INFORMS Journal on Computing
Vol. 23, No. 2, Spring 2011, pp. 189–204
ISSN 1091-9856 | EISSN 1526-5528 | 11 | 2302 | 0189

DOI 10.1287/ijoc.1100.0393
© 2011 INFORMS

Decision Support and Optimization in Shutdown and Turnaround Scheduling

Nicole Megow
Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany, nmegow@mpi-inf.mpg.de

Rolf H. Möhring, Jens Schulz
Institut für Mathematik, Technische Universität Berlin, 10623 Berlin, Germany
{moehring@math.tu-berlin.de, jschulz@math.tu-berlin.de}

Large-scale maintenance in industrial plants requires the entire shutdown of production units for disassembly, comprehensive inspection, and renewal. We derive models and algorithms for this so-called turnaround scheduling that include different features such as time-cost trade-off, precedence constraints, external resource units, resource leveling, different working shifts, and risk analysis. We propose a framework for decision support that consists of two phases. The first phase supports the manager in finding a good makespan for the turnaround. It computes an approximate project time-cost trade-off curve together with a stochastic evaluation. Our risk measures are the expected tardiness at time t and the probability of completing the turnaround within time t. In the second phase, we solve the actual scheduling optimization problem for the makespan chosen in the first phase heuristically and compute a detailed schedule that respects all side constraints. Again, we complement this by computing upper bounds for the same two risk measures.

Our experimental results show that our methods solve large real-world instances from chemical manufacturing plants quickly and yield an excellent resource utilization. A comparison with solutions of a mixed-integer program on smaller instances proves the high quality of the schedules that our algorithms produce within a few minutes.

Key words: project management; planning; scheduling; resource constraints; risk analysis; applications; large-scale systems; chemical industries
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received August 2009; revised February 2010; accepted March 2010. Published online in *Articles in Advance* July 2, 2010.

1. Introduction

Large-scale maintenance activities are conducted on a regular basis in industrial settings such as chemical manufacturing, refining, or power plants. Entire production units are shut down for disassembly, comprehensive inspection, and renewal. Such a process is called *shutdown and turnaround* (or turnaround, for short). It is an essential process but causes high out-of-service cost. Therefore a good schedule for the turnaround has a high priority to the manufacturer. A good schedule is not simply a short schedule. The project execution can be speeded up at the expense of adding resource units, mostly in the form of additional workers. Thus, short projects cause high resource costs, whereas cheap projects take a long time. Moreover, in practice, task execution times typically involve uncertainty. Such uncertainty arises as a result of unforeseen repair jobs, and naturally, a short schedule is less robust against unexpected repair jobs or processing delays than a schedule with a long duration that offers more flexibility for rescheduling. Such considerations are fundamental in the decision process of a turnaround project manager. We support this process by analyzing the trade-off between project duration and project cost as well as the effects on the stability of schedules. Our main contribution is an optimization algorithm within a larger decision support framework that computes a detailed schedule of given project duration with the aim of minimizing the total resource cost.

Clearly, turnaround projects differ in size, duration, and particular specifications depending on the actual industrial site. Generally, the turnaround of a large plant may take in total up to one year and must be repeated every four to six years. Typically, such large projects are split into a sequence of shutdowns of single production units that are planned individually. The time horizon for those projects is often between two and four weeks. The complex working steps of a turnaround are planned in advance with very high granularity. They are split into many jobs that must be executed in parallel or directly after each other by workers with a particular specialization such as electricians, pipe fitters, inspectors, cranes, crane drivers, or other craftsmen. In theory, models with jobs that are executed by workers with different specializations seem plausible, but here we focus on the case where each job needs exactly one specialized type of worker.

This is a consequence of the fine resolution of working steps in turnaround planning, and it is not too restrictive because we can model the more general case by enforcing parallel precedence relations for subsets of jobs; see §3 for more details. At this level of planning, the turnaround of an entire manufacturing site may consist of 100,000 to 150,000 jobs, whereas the typically considered (sub)project size for shutting down; e.g., a single cracker is between 500 and 2,000 jobs.

More explicitly, we model a turnaround project as a huge number of precedence-constrained operations or jobs that must be executed by maintenance groups of different specializations. Scheduling these is already a complex task because various working shifts must be respected. However, in this particular problem another issue increases the complexity drastically: the duration of a job is flexible in the sense that a job can be accelerated by increasing the number of resource units (workers) allocated to it. Typically, technical reasons restrict the choice to a range between a maximum and minimum number of workers. We can assume that the duration of a job is a nonincreasing discrete function of the number of workers allocated to it. Because of communication overhead between the workers, the duration of a job decreases at a smaller rate than the rate at which the number of assigned workers increases.

We call workers with a particular specialization a *resource type* and an individual worker from such a group a *resource unit* of that type. Each resource unit causes a certain cost for each time unit that it is used. For various resource types, a certain number of resource units is generally available on the manufacturing site for flexible job assignments. We call them *internal* resources. However, during a turnaround, a significant overhead of work is to be done in a short time. This requires hiring *external* resources if internal capacities are exhausted. Typically, external resource units can only be hired for a certain minimum time period and must be paid for even when they are idle. Therefore, we aim for a schedule in which the consumption of these resource types is *leveled* over the period in which they are hired—in our model, this is the entire turnaround period. By *leveled*, we mean that resource units are allocated to jobs in such a way that the resource usage of each individual resource type does not change much over time. In the end, we want to hire as few workers as possible and pay as little as possible for them being idle. In §4 we introduce concrete measures for the quality of leveling.

A feasible schedule consists of an allocation of resource units to jobs and a feasible temporal planning of jobs with respect to given precedence constraints and working shifts. Ideally, we would like to minimize both the project duration and its cost. However, there is the trade-off mentioned above; fast project executions cause high cost, whereas cheap project executions take a long time.

Determining a good project duration depends on several aspects that need to be balanced against each other. These include the total resource cost for hiring resource units, the total production loss caused by the shutdown during the turnaround period, and a "risk cost" because of unexpected repairs and delays that are inherent in maintenance jobs and tend to become more influential the more ambitious, i.e., the shorter the project duration. Let us neglect the risk cost for a moment. Then, for a given production loss per time unit, one would ideally like to find a schedule that minimizes total cost, i.e., the sum of out-of-service cost and the cost for hiring resource units over the turnaround period. If the out-of-service-cost cannot be quantified, then the manager defines a deadline for the turnaround; our goal is to find a feasible schedule of minimum resource cost that meets this deadline.

However, risk issues cannot be neglected in practice—in particular, not in turnaround projects that contain many maintenance jobs that may cause unforeseen repair work or, even worse, delay the completion of the job until, say, the delivery of a spare part. Thus the deadline for a turnaround can only be met with a certain probability that tends to decrease with an ambitious deadline. Information on the risk involved with a decision about the length of a turnaround is crucial to a manager. Fortunately, companies typically have stochastic information based on experiences with earlier turnaround projects. This information permits a stochastic evaluation of the risk of the computed schedule with respect to (w.r.t.) meeting the deadline. The risk measures we use are (i) the expected tardiness of a schedule w.r.t. the chosen deadline and (ii) the probability distribution of the project duration. Computing these measures is #P-complete in general, which makes efficient computations unlikely. However, for problems without shifts and with unlimited resource units, we can apply known techniques to determine an upper bound on the expected tardiness for a given project schedule as a function of its completion time. Interestingly, the computation of this function is algorithmically strongly related to the algorithm we use to determine the deterministic trade-off between project duration and project cost. Our stochastic evaluation of relevant schedules enables the project manager to select a particular schedule according to his or her own risk preferences.

1.1. Our Contribution

In this paper, we first introduce the problem of turnaround scheduling and give an overview on the large variety of related scheduling problems. We then develop a two-phase model for turnaround scheduling and present techniques to solve the individual phases. The first phase supports the project manager in finding a good project duration t observing his or her risk preferences, and the second phase optimizes the use of resources for the chosen duration t. A case study with real-world turnaround scheduling problems carried out in cooperation with the management consulting company T.A. Cook Consultants and two of their customers at chemical manufacturing sites shows that our methods yield provably good solutions to large-scale turnaround problems in a very short time.

The two-phase approach that we implemented serves as a decision support tool. In the first phase, the *strategic planning phase*, a project manager has to decide on the desired makespan for the turnaround project and has to quantify the number of workers and other resources available for the project. To support this decision process, we provide an approximation of the trade-off between project duration and cost as well as a stochastic evaluation of the risk for meeting the makespan. The second phase, the *detailed planning phase*, is then a combination of resource allocation and leveling for the chosen deadline that we solve heuristically. We complement this by a risk analysis of the computed detailed schedule that provides upper bounds for the risk measures "expected tardiness" and "probability of meeting the project duration."

Our methods can handle real-world instances with 100,000 to 150,000 jobs from chemical manufacturing plants within a few minutes and yield solutions with a leveled resource consumption. To evaluate the performance of our methods, we compare our solutions with optimal solutions for problems with up to 50 jobs that are computed by solving a mixed-integer program (MIP) formulation of the turnaround problem that includes all the deterministic features such as different shifts, variable resource allocation, resource leveling, and complex precedence constraints. Our MIP is time indexed and thus is much too large for typical problem sizes of turnarounds. In contrast, our heuristic algorithm is fast and produces solutions of good quality as the comparison with the MIP shows.

To the best of our knowledge, this is the first time that the turnaround scheduling problem is treated with this combination of optimization techniques. The interest of our cooperation partner T.A. Cook Consultants in these methods has led to a commercial initiative in which a software company integrated our methods as add-ons to Microsoft Office Project.

