OR Spektrum (2001) 23: 297–324

OR Spektrum