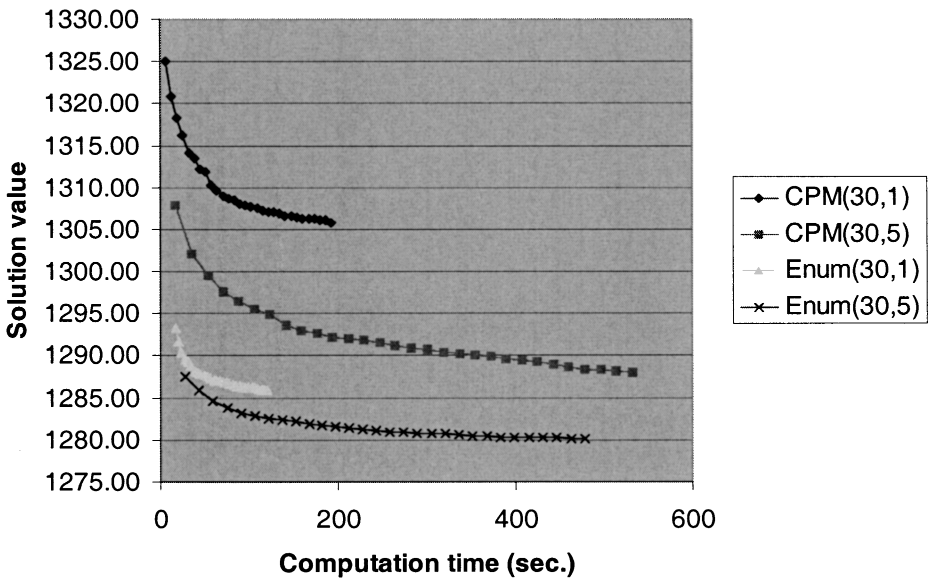

Fig. 2. Solution value as a function of computation time.

Table 3. Number of found optimal solutions

		n									
		10			*20*			*50*			
Algorithm	*K/φ*	*3*	*10*	*20*	*3*	*10*	*20*	*3*	*10*	*20*	*Total*
CPM(30,5)	2	10	9	10	10	10	10	8	8	9	84
Enum(30,5)		10	9	9	10	10	10	8	8	10	84
Best GS		10	10	10	10	10	10	10	10	10	90
Best HGVZ		10	10	10	10	10	10	9	9	7	85
MIP		10	10	10	10	10	10	10	10	10	90
CPM(30,5)	5	8	9	8	6	7	5	2	3	2	50
Enum(30,5)		9	8	7	9	8	6	8	5	2	62
Best GS		10	10	9	9	9	8	8	6	3	72
Best HGVZ		10	10	10	9	3	3	0	0	0	45
MIP		10	10	10	10	10	10	10	10	5	85
CPM(30,5)	10	6	7	5	2	4	7	0	0	0	31
Enum(30,5)		8	10	5	6	8	3	2	0	0	42
Best GS		10	10	7	6	9	7	3	0	0	52
Best HGVZ		10	10	7	0	1	0	0	0	0	28
MIP		10	10	10	10	10	10	10	1	0	71
CPM(30,5)	15	4	8	4	5	0	1	0	0	0	22
Enum(30,5)		6	7	6	7	1	2	0	0	0	29
Best GS		6	9	8	8	1	2	0	0	0	34
Best HGVZ		1	3	3	0	1	0	0	0	0	8
MIP		10	10	10	10	8	5	2	0	0	55
CPM(30,5)	20	1	5	5	1	0	0	0	0	0	12
Enum(30,5)		7	9	6	5	0	2	0	0	0	29
Best GS		9	9	7	5	0	2	0	0	0	32
Best HGVZ		3	4	3	0	0	0	0	0	0	10
MIP		10	10	10	10	4	2	0	0	0	46