2. Related Work

The turnaround scheduling problem has many relationships with well-established areas of scheduling.

Time-Cost Trade-off. Given a project network of jobs and precedence constraints, a job may be executed in different modes, each associated with a certain processing time and resource requirement. The *time-cost trade-off problem* asks for the relation between the duration of a project and its cost, which is determined by the amount of nonrenewable resource units necessary to achieve the project duration. For a nice survey, we refer to De et al. (1995). Fixing either the project duration or the cost leads to the closely related optimization problems with the objective to minimize the other parameter; these problems are referred to as the *deadline problem* and the *budget problem*, respectively.

When the resource costs for the jobs are continuous linear nonincreasing functions of the job-processing times, then the deadline and the budget problem can be solved optimally in polynomial time as has been shown independently by Fulkerson (1961) and Kelley (1961). Later, Phillips and Dessouky (1977) gave an improved version of the original algorithms in which iterative cut computations in a graph of critical jobs yield the piecewise linear time-cost trade-off curve that describes the trade-off between project duration t and associated cost for all t. The running time is polynomial in the number of breakpoints of the optimal time-cost trade-off curve, which may, however, be exponential in the input size (see Skutella 1998b). (A breakpoint of such a piecewise linear function is a point in which the function is continuous but not differentiable.) Elmaghraby and Kamburowski (1992) generalized previous algorithms to solve a more general problem variant in which jobs may have release dates and deadlines and arbitrary time lags between them. They provided a combinatorial algorithm that iteratively computes minimum cost flows in an iteratively transformed network modeling the time lags. Other cost functions have been considered such as convex (Lamberson and Hocking 1970, Kapur 1973, Siemens and Gooding 1975, Adel and Elmaghraby 1984) and concave function (Falk and Horowitz 1972).

In practical applications, the discrete version of this problem plays an important role. Here, the processing time of a job is a discrete nonincreasing function of the amount of the renewable resource allocated to it. This problem is known to be $\mathcal{NP}$-hard (see De et al. 1997). Skutella (1998a) derived approximation algorithms for the deadline and budget problem as well as bicriteria approximations, and Deineko and Woeginger (2001) gave lower bounds on the approximability of the problems.

Various exact algorithms and metaheuristics have been implemented for the discrete time-cost trade-off problem. For an overview, we refer to Chapter 8 in the book by Demeulemeester and Herroelen (2002).

Time-Cost Trade-off with Capacity Constraints. Motivated by restrictions on resource capacities in real-world applications, the time-cost trade-off problem has been investigated in problem variants with renewable as well as nonrenewable resource types of limited capacity. Such problems are also known as *multimode* (*resource-constrained*) *project scheduling problems* (see, e.g., Demeulemeester and Herroelen 2002).

Various versions of linear and discrete time-cost trade-off–related problems have also been considered in the theory of machine scheduling. Besides the available machines, there is an additional resource that allows us to accelerate the processing of jobs. Approximation algorithms and even polynomial-time approximation schemes have been derived. Reviewing these results in detail is beyond the scope of this paper. We only mention scheduling with *controllable processing times*, which concerns the allocation of nonrenewable resource units (see the recent survey of Shabtay and Steiner 2007), scheduling jobs with *resource-dependent processing times*, which assumes a discrete renewable resource (see Grigoriev et al. 2007 and references therein), and scheduling *malleable jobs* on one or several machines, where the duration of a job is determined by the number of machines allocated to it (see, e.g., Du and Leung 1989, Lepère et al. 2002, and Jansen and Zhang 2006).

Resources with Calendars and Working Shifts. In real-world applications, resources are rarely continuously available for processing. Working shifts, machine maintenance, or other constraints may prohibit the processing in certain time intervals. Also, in machine scheduling, various problems with limited machine availability have been considered, and we refer to the survey by Schmidt (2000) for complexity and approximation results.

In project scheduling, such constraints are known as break calendars or shift calendars. Zhan (1992) provides an exact pseudopolynomial algorithm for computing earliest and latest start times in a generalized activity network that may contain minimum and maximum time lags but no capacity bounds on the resource types. His modified label-correcting algorithm respects jobs that may be preempted and those that must not. This algorithm has been modified into a polynomial-time algorithm by Franck et al. (2001). In the same paper, they also provide priority rule-based heuristics for solving resource-constrained project scheduling problems, where each job may require different resource types.

Yang and Chen (2000) consider a job-based, more flexible version of calendars represented by *time-switch constraints*. These constraints specify for any job several time windows in which the job may be processed. They also extend the classical critical path method to analyze project networks when resource capacities are unbounded. Time-switch constraints have also been incorporated by Vanhoucke et al. (2002) in the deadline version of the discrete time-cost trade-off problem to model different working shifts. They present a branch-and-bound algorithm that was later improved by Vanhoucke (2005). Experimental results were shown for instances with up to 30 jobs and up to 7 different processing modes. Recently, Vanhoucke and Debels (2007) investigated the deadline problem with time-switch and other side constraints with the objective to minimize the net present value.

Resource Leveling. Typical goals in project management are the minimization of the total project duration (makespan), the maximization of net present value or more service-oriented goals such as minimizing waiting time or lateness. In certain applications, the objective functions are based on resource utilization (see, e.g., Neumann et al. 2003). In particular, when resource units are rented for a fixed time period, then they should be utilized evenly over this time.

Harris (1990) developed a critical path-based heuristic for resource leveling of precedence-constrained jobs with fixed processing times and no side constraints. Neumann and Zimmermann (2000) presented an heuristic and exact algorithms for the resource-leveling problem with temporal and resource constraints. In a number of earlier publications, such as Bourges and Killebrew (1962), Easa (1989), Phillips and Garcia-Diaz (1981), and Bandelloni et al. (1994), heuristics and exact algorithms for simplified problem versions can be found.

Considering a variant of interval scheduling, Cieliebak et al. (2004) aim for minimizing the maximum number of used resource units. They derive approximation algorithms and hardness results.

Stochastic Analysis of Project Networks. The importance of dealing with uncertainty in scheduling is reflected by the large number of publications on various aspects of this topic. These results are mostly restricted to problems without resource constraints and without shift calendars. Several methods have been developed for analyzing the makespan $C_{\max}$ in project networks with random processing times, e.g., bounding the expected makespan or its distribution function. An exact computation of the makespan distribution—even just a single point of this function—is, in general, a #P-complete problem (as shown by Hagstrom 1988) that presumably rules out its efficient computation. For a general overview, we refer to Adlakha and Kulkarni (1989); for a survey on methods for bounding the makespan distribution, we refer to Möhring (2001). An experimental study comparing the performance of various such methods has been pursued by Ludwig et al. (2001).

In particularly, we want to mention the works of Meilijson and Nádas (1979) and Weiss (1986). They consider jobs with stochastically dependent processing times and determine an upper bound on the expected tardiness $\mathbb{E}[\max\{t - C_{\max}, 0\}]$ for a given project schedule with makespan $C_{\max}$ as a function of the completion time t (in the model without resource constraints and calendars). Interestingly, the computation of this bound is strongly related to solving a linear time-cost trade-off problem.

3. Problem Description

In turnaround scheduling we are given a set $\mathcal{J}$ of n jobs and a set $\mathcal{R}$ of renewable resource types. Each job needs a finite number of resource units of exactly one resource type $k \in \mathcal{R}$ for its processing.

Processing alternatives of job $j \in \mathcal{J}$ are characterized by the number r_j of allocated resources and its resulting processing time $p_j(r_j)$. We assume that r_j is integral and bounded from below and above by $r_j^{\min}$ and $r_j^{\max}$, respectively. Let $\mathcal{J}_k \subseteq \mathcal{J}$ denote the set of jobs that requires resource type $k \in \mathcal{R}$. Because each job requires exactly one resource type, we can partition the set of jobs $\mathcal{J}$ into disjoint subsets $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_{|\mathcal{R}|}$.

The amount of work for processing a job $j \in \mathcal{J}$ is given by $w_j(r_j) = r_j \cdot p_j(r_j)$. We assume that the processing time is nonincreasing and the work is nondecreasing in the number of resource units. Thus the *monotonicity properties*

$$p_j(r_1) \geq p_j(r_2) \quad \text{and} \quad w_j(r_1) \leq w_j(r_2)$$

hold for any r_1, r_2 with $r_j^{\min} \leq r_1 \leq r_2 \leq r_j^{\max}$. We denote each processing alternative for a job $j \in \mathcal{J}$ given by the resource allocation r_j as a *mode* of job j and denote the set of feasible modes for job j by $\mathcal{M}_j$. Thus each feasible mode for job j defines a tuple (r_j, p_j) of allocated resource units r_j and associated processing time p_j. Furthermore, each job $j \in \mathcal{J}$ has associated a release date $s_j \in \mathbb{Z}^+$ and a due date $d_j \in \mathbb{Z}^+$ that define the time window $[s_j, d_j[$ in which j must be processed. Due dates are quite rare in our application, but they generalize the model, and they turned out to be useful for the project planner. Precedence constraints are given by a directed acyclic graph $G = (V, E)$, where the vertices correspond to jobs and there is an edge $(i, j) \in E$ if job i precedes job j (see the end of this section for generalized precedence constraints).

A vector of processing times $p = (p_1, \ldots, p_n)$ is a *feasible realization* if for each $j = 1, \ldots, n$ there is a resource allocation $r_j \in \{r_j^{\min}, \ldots, r_j^{\max}\}$ such that $p_j = p_j(r_j)$. If the resource allocation r_j is clear from the context, we will simply write p_j instead of $p_j(r_j)$ and w_j instead of $w_j(r_j)$.

A *schedule* for a turnaround problem is given by a pair (S, r), where $r = (r_1, \ldots, r_n)$ with $r_j \in \{r_j^{\min}, \ldots, r_j^{\max}\}$ for each job j is a vector of feasible resource allocations, and S is a vector $S = (S_1, \ldots, S_n)$ of start times for the jobs. A schedule (S, r) is *time feasible* if it respects the release dates, due dates, and precedence constraints; i.e.,

$$S_i + p_i \leq S_j \quad \text{for all } (i, j) \in E$$

and

$$s_j \leq S_j \leq S_j + p_j \leq d_j \quad \text{for all } j \in \mathcal{J}.$$

The maximum completion time of all jobs, the *makespan*, is denoted by

$$C_{\max}(S, r) := \max_{j \in \mathcal{J}} \{S_j + p_j\}.$$

For a time-feasible schedule (S, r), we denote

$$r_k(S, r, t) := \sum_{j \in \mathcal{J}_k : S_j \leq t < S_j + p_j} r_j$$

as the *resource utilization* of resource type $k \in \mathcal{R}$ at time t, and

$$R_k := \max_t r_k(S, r, t) \quad \text{for all } k \in \mathcal{R}$$

as the *maximum resource utilization* of resource type $k \in \mathcal{R}$, i.e., the maximum number of resource units of type k utilized at any time during the turnaround. The maximum resource utilization of a resource type k may be bounded by a constant $\bar{R}_k \in \mathbb{Z}^+$, which we call the *capacity* of that resource type.

Each resource type $k \in \mathcal{R}$ has an individual *calendar* of working shifts, which represents the *availability periods* of k. Jobs cannot be interrupted; thus they need to be executed during one availability period. Given a project deadline T, we define the actual *working time* $T_k \leq T$ of resource type $k \in \mathcal{R}$ as the total time that resources units of type k are available in the time interval $[0, T]$.

We call a schedule *resource feasible* w.r.t. resource capacities and calendars if for any resource type $k \in \mathcal{R}$, the capacity bounds are respected, $R_k \leq \bar{R}_k$, and each job is executed during one of the availability periods of the corresponding resource type k.

For each $k \in \mathcal{R}$, we are given a cost rate c_k that represents the cost per unit of resource type k per time unit. The set of resource types is partitioned into two disjoint subsets, $\mathcal{R}^l$ and $\mathcal{R}^f$, depending on their payment type. Resources of type $k \in \mathcal{R}^l$ have to be paid during the entire turnaround period for the maximum amount needed. These are mainly external workers who are hired for the complete turnaround period, and they must be paid for even if they are temporarily idle. Clearly, the goal of a project manager is to minimize the amount of paid idle time. In other words, the maximum resource utilization of those resource types should be minimized. We say that these resource types *need to be leveled*. In contrast, resource types from set $\mathcal{R}^f$ are paid for the actual work they perform. We say they are *free of leveling*. In our application, these are mainly internal resource types. The job sets corresponding to $\mathcal{R}^l$ and $\mathcal{R}^f$ are denoted by $\mathcal{J}^l$ and $\mathcal{J}^f$, respectively.

Now, we can express the total cost of a schedule (S, r) as

$$\sum_{k\in\mathcal{R}^l} c_k \cdot R_k \cdot T_k + \sum_{k\in\mathcal{R}^f}\sum_{j\in\mathcal{J}_k} c_k \cdot w_j(r_j).$$

The first term is called the *resource availability cost* and represents the cost of resource types that must be leveled; the second term, called the *resource utilization cost*, represents the cost of jobs that do not need to be leveled.

The turnaround scheduling task is to find a schedule that is *time feasible* and *resource feasible* and has minimum total cost. In practice, the first term clearly dominates the cost function. If we neglect the cost for jobs that do not need to be leveled, we speak of the *resource-leveling problem*. This is the problem on which we focus in this paper.

For later use, we introduce two additional parameters that are related to the goal of minimizing the resource availability cost. The first parameter indicates how well a single resource type k is leveled and is called the *relative resource consumption* of resource type k and is denoted by ν_k. This parameter is defined for a schedule (S, r) as the total work done by resource type k relative to the maximum resource utilization R_k over the total available working time of resource type k; i.e.,

$$\nu_k := \frac{\sum_{j\in\mathcal{J}_k} w_j}{T_k \cdot R_k}. \tag{1}$$

The second additional parameter μ_k is very useful within the resource-leveling algorithm itself when we need to identify a resource type that is badly leveled and causes high cost. We call it the *relative idleness cost* μ_k and define it as the cost for the wasted available but not utilized work volume over the available work volume; i.e.,

$$\mu_k := (1 - \nu_k) \cdot c_k. \tag{2}$$

Next, we present a mixed-integer programming formulation of the turnaround scheduling problem for a given project deadline T. It borrows from the classical time-indexed formulation based on start times for resource-constrained project scheduling by Pritsker et al. (1969) and incorporates the multimode characteristics of jobs. For an overview on models and notation in resource-constrained project scheduling, we refer to Brucker et al. (1999). We use binary decision variables x_{jlt} that indicate whether job $j \in \mathcal{J}$ starts in mode $l \in \mathcal{M}_j$ at time $t \in \{0, 1, \ldots, T-1\}$.

We model resource calendars implicitly using start-time-dependent processing times. In a preprocessing step we compute for each job $j \in \mathcal{J}$ the processing time p_{jlt} when the job starts at time t in mode $l \in \mathcal{M}_j$. If a job cannot be scheduled with respect to calendars at time t, we set the corresponding variable x_{jlt} to zero. The resource requirements of job j in mode l is denoted by r_{jl}. The formulation is as follows:

$$\min \sum_{k\in\mathcal{R}} c_k \cdot R_k \cdot T_k$$

$$\text{s.t.} \sum_{l\in\mathcal{M}_j}\sum_{t=0}^{T-1} x_{jlt} = 1 \quad \forall\, j\in\mathcal{J}, \tag{3}$$

$$\sum_{l\in\mathcal{M}_j}\sum_{t=0}^{T-1} t\cdot x_{jlt} - \sum_{l\in\mathcal{M}_j}\sum_{t=0}^{T-1}(t+p_{ilt})\cdot x_{ilt} \ge 0 \quad \forall\,(i,j)\in E, \tag{4}$$

$$\sum_{j\in\mathcal{J}_k}\sum_{\tau=0}^{t}\sum_{l\in\mathcal{M}_j} r_{jl}\cdot x_{jl\tau}\cdot \mathbb{1}_{\{\tau+p_{jl\tau}>t\}}(\tau) \le R_k \quad \forall\, k\in\mathcal{R},\ t=0,\ldots,T-1, \tag{5}$$

$$0 \le R_k \le \bar{R}_k \quad \forall\, k\in\mathcal{R}, \tag{6}$$

$$x_{jlt}\in\{0,1\} \quad \forall\, j\in\mathcal{J}, l\in\mathcal{M}_j, t=0,\ldots,T-1.$$

Constraint (3) assures that each job starts exactly once in one of its modes. Because of (4), every two jobs i, j respect the precedence constraints.

Finally, inequalities (5) and (6) guarantee that the capacity constraints are met.

Because of the time expansion, time-indexed formulations for scheduling problems are usually hopeless for large problem instances. However, small instances can be solved using integer programming solvers such as CPLEX, and we can thus evaluate the performance of our algorithm by comparing the computed solution with an optimal solution for such instances.

We conclude this section with a remark on further so-called *generalized precedence constraints* that are important in some practical turnaround problems. Our algorithms can handle these requirements, but we do not elaborate on this in the present paper.

Constraints occurring in our applications are as follows:

- *Fixed start times:* A job $j \in \mathcal{J}$ must start at its release date $S_j = s_j$.
- *Parallel sets:* Two jobs $i, j \in \mathcal{J}$ must be executed in parallel for the processing time of the shorter job. Without loss of generality we assume that $p_i(r_i) < p_j(r_j)$. The condition is fulfilled if $S_i \ge S_j$ and $S_i + p_i(r_i) \le S_j + p_j(r_j)$.
- *Forbidden sets:* Two jobs $i, j \in \mathcal{J}$ that form a forbidden set must not be executed in intersecting time intervals. This is expressed by the following condition $S_j \ge S_i + p_i(r_i)$ or $S_i \ge S_j + p_j(r_j)$.
- *Zero maximum finish-start time lags:* Job j has to be executed immediately after the completion of job i; i.e., $S_j = S_i + p_i(r_i)$ must hold.

- *Zero maximum start-start time lag:* Two jobs $i, j \in \mathcal{J}$ have to start at the same time $S_i = S_j$.

4. Solution Methods

We implemented a two-phase solution method for turnaround scheduling problems that serves as a decision support tool. In the following subsections, we give a brief outline of the method and describe it in more detail.

Phase I. We compute a time-cost trade-off curve that provides the approximate cost for any possible choice of duration T for the turnaround. To this end, we relax the integrality of the resource usage and neglect the resource capacities and working shifts. Then we solve a linear time-cost trade-off problem with time windows to optimality. Finally, we apply an heuristic scaling technique to approximate the true costs and to compute associated feasible schedules. This includes computing job modes for every T that results from scaling the breakpoints of the linear time-cost trade-off curve.