Table 4. Detailed comparison of the results

		n								
		10			*20*			*50*		
Algorithm	*K/φ*	*3*	*10*	*20*	*3*	*10*	*20*	*3*	*10*	*20*
CPM(30,5)	2	0.0	0.1	0.0	0.0	0.0	0.0	0.1	0.2	0.0
Enum(30,5)		0.0	0.1	0.0	0.0	0.0	0.0	0.1	0.0	0.0
Best HGVZ		0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1
MIP		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
CPM(30,5)	5	0.2	0.0	0.0	0.6	0.5	0.2	0.8	0.7	0.2
Enum(30,5)		0.0	0.1	0.2	0.0	0.1	0.0	0.0	0.0	0.1
Best HGVZ		0.0	0.0	−0.1	−0.5	0.7	0.0	5.7	4.2	0.8
MIP		0.0	0.0	−0.1	−0.9	−0.1	−0.2	−0.1	−0.1	−0.1
CPM(30,5)	10	6.9	0.3	0.1	6.7	0.5	0.0	9.0	1.8	0.7
Enum(30,5)		4.9	0.0	0.2	0.1	0.4	0.3	0.4	0.2	0.1
Best HGVZ		0.0	0.0	−0.1	29.6	3.8	1.5	38.5	7.0	3.6
MIP		0.0	0.0	−0.3	−1.8	0.0	−0.1	−2.2	0.4	1.5
CPM(30,5)	15	7.0	0.0	1.0	8.3	1.8	0.1	13.3	1.3	0.9
Enum(30,5)		0.0	0.1	0.7	0.0	0.4	0.3	0.0	0.5	0.3
Best HGVZ		13.9	1.7	1.1	158.2	6.7	2.3	68.4	12.8	6.5
MIP		−0.2	0.0	0.0	−2.0	−0.5	−0.2	−0.6	1.7	2.8
CPM(30,5)	20	44.3	1.5	0.4	16.1	2.0	1.5	18.0	6.3	3.5
Enum(30,5)		0.2	0.0	0.6	0.0	0.5	0.2	0.2	0.0	0.0
Best HGVZ		13.1	1.2	0.6	49.2	9.5	3.5	109.9	42.4	36.9
MIP		−0.2	−0.5	−0.6	−1.4	−0.6	0.8	2.9	5.1	5.7

used for the other experiments. Solving problem (P_E) had more success than the approach of Hans *et al.* (2000b). Every instance that was solved to optimality by Hans *et al.*, was also solved to optimality using problem (P_E). Rows labeled as "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 four times, finds the known optimum solution. Clearly, the number presented in those rows cannot be larger than those in "MIP," since we do not know the optimal solution for those instances that were not solved exactly using problem (P_E). In the same way, rows labeled as "Enum(30,5)" present results for the same heuristic that only uses H_{ENUM} as a starting solution. The rows labeled as "Best GS" present the numbers for the best of the latter two heuristics. We see that all instances with 10 jobs and almost all instances with 20 jobs are solved to optimality by using problem (P_E). For problems with a large average float and a large number of resources, this approach becomes less successful. In total, 347 cases are solved to optimality (using problem (P_E)). Hans *et al.* (2002b) were particularly able to solve instances with a small number of jobs and with a small average float. In total, they were able to solve 176 case optimally. The heuristic based on H_{ENUM} also found an optimal solution in 246 cases, whereas the best algorithm found an optimal solution in 280 cases.

Table 4 shows detailed results on the average solution values, depending on n, K and ϕ. In this table, we compare the heuristics in a relative form with "Best GS." For example, the objective function value of "CPM(30,5)" is on average 44.3% worse than the objective function value of "Best GS" for instances with $n = 10$, $K = 3$ and $\phi = 20$. We see that our solutions are comparable to the solutions obtained by solving problem (P_E) for instances with a small number of jobs or a small average float. For instances with a large number of jobs and with a large average float, our solutions are on average often better. Moreover, the required CPU time for solving problem (P_E) is comparable to the required CPU time for our best heuristic. However, problem (P_E) was solved on a computer that was more than four times faster than the computer used for the experiments with our heuristics. Thus, our heuristics are expected to be at least four times faster.