Based on that curve (and information about the risk involved; see below), the decision maker chooses a particular makespan for the turnaround duration. This may be fixed by the decision maker based on his or her risk aversion and other preferences but could also be the result of minimizing the total cost when out-of-service costs are available. The job modes computed for the chosen makespan are the basis for the second phase.

Phase II. In this phase, we solve the actual turnaround scheduling problem with all side constraints for the turnaround duration chosen in the first phase. We determine feasible start times for all jobs and adjust the resource allocation such that the temporal unavailabilities of resources as well as the given deadline T are respected. We find a feasible schedule with a resource profile that is leveled over the project duration, i.e., a schedule with minimized resource availability cost.

Stochastic support. The decision-making process of the user is supported in both phases by a risk analysis of the respective solutions. We estimate the expected tardiness of relevant schedules for a deadline T and the probability of meeting it. In the first phase, this is done for each schedule corresponding to a breakpoint of the (relaxed) time-cost trade-off curve, and in the second phase, it is done for the final schedule after resource leveling. We complement the expected tardiness of a schedule with confidence intervals that tell the project manager how likely that certain project deadlines will be met.

In the following sections, we describe the two main phases of our algorithm and the stochastic analysis in detail.

4.1. Phase I—The Time-Cost Trade-off Curve

The trade-off between project duration and project cost can be represented as a so-called *time-cost trade-off curve*. For each possible project duration the curve provides the minimum cost. Such a curve can guide a manager when making a decision on a project deadline. Clearly, computing the exact curve is computationally intractable for a complex turnaround problem; however, an approximate curve is sufficient for decision support.

In this phase, we do not consider the resource availability cost as defined in §3. Instead, we compute the actual resource utilization cost without the additional cost for idle times when renting and leveling resources. The reason is that the cost of idle times is very sensitive to small changes in the schedule that occur only at project execution. Therefore, at this stage, when a project manager must decide on a project duration, he or she is interested only in the trade-off between project duration and resource utilization cost and does not consider the cost for idle times.

To compute such an approximate time-cost curve, we first compute an optimal curve for a relaxed problem and turn it into a feasible solution using rounding and scaling techniques. To help decide on the project duration, we add the out-of-service-cost per time unit and thus determine a minimum point of the resulting curve. The general outline of the procedure is summarized below, followed by a detailed description. Figure 1 visualizes the effects of the procedure and the curves.

Algorithm 1 (Time-cost trade-off algorithm with scaling)

```
Input: Turnaround scheduling instance.
Output: Approximate time-cost trade-off curve
    with respect to resource availability cost.
1 Compute the optimal time-cost trade-off curve
      for the linearized problem version without
      resource constraints (calendars, capacities).
2 for each breakpoint p of the curve do
3     Round up fractional resource assignments
          to the nearest integral value $r_j := \lceil r_j \rceil$.
4     Apply list scheduling heuristics that respect
          calendars and capacity constraints.
5     Scale point p and apply dominance rules.
```

In the first step we relax our problem. We linearize the job modes $\mathcal{M}_j$ and allow nonintegral resource allocations r_j. We convert the modes of each job into a linear function by linear interpolation. This cost function defines the dependency between processing time and cost. Furthermore, we assume that all resource types are available continuously, and we do not level the resource usage. Then the problem reduces to assigning a (possibly nonintegral) resource consumption r_j,

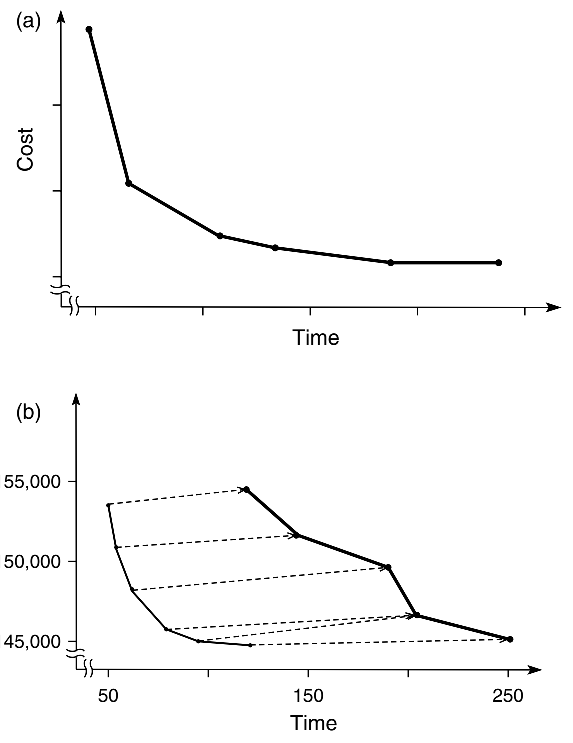

Figure 1 Time-Cost Trade-off Curves

Notes. The optimal solution for the linear time-cost trade-off problem for a relaxation of the original turnaround problem is shown in (a). The optimal solution of the relaxation is turned into an approximate solution for the original turnaround problem by applying heuristics at the breakpoints in (b). The time is given in hours and the cost is given in euros. Adding the out-of-service cost to the cost function yields a cost optimal makespan and serves together with risk analysis to support managerial decision making.

and thus a processing time p_j, to each $j \in \mathcal{J}$ and finding a time-feasible schedule of minimum resource utilization cost. This is the classical time-cost trade-off problem with additional release dates and deadlines (time windows) for jobs.

This problem without time windows can be solved optimally in polynomial time in the input and the number of breakpoints of the curve (Fulkerson 1961, Kelley 1961, Phillips and Dessouky 1977). In fact, most of our real-world instances do not require the more sophisticated and time-consuming algorithm that respects also time windows by Elmaghraby and Kamburowski (1992).

The optimal time-cost curve for the relaxed problem is a lower bound on the optimal time-cost curve for the trade-off problem including all other side constraints.

Now, the schedules associated with points on the time-cost trade-off curve need not be feasible. Usually, resource assignments are nonintegral and resource calendars are not respected. To obtain a cost curve respecting these conditions, we consider all breakpoints T^i of the relaxed curve and turn the corresponding schedules into feasible schedules. Interpolation between the cost of these schedules then gives the new approximate-cost curve.

For a particular T^i, we round up a nonintegral resource allocation r_j to the nearest integer. This increases the resource utilization cost but also guarantees that we do not exceed the job completion times from the linear relaxations. Once all jobs $j \in \mathcal{J}$ have integral values r_j, we greedily try to decrease the resource allocation and thus the cost without violating the deadline T^i.

Furthermore, schedules must be adapted to respect the given calendars. This will usually result in job deferrals and increased cost, but this way, we obtain feasible solutions. We apply simple but fast list-scheduling heuristics that reschedule with respect to the resource availability (timewise and capacitywise). We refer to this as scaling a breakpoint T^i and the associated schedule. Within this heuristic approach, we may turn two infeasible schedules (S_1, r_1) and (S_2, r_2) with makespans $C_{\max}(S_1, r_1) > C_{\max}(S_2, r_2)$ and costs $cost(S_1, r_1) < cost(S_2, r_2)$ into two feasible schedules (S_1', r_1') and (S_2', r_2') with costs $cost(S_1', r_1') < cost(S_2', r_2')$ but with makespans $C_{\max}(S_1', r_1') < C_{\max}(S_2', r_2')$. In that case, schedule (S_2', r_2') is dominated by (S_1', r_1'), and therefore, we do not store (S_2', r_2'). The scaled time-cost trade-off curve is obtained by interpolation between the points obtained after scaling and applying this dominance rule; see Figure 1(b). The difference between the linearized curve and the scaled curve obviously depends on the resource calendars and the resource capacities. In our real-world instances the project durations are scaled by a factor between 2 and 3, whereas the costs hardly differ.

The resulting curve is an approximation of the optimal time-cost trade-off curve where the computed costs in the breakpoints are an upper bound for the optimal resource utilization cost. We guarantee that each point of the curve corresponds to a feasible schedule respecting all constraints. However, the resulting curve need not be convex anymore; see Figure 1(b).

Although resource calendars and precedence constraints are considered, leveling the resource usage over time has not yet been addressed. This objective is taken care of in the next phase when a project deadline is fixed. In the current stage, the approximated time-cost trade-off curve together with the stochastic analysis as described in §4.3 guides a turnaround manager in his or her decision on the project duration T.

4.2. Phase II—Resource Leveling and Detailed Scheduling

We enter the second phase with a maximum project duration T and a feasible choice of processing times $p^\star$ (with corresponding resource consumptions $r_j^\star$ for $j \in \mathcal{J}$) given by the chosen approximate solution of the first phase. Whereas T remains fixed, the choice of job modes $(r_j^\star, p_j^\star)$ serves solely as a starting point for solving the resource-leveling problem. We construct a schedule (S, r) that is time feasible and resource feasible and minimizes the resource availability cost $\sum_{k\in\mathcal{R}^l} c_k \cdot R_k \cdot T_k$. In Figures 4 and 5 in §5 we illustrate the difference between a schedule with temporary high-resource consumptions and a schedule with evenly used resource types that causes lower resource availability cost based on real turnaround data.

Our solution approach focuses on the minimization of the resource availability cost with respect to the feasibility of a schedule. To that end, we combine binary search on capacity bounds with different list-scheduling procedures. Recall that only resource types $k \in \mathcal{R}^l$ need to be considered for leveling.