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 LP problem by introducing ATW 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_{BASIC} and H_{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_{ENUM} that repairs violated precedence constraints one by one. Using a given 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 $H_{FEAS(BASIC)}$ and $H_{FEAS(CPM)}$, 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 previously reported in the literature. Moreover, it turns out that the solutions still significantly improve 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 an 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 four times, giving five final solutions.

We compared our results to the solutions from Hans *et al.* (2002b) and to solutions obtained by solving problem (P_E). Optimality could be proven for 347 instances. For 280 out of those 347 instances, we also found the optimal solution with one of our heuristics and for 246 instances the optimal solution was found by the best heuristic ("Enum(30,5)"). For small instances, our heuristics produce solutions comparable to those produced by solving problem (P_E). For large instances, our heuristics are generally better and much faster than solving problem (P_E). Therefore, we may conclude that our heuristics are very good heuristics for these type of capacity planning problems.

Both from the theoretical and the practical point of view, it would be very interesting to extend the model in 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 the 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.

Acknowledgements

The authors gratefully acknowledge the constructive comments of S. Rajagopalan and of the anonymous referees.

References

De Boer, R. (1998) Resource-constrained multi-project management. PhD thesis, University of Twente, Enschede, The Netherlands.

De Boer, R. and Schutten, J.M.J. (1999) Multi-project rough-cut capacity planning. in *Flexible Automation and Intelligent Manufacturing*, Ashayeri, J., Sullivan, W.G. and Ahmad, M.M. (eds.), Begell House, Wallingford, UK, pp. 631–644.

Demeulemeester, E.L. and Herroelen, W.S. (1992) A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, **38**, 1803–1818.

De Waard, A.J. (1999) Re-engineering non-flow large scale maintenance organisations. PhD thesis, University of Twente, Enschede, The Netherlands.

Hans, E.W. (2001) Resource loading by branch-and-price techniques. PhD thesis, University of Twente, Enschede, The Netherlands.

Hans, E.W., Gademann, A.J.R.M., Van de Velde, S.L. and Zijm, W.H.M. (2002a) Resource loading by branch-and-price techniques: models and algorithms. Technical report BETA-WP87, University of Twente, Enschede, The Netherlands.

Hans, E.W., Gademann, A.J.R.M., Van de Velde, S.L. and Zijm, W.H.M. (2002b) Rough-cut capacity planning by branch-and-price techniques. Technical report, University of Twente, Enschede, The Netherlands.

Kolisch, R. and Drexl, A. (1996) Adaptive search for solving hard project scheduling problems. *Naval Research Logistics*, **43**, 23–40.

Kolisch, R., Sprecher, A. and Drexl, A. (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, **41**, 1693–1703.

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

Özdamar, L. and Ulusoy, G. (1995) A survey on the resource-constrained project scheduling problem. *IIE Transactions*, **27**, 574–586.

Platje, A., Seidel, H. and Wadman, S. (1994) Project portfolio planning cycle: project based management for the multi-project challenge. *International Journal of Project Management*, **12**(2), 100–106.

Biographies

Since 1998, Noud Gademann has been a managing consultant with ORTEC bv, a leading software and consulting company in advanced planning and scheduling. He has both an M.Sc. and a Ph.D. in Applied Mathematics from the University of Twente, The Netherlands. From 1993 to 2003, he was affiliated with the same university, first as an Assistant Professor in Production and Operations Management in the Faculty of Mechanical Engineering (till 1998), and then as a part-time research fellow in the Center for Production, Logistics and Operations Management. Although he has now left academia, he continues to do research, particularly in the area of combinatorial optimization, warehousing, and production and project management.

Marco Schutten is an Assistant Professor at the School of Business, Public Administration and Technology of the University of Twente, The Netherlands. He received his Ph.D. in 1996 from this university on the subject of job shop scheduling. His current research interests include production and project planning and scheduling. In 1999, he started working at ORTEC, a company developing decision support software for planning questions and giving advice on these subjects. Currently, he is a senior consultant in the area of production and project planning.

Contributed by the Scheduling/Production Planning/Capacity Planning Department