We compute initial lower and upper bounds, LB_k and UB_k, respectively, on the maximum resource utilization R_k for each resource type $k \in \mathcal{R}^l$. A lower bound is given by the minimum resource requirement of each job and by the minimum total workload divided by the working time of the corresponding resource. More formally,

$$R_k \geq \mathrm{LB}_k = \max\left\{\max\{r_j^{\min} \mid j \in \mathcal{J}_k\}, \sum_{j\in\mathcal{J}_k} \frac{r_j^{\min} \cdot p_j(r_j^{\min})}{T_k}\right\}.$$

The upper bounds UB_k for $k \in \mathcal{R}$ might be part of the input; otherwise, we compute an earliest start schedule (S, r) without limitations in the resource availability and use the resulting maximum resource utilization as upper bounds; i.e., $\mathrm{UB}_k := \max_t r_k(S, r, t)$.

In our algorithm we iteratively choose a resource type $k \in \mathcal{R}^l$ that is badly leveled, meaning that a large fraction of the total availability cost is spent on idle times of resource type k with respect to its current bound UB_k. In §3 we therefore introduced the measure of relative idleness cost (2) for each resource type k depending on the maximum resource utilization R_k. At this stage of the algorithm, the relative idleness cost is defined based on the current upper bound UB_k on the maximum resource utilization R_k, i.e., with a slight abuse of notation,

$$\mu_k = \frac{T_k \cdot \mathrm{UB}_k - \sum_{j\in\mathcal{J}_k} w_j(r_j)}{T_k \cdot \mathrm{UB}_k} \cdot c_k.$$

In each iteration, we choose a resource type $k^\star$ with maximum relative idleness cost, $\mu_{k^\star} = \max_{k\in\mathcal{R}^l} \mu_k$, and try to decrease the upper bound $\mathrm{UB}_{k^\star}$ on its capacity while all other resource capacities remain fixed. We aim at decreasing $\mathrm{UB}_{k^\star}$ to $\alpha \cdot \mathrm{UB}_{k^\star} + (1-\alpha) \cdot \mathrm{LB}_{k^\star}$ for some $0 < \alpha < 1$. An upper bound can be decreased to a value u if we find a time- and resource-feasible schedule for the given total project duration T that utilizes at no time more than u units of resource type $k^\star$. We verify this property heuristically by using list-scheduling procedures. Each of these procedures considers a different ordering (list) of jobs by which jobs are inserted into a partial schedule. With respect to the given precedence constraints, it is desirable to place jobs according to a topological ordering. Such orderings can be obtained by forward and backward computations of earliest start dates, earliest completion times, latest start dates, and latest completion times of the jobs with respect to the given shifts and the given makespan T. We also use lists that have a random switch; i.e., the beginning of the list up to a randomly chosen position is sorted by increasing earliest start dates, whereas the jobs after that position are sorted by increasing latest completion times. We use five different such lists.

If the list-scheduling procedures do not yield a feasible schedule, we find the lowest upper bound that allows a feasible schedule by binary search. If an upper bound UB_k cannot be decreased in this way, we do not consider resource type k for leveling anymore and set its lower bound to $\mathrm{LB}_k = \mathrm{UB}_k$.

Notice that in this procedure, we do not aim at decreasing one particular badly leveled resource type immediately down to its lowest utilization limit. Instead, we consider all badly leveled resource types in a round-robin fashion and decrease their utilization in each round to a certain fraction of the previous bound. The idea behind this is to balance the effects that the decreases in utilization of different resource types have on each other. Experiments have revealed that it is beneficial to focus not only on a single resource type but on all of them, one after another. The relative idleness cost μ_k steers the prioritization of resource types within the round-robin selection.

A more formal description of our algorithm is as follows.

Algorithm 2 (Resource leveling)

```
Input: Set $\mathcal{R}^l$ of resource types that must be
    leveled, set $L$ of topologically sorted
    lists of jobs, maximum total project
    duration $T$, and parameter $\alpha \in (0, 1)$.
Output: Leveled resource utilization $R_k$
    for each resource type $k \in \mathcal{R}^l$.
1 Set $\mathrm{LB}_k$ and $\mathrm{UB}_k$ to initial values
      for each resource type $k \in \mathcal{R}^l$.
2 while $\exists\ k \in \mathcal{R}^l$: $\mathrm{LB}_k < \mathrm{UB}_k$ do
3     Choose resource type $k^\star \in \mathcal{R}^l$ with
          $\mathrm{LB}_{k^\star} < \mathrm{UB}_{k^\star}$ and $\mu_{k^\star}$ maximum.
4     Set a temporary upper bound
          $u_{k^\star} := \alpha \cdot \mathrm{UB}_{k^\star} + (1-\alpha) \cdot \mathrm{LB}_{k^\star}$.
5     for each list in $L$ do
6         if List scheduling yields a feasible schedule
              with makespan at most $T$ then
7             Set $\mathrm{UB}_{k^\star} := u_{k^\star}$.
```

```
8             goto Step 3.
9     // if List scheduling failed for each list:
10    Set $\mathrm{LB}_{k^\star} := u_{k^\star}$.
11    **if** $\mathrm{LB}_{k^\star} = \mathrm{UB}_{k^\star}$ **then**
12        $R_{k^\star} := \mathrm{LB}_{k^\star}$.
13        $\mathcal{R}^l := \mathcal{R}^l \setminus \{k^\star\}$.
14    **else**
15        goto Step 4.
```

At the end of §3 we introduced additional, more-complex *generalized* precedence relations between jobs. Our algorithm can handle these constraints: when inserting a job into a partial schedule during the list-scheduling procedure (Step 6), this type of additional constraint can be respected. However, depending on their complexity, these constraints weaken the performance of the list-scheduling heuristics and clearly slow down the computation process. In our real-world instances, these constraints have mainly a common structure. In most cases they involve cranes that are required for a short time in parallel to a long job. It appears that cranes are no bottleneck resource in our instances and they are free of leveling. Therefore we can eliminate those precedence constraints in the resource leveling and handle them in a post-processing step without causing major conflicts. Similarly, we can substitute a group of jobs associated within generalized precedence relations by a single one if their time windows and respective resource capacities are no bottleneck and reinsert them after the resource-leveling procedure. In this way, generalized precedence constraints become rare and the increase in the running time is acceptable.

Another refinement of our list scheduling heuristic concerns the flexibility in the resource utilization and thus the influence on the processing times of jobs. Traditional list-scheduling procedures deal with fixed realizations of processing times. Notice that so far we have only used the unchanged job modes as they were fixed in the first phase of the turnaround decision framework. Certainly, these first choices are based on an optimal solution for a relaxed problem version, and thus they are a good starting point, but we would neglect optimization potential if we kept them fixed. In particular, in such a complex scheduling situation as the turnaround problem, the "wrong" resource allocation for a single job may block the resource unfavorably and prohibit an already feasible solution for a certain resource capacity.

We address this issue by incorporating a local search on the resource allocation r_j of jobs $j \in \mathcal{J}$ into our list-scheduling procedure. If a job j cannot be inserted because its current processing time exceeds the beginning of a nonavailable period of the respective resource for a small amount, then we increase the number of assigned resource units r_j (if feasible), and thus, we decrease the processing time until the job eventually fits feasibly into the schedule. If this is not successful, we recursively include predecessors of j in this search. This refinement may speed up the resource leveling algorithm noticeably. The reason is because Steps 5 and 6 of the binary search find a feasible solution much faster and accept a decrease of the upper bound on the capacity.

4.3. Risk Analysis in Phases I and II

One of the essential features of turnaround scheduling is the fixed upper bound on the makespan T for the whole turnaround. This bound T determines the hiring of external resources, the loss of production of the factory unit undergoing the turnaround, and the shifting of production to other factory units. It is therefore crucial to make a good decision about the makespan T in Phase I and to be aware of possible risks of the actual turnaround schedule computed in Phase II. Such risks are inherent in a turnaround project because of the many maintenance jobs that may involve unforeseen repair activities. Thus overambitious values of T will involve a high risk of exceeding T, whereas values that are too pessimistic will result in an unnecessary loss of production as well as possibly higher external resource costs.

To aid the decision makers in choosing a good makespan T, we have implemented methods for evaluating two risk measures that are used in both phases. These are based on the assumption that the processing times of (some) jobs j are random variables X_j with (roughly) known probability distribution P_j and modal value p_j, where p_j is the deterministic processing time resulting from the given mode (r_j, p_j) of job j (either at a particular breakpoint of the time-cost trade-off curve in Phase I or caused by the fixed mode of the final schedule in Phase II). For computational reasons that will become obvious below, we also assume that the X_j are *discrete* random variables. In our real-world instances, they have up to four values, $p_j^1 < p_j^2 = p_j < p_j^3 < p_j^4$, where p_j^1 denotes an early completion, p_j^2 is the planned processing time, and p_j^3 and p_j^4 model an increase in processing time because of repair work and waiting for spare parts plus repair, respectively. From the theoretical point of view, one can easily imagine more stochastic events, but the restriction to at most four events seems to be reasonable and common in practice. More parameters are considered as unnecessary overhead by our industrial partners, because this information is hard to specify.

The makespan $C_{\max}$ is then a function of the random processing times X_j and is thus also a random variable that depends on the joint distribution P of the job-processing times X_j. Because these are typically stochastically dependent (often with a positive correlation), an exact calculation of percentiles or other values of the distribution function of $C_{\max}$ is not feasible because this is #P-complete even for independent processing times (see Hagstrom 1988). We therefore compute worst-case measures that are valid for arbitrary dependencies among jobs.

The first such risk measure is an upper bound ψ on the expected tardiness of the makespan, which is defined as

$$\psi(t) := \sup_{Q:\, Q_j = P_j} \mathbb{E}_Q[\max\{C_{\max} - t, 0\}],$$

where Q ranges over all joint distributions of the processing times whose marginal distributions Q_j equal the given distributions P_j of X_j. Thus $\psi(t)$ is an upper bound on the expected time by which $C_{\max}$ will exceed the value t as a function of t.

This bound has been extensively investigated by Meilijson and Nádas (1979) and Weiss (1986). As a function of t, $\psi(t)$ is convex decreasing with a slope of -1 for all t not exceeding a particular value t_0 and can for all $t \geq t_0$ be computed as

$$\begin{aligned} \psi(t) = \min_{(x_1,\ldots,x_n)} & \sum_{j \in \mathcal{J}} \mathbb{E}[\max\{X_j - x_j, 0\}] \\ \text{s.t.} \quad & C_{\max}(x_1, \ldots, x_n) \leq t, \end{aligned}$$

where $C_{\max}(x_1, \ldots, x_n)$ is the makespan resulting from the processing times x_j of jobs j.

This minimization problem is the deadline version of a continuous time-cost trade-off problem in which the cost of job j as a function of its processing time x_j is just the expected tardiness $\mathbb{E}[\max\{X_j - x_j, 0\}]$ of that job and is thus convex and even piecewise linear because the random processing times are discrete. These functions $\mathbb{E}[\max\{X_j - x_j, 0\}]$ are directly obtained from the values $p_j^1 < p_j^2 = p_j < p_j^3 < p_j^4$ and their probabilities, and the standard algorithm for linear time-cost trade-off problems can straightforwardly be adapted to piecewise linear and convex cost functions. Altogether, this yields a very efficient computation of $\psi(t)$ as a function of t.

Meilijson and Nádas (1979) also show that the bound $\psi(t)$ is tight in the sense that for every $t \geq t_0$, there is a joint distribution Q such that $\psi(t) = \mathbb{E}_Q[\max\{C_{\max} - t, 0\}]$. In general, Q may depend on t, but if the precedence constraints form a series-parallel partial order (which is the case in our real-life instances; see §5), then there is such a joint distribution Q attaining the worst-case bound for all t. Moreover, the distribution function F of Q is then given by $F = 1 - \psi$. This implies that

$$\text{Prob}(C_{\max} \leq t) \geq F(t) = 1 - \psi(t),$$

which means that the probability that the makespan $C_{\max}$ does not exceed time t is at least the value $1 - \psi(t)$ for all possible dependencies among jobs. This probability is exactly the second risk measure that we compute, and it can be directly obtained from the function $\psi(t)$.

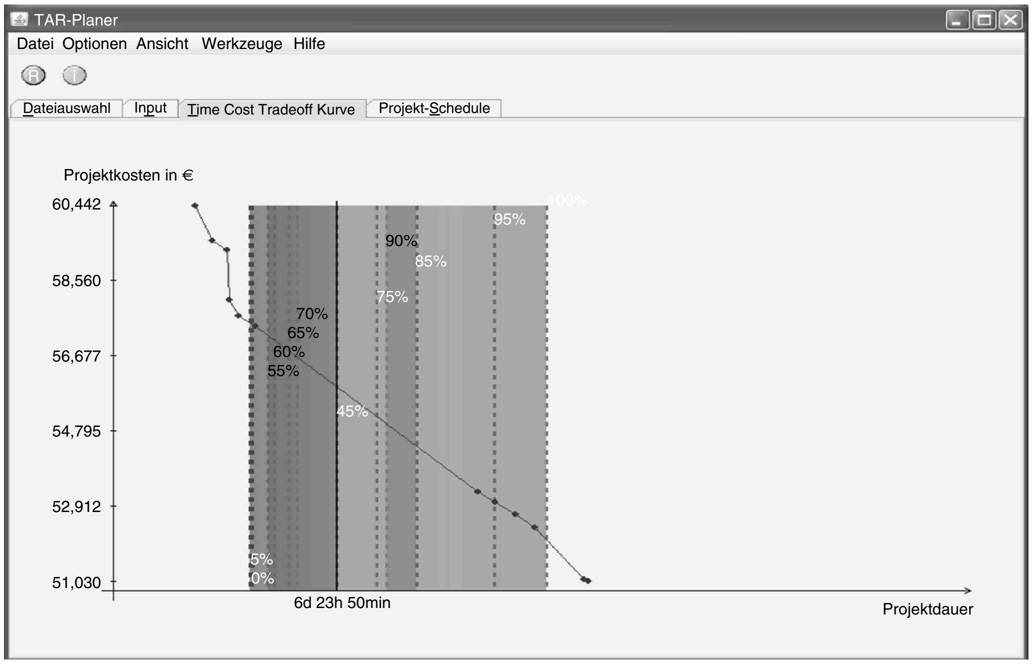

Figure 2 **Evaluation of the Risk in Phase I Illustrated by a Snapshot of Our Software Tool in German**
Notes. The vertical percentage lines refer to the probability that the corresponding time t is met when the turnaround is carried out with the chosen value T. In the example, $T \approx 7$ days, and the probability of meeting T is only 45%.

Altogether, the solution of a continuous time-cost trade-off problem derived from the stochastic information about job-processing times gives us directly two risk measures: the expected tardiness of exceeding the envisioned or chosen makespan T as $\psi(T)$ and the probability of exceeding it as $1 - \psi(T)$. Figure 2 shows the use of the second risk measure in Phase I, where the probabilities of exceeding an envisioned makespan T are indicated by different colors.

5. Computational Results

In this section we report on our computational results obtained on the turnaround scheduling algorithm, that is, the resource-leveling heuristic in §4.2 based on the project manager's decision made after the time-cost trade-off computation described in §4.1. We tested it on three real-world instances from chemical manufacturing sites and on additional smaller instances with similar characteristics that we generated randomly. We evaluate the performance of the real-world instances by the means of the relative resource consumption ν_k, which we introduced as Equation (1) in §3 as an indicator for how well the resource type is leveled. For a large class of artificially generated instances, we can compare the resource availability cost of our heuristic solution with the cost of optimal solutions or lower bounds, both obtained by solving the MIP introduced in §3. All computations were done on a 64-bit Linux machine equipped with a 2.66 GHz Intel Core 2 Duo processor with 2 GB of RAM. To solve the integer programs, we used ILOG CPLEX 11.

The real-world instances consist of about 1,000 jobs, and in one case, 100,000 jobs. They require roughly 15 different resource types, out of which 8 shall be leveled. The processing times per job are widely spread between 20 minutes and 2 days. The precedence relations between jobs form a series-parallel structure that is somewhat dominated by parallel chains; see Figure 3 for a visualization of such a structure for a part of a real-world input instance. A (slightly distorted) example of such a real-world instance can be found in the Online Supplement (available at http://joc.pubs.informs.org/ecompanion.html). The data set consists of three files with job, calendar, and resource parameters for one turnaround project. To imitate a project manager's decision on the total project duration (based on Phase I), we compute the time-cost trade-off curve for a turnaround instance and choose minimum, medium, and maximum feasible project durations. This yields three test instances per input instance to test the resource-leveling heuristic. The chosen durations are between 4 and 15 days for the smaller instances.

We compute solutions for the huge real-world instance in less than 10 minutes, whereas solving the 1,000 job instances takes only a few seconds. The relative resource consumption ν_k of the most important resource types in these examples lies between 95% and 99%. Other less costly resource types yield a lower relative resource consumption; see Figures 4 and 5 for a visualization of the resource consumption of selected resource types before and after the leveling. Nevertheless, this is a high degree of resource utilization, and the precedence constraints between jobs and resource calendars prohibit a much larger resource utilization. To quantify the optimality gap, we consider instances of smaller size for which we can compute an optimal solution or obtain lower bounds from solving the MIP.

We generated such additional turnaround instances by randomly setting the parameters mainly within the bounds given by the real-world instances. We created test sets with 30, 40, 50, and 60 jobs and variable resource allocation of one or two resource units

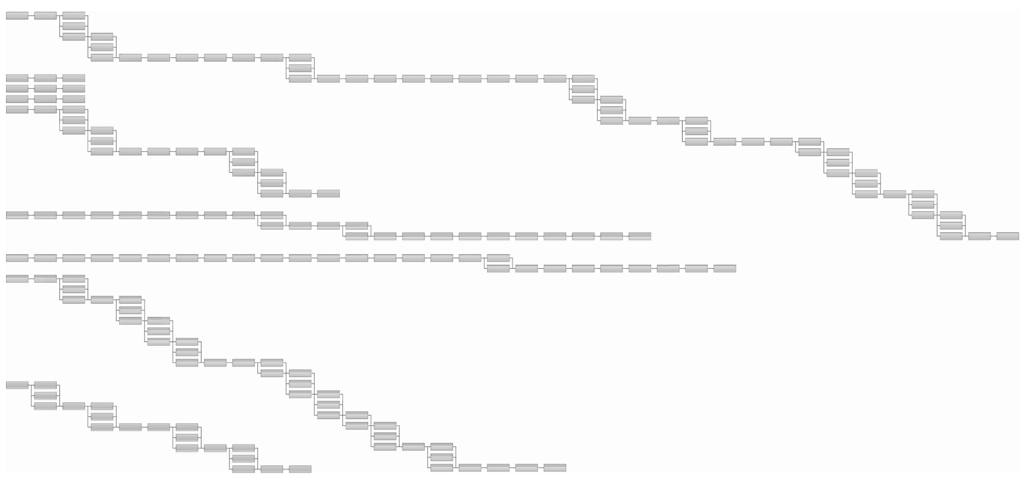

Figure 3 Series-Parallel Structure of Precedence Constraints Between Jobs; Part of a Real-World Turnaround Instance

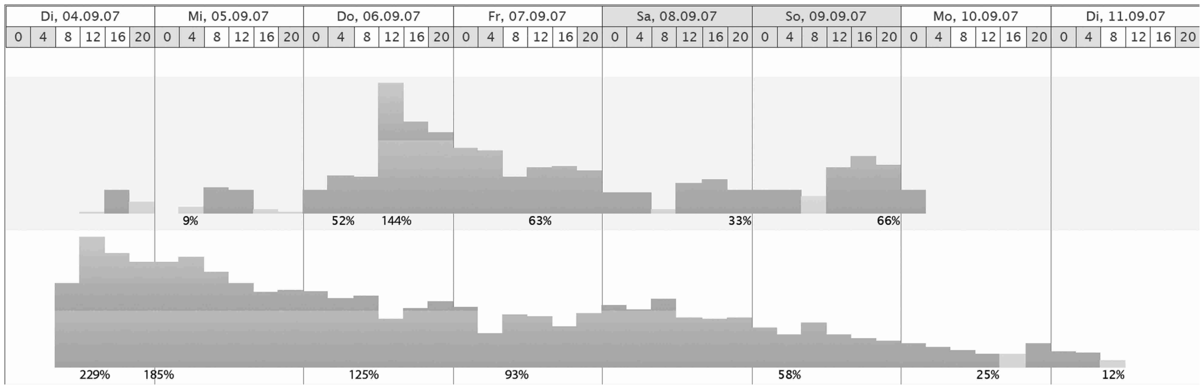

Figure 4 **Visualization of the Unleveled Resource Consumption for Two Selected Resource Types in a Real-World Project, Aggregated on a Four-Hour Basis**

Note. The resource types are cleaners (on top, 100% = 3 units) and metal workers (bottom, 100% = 22 units), respectively.

per job. We used 10 randomly generated instances per set. The work volume of any job lies between 6 and 20. This is clearly a deviation from the real-world data we received, but this simplification enables us to compute optimal solutions. Additionally, we generated instances with six or seven resource units per job. To compare the results with the previous test sets, we scaled the work volume for each job with a factor of 6.5. Each job requires one out of two resource types that must be leveled and have an upper bound on the resource capacity of 30 or 40. The resource costs are chosen as in the real-world instances. The precedence relations are chosen randomly in such a way that the precedence graph is again series parallel as in the real-world instances.

For each of these randomly generated instances, we determine deadlines (that is, turnaround makespans) that a manager may choose in the same way as for the real-world instances. We compute for each instance the minimum and maximum project duration and a value in between, which gives three instances for the resource-leveling heuristic. The total computation time of our algorithm is less than a second for each instance. We assess the quality of our solutions by comparing against CPLEX solutions for the corresponding MIP formulation stated in §3. As mentioned earlier, the model uses time-indexed variables and is therefore not applicable to instances of the size as is typical in practice. To solve our smaller-size test instances, we bound the computation time for CPLEX by one hour. If CPLEX does not find a provably optimal solution, then we use the current lower bound on the optimum value for comparison with the results of our heuristic. Tables 1 and 2 summarize the computational results for each of the two classes of instances, with resource requirements of one to two and six to seven units per job.

As already mentioned, our heuristic solves all test instances in less than one second. In contrast, CPLEX needs for instances with 60 jobs on average nearly the total given time limit of one hour. The fourth

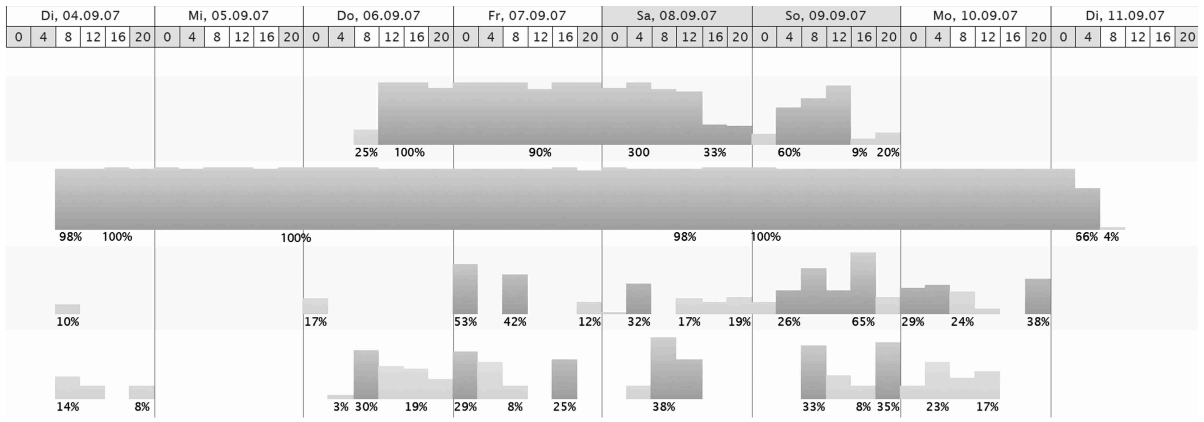

Figure 5 **Visualization of the Leveling for Four Selected Resource Types in a Real-World Project, Aggregated on a Four-Hour Basis**

Notes. The top two resource types are the same as in Figure 4, i.e., cleaners (3 units) and metal workers (22 units), respectively. The bottom two resource types correspond to two different cranes (14 tons and 22 tons) that are only used sporadically. Such resource types are excluded in the leveling.

Table 1 Comparison of Runtime and Resource Availability Cost (or Lower Bounds) of Optimal Schedules Obtained by CPLEX and Schedules Obtained by Our Heuristic (Between One and Two Resource Units per Job)

Number of jobs	Runtime heuristic	Average runtime CPLEX	Optimal solutions CPLEX (%)	Optimal solutions heuristic (%)	Gap heuristic vs. opt. (%)	Max. gap heuristic vs. opt. (%)	Gap heuristic vs. LB (%)	Max. cost gap heuristic vs. LB (%)
30	<1 sec	20 sec	100	61	8	33	—	—
40	<1 sec	20 sec	100	47	10	38	—	—
50	<1 sec	25 sec	100	59	7	29	—	—
60	<1 sec	12 min	83	40	9	33	10	33

Table 2 Comparison of Runtime and Resource Availability Cost (or Lower Bounds) of Optimal Schedules Obtained by CPLEX and Schedules Obtained by Our Heuristic (Between Six and Seven Resource Units per Job)

Number of jobs	Runtime heuristic	Average runtime CPLEX	Optimal solutions CPLEX (%)	Optimal solutions heuristic (%)	Gap heuristic vs. opt. (%)	Max. gap heuristic vs. opt. (%)	Gap heuristic vs. LB (%)	Max. cost gap heuristic vs. LB (%)
30	<1 sec	20 min	68	25	12	48	15	48
40	<1 sec	36 min	43	0	13	24	14	30
50	<1 sec	36 min	41	7	16	33	17	38
60	<1 sec	54 min	10	3	10	19	18	36

Table 3 Comparison of Runtime and Resource Availability Cost (or Lower Bounds) of Optimal Schedules Obtained by CPLEX and Schedules Obtained by Our Heuristic (Between Two and Four Resource Units per Job)

Number of jobs	Runtime heuristic	Average runtime CPLEX	Optimal solutions CPLEX (%)	Optimal solutions heuristic (%)	Gap heuristic vs. opt. (%)	Max. gap heuristic vs. opt. (%)	Gap heuristic vs. LB (%)	Max. cost gap heuristic vs. LB (%)
30	<1 sec	29 sec	100	43	9	31	—	—
40	<1 sec	19 min	73	10	11	57	14	69
50	<1 sec	45 min	15	0	11	17	15	25

column in Tables 1 and 2 reveals that with the increasing the number of jobs, CPLEX computations reach the given time limit, and thus, the computation process is aborted without having found an optimal solution. This situation occurs, in particular, if we set the resource allocation to six or seven resource units. In a few cases, our heuristic actually found an optimal solution after a few seconds, whereas CPLEX did not within one hour. The fifth column in Tables 1 and 2 quantifies how often our heuristic yields provable optimal solutions. For resource allocations between one and two units, we solve a significant number of instances to optimality. This changes when the resource requirements increase to six and seven units. In those cases, in which a provably optimal solution is found (by CPLEX), we compare its cost with those of our heuristic. The sixth column in the tables shows that we obtain solutions that are close to optimum. We leave, on average, a gap of about 7% to 10% and 38% in the worst case; see Table 1. Increasing the number of resource units yields somewhat worse results with an average gap of up to 16% and 48% in the worst case; see Table 2. The last columns in Tables 1 and 2 compare the results of our heuristic to the lower bounds obtained by CPLEX. If CPLEX has solved all instances to optimality, we omit the last entries because they are equal to the sixth and seventh columns. We compute the cost gap of our heuristic to the lower bound over all instances and to those where CPLEX found an optimal solution. In total, this does not dramatically increase the average gap, which shows that our solutions leave a gap of same size to the lower bounds as to the suboptimal solutions by CPLEX.

So far we have evaluated our heuristic on instances with up to two processing alternatives per job. Table 3 shows our computational results on instances where each job requires two, three, or four resource units that are given in advance per job. The work volume per job lies between 20 and 50. The other parameters are equal to those of the instances before. With increasing numbers of jobs as well as increasing project durations, the number of optimal solutions found by CPLEX decreases and thus the number of proven optimal solutions found by our heuristic solutions also decreases. It turns out that the average optimality gap is again about 10%. The maximum gap has been 57% for a single instance.

6. Conclusions and Research Perspectives

To the best of our knowledge, popular project management software does not support a time-cost trade-off related analysis. Resource-leveling packages do exist but seem to use very simple heuristics. Moreover, they have their limitations in the presence of working shifts, capacity bounds, or other specialized constraints such as conflicting job sets.

Motivated by applications in chemical manufacturing, we have formulated the shutdown and turnaround scheduling problem as an integrated problem that contains various optimization problems as subproblems, such as the time-cost trade-off problem, the problem of scheduling with resource capacities and working shifts, and the resource leveling problem, all of which have been considered individually previously. We reported on our successful solution approach within a more comprehensive decision support tool that additionally provides tools for risk analysis during the decision process and for the final schedule. Our optimization algorithm yields near-optimal solutions in a very short time. We hope that our work initiates more research on this general and integrated model to overcome the deficiencies of current project management tools.

Another very challenging line of research has come out of extensive discussions with practitioners about how to cope with uncertainty in turnaround scheduling. One question addresses the risk inherent in the whole planning process. So far we have only implemented tools for risk evaluation of given schedules. We applied techniques to determine an upper bound on the expected tardiness and the probability of meeting the makespan for a given project schedule. This allows the project manager to choose a schedule according to his or her risk affinity. Nevertheless, this method is only applied after the schedule optimization. We expect that an integrated approach that combines risk analysis and scheduling might yield better decision support for turnaround projects, but this is currently beyond the optimization methods available in practice.

Acknowledgments
The authors thank T.A. Cook Consultants for providing the data used to test their solution method and for explaining the details of shutdown and turnaround scheduling projects. The authors also thank Eamonn T. Coughlan, Elisabeth Günther, Ralf Hoffmann, Robert Pankrath, and Ines Spenke for their support in implementing the algorithms and developing their software tool.

References

Adel, S. M., S. E. Elmaghraby. 1984. Optimal linear approximation in project compression. *IIE Trans.* **16**(4) 339–347.

Adlakha, V. G., V. G. Kulkarni. 1989. A classified bibliography of research on stochastic PERT networks: 1966–1987. *Inform. Systems Oper. Res.* **27**(3) 272–296.

Bandelloni, M., M. Tucci, R. Rinaldi. 1994. Optimal resource leveling using non-serial dynamic programming. *J. Soc. Indust. Appl. Math.* **78**(2) 162–177.

Bourges, A., J. Killebrew. 1962. Variation in activity level in a cyclical arrow diagram. *J. Indust. Engrg.* **13**(2) 76–83.

Brucker, P., A. Drexl, R. Möhring, K. Neumann, E. Pesch. 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *Eur. J. Oper. Res.* **112**(1) 3–41.

Cieliebak, M., T. Erlebach, F. Hennecke, B. Weber, P. Widmayer. 2004. Scheduling with release times and deadlines on a minimum number of machines. J.-J. Lévy, E. W. Mayr, J. C. Mitchell, eds. *Exploring New Frontiers of Theoretical Informatics: IFIP 18th World Comput. Congress. Proc. Third IFIP Internat. Conf. Theoret. Comput. Sci.* (*TCS 2004*), Kluwer Academic Publishers, Norwell, MA, 209–222.

De, P., E. J. Dunne, J. B. Ghosh, C. E. Wells. 1995. The discrete time-cost tradeoff problem revisited. *Eur. J. Oper. Res.* **81**(2) 225–238.

De, P., E. J. Dunne, J. B. Ghosh, C. E. Wells. 1997. Complexity of the discrete time-cost tradeoff problem for project networks. *Oper. Res.* **45**(2) 302–306.

Deineko, V. G., G. J. Woeginger. 2001. Hardness of approximation of the discrete time-cost tradeoff problem. *Oper. Res. Lett.* **29**(5) 207–210.

Demeulemeester, E. L., W. S. Herroelen. 2002. *Project Scheduling: A Research Handbook*. Kluwer Academic Publishers, Norwell, MA.

Du, J., J. Y.-T. Leung. 1989. Complexity of scheduling parallel task systems. *SIAM J. Discrete Math.* **2**(4) 473–487.

Easa, S. M. 1989. Resource leveling in construction by optimization. *J. Construction Engrg. Management* **115**(2) 302–316.

Elmaghraby, S. E., J. Kamburowski. 1992. The analysis of activity networks under generalized precedence relations (GPRs). *Management Sci.* **38**(9) 1245–1263.

Falk, J. E., J. L. Horowitz. 1972. Critical path problems with concave cost-time curves. *Management Sci.* **19**(4, Part 1) 446–455.

Franck, B., K. Neumann, C. Schwindt. 2001. Project scheduling with calendars. *OR Spectrum* **23**(3) 325–334.

Fulkerson, D. R. 1961. A network flow computation for project cost curves. *Management Sci.* **7**(2) 167–178.

Grigoriev, A., M. Sviridenko, M. Uetz. 2007. Machine scheduling with resource dependent processing times. *Math. Programming* **110**(1) 209–228.

Hagstrom, J. N. 1988. Computational complexity of PERT problems. *Networks* **18**(2) 139–147.

Harris, R. B. 1990. Packing method for resource leveling (PACK). *J. Construction Engrg. Management* **116**(2) 331–350.

Jansen, K., H. Zhang. 2006. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Trans. Algorithms* **2**(3) 416–434.

Kapur, K. C. 1973. An algorithm for the project cost-duration analysis problem with quadratic and convex cost functions. *IIE Trans.* **5**(4) 314–332.

Kelley, J. E., Jr. 1961. Critical-path planning and scheduling: Mathematical basis. *Oper. Res.* **9**(3) 296–320.

Lamberson, L. R., R. R. Hocking. 1970. Optimum time compression in project scheduling. *Management Sci.* **16**(10) B597–B606.

Lepère, R., D. Trystram, G. J. Woeginger. 2002. Approximation algorithms for scheduling malleable tasks under precedence constraints. *Internat. J. Found. Comput. Sci.* **13**(4) 613–627.

Ludwig, A., R. H. Möhring, F. Stork. 2001. A computational study on bounding the makespan distribution in stochastic project networks. *Ann. Oper. Res.* **102**(1–4) 49–64.

Meilijson, I., A. Nádas. 1979. Convex majorization with an application to the length of critical paths. *J. Appl. Probab.* **16**(3) 671–677.

Möhring, R. H. 2001. Scheduling under uncertainty: Bounding the makespan distribution. H. Alt, ed. *Comput. Discrete Math. Lecture Notes in Computer Science*, Vol. 2122. Springer, Berlin, 79–97.

Neumann, K., J. Zimmermann. 2000. Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. *Eur. J. Oper. Res.* **127**(2) 425–443.

Neumann, K., C. Schwindt, J. Zimmermann. 2003. *Project Scheduling with Time Windows and Scarce Resources*, 2nd ed. Springer, Berlin.

Phillips, D. T., A. Garcia-Diaz. 1981. *Fundamentals of Network Analysis*. Prentice-Hall, Englewood Cliffs, NJ.

Phillips, S., Jr., M. I. Dessouky. 1977. Solving the project time/cost tradeoff problem using the minimal cut concept. *Management Sci.* **24**(4) 393–400.

Pritsker, A. A. B., L. J. Watters, P. M. Wolfe. 1969. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Sci.* **16**(1) 93–108.

Schmidt, G. 2000. Scheduling with limited machine availability. *Eur. J. Oper. Res.* **121**(1) 1–15.

Shabtay, D., G. Steiner. 2007. A survey of scheduling with controllable processing times. *Discrete Appl. Math.* **155**(13) 1643–1666.

Siemens, N., C. Gooding. 1975. Reducing project duration at minimum cost: A time-cost tradeoff algorithm. *OMEGA* **3**(5) 569–581.

Skutella, M. 1998a. Approximation algorithms for the discrete time-cost tradeoff problem. *Math. Oper. Res.* **23**(4) 909–929.

Skutella, M. 1998b. Approximation and randomization in scheduling. Ph.D. thesis, Technische Universität Berlin, Berlin.

Vanhoucke, M. 2005. New computational results for the discrete time/cost trade-off problem with time-switch constraints. *Eur. J. Oper. Res.* **165**(2) 359–374.

Vanhoucke, M., D. Debels. 2007. The discrete time/cost trade-off problem: Extensions and heuristic procedures. *J. Sched.* **10**(4–5) 311–326.

Vanhoucke, M., E. Demeulemeester, W. Herroelen. 2002. Discrete time/cost trade-offs in project scheduling with time-switch constraints. *J. Oper. Res. Soc.* **53**(7) 1–11.

Weiss, G. 1986. Stochastic bounds on distributions of optimal value functions with applications to PERT, network flows and reliability. *Oper. Res.* **34**(4) 595–605.

Yang, H.-H., Y.-L. Chen. 2000. Finding the critical path in an activity network with time-switch constraints. *Eur. J. Oper. Res.* **120**(3) 603–613.

Zhan, J. 1992. Calendarization of time planning in MPM networks. *ZOR—Methods Models Oper. Res.* **36**(5) 423–438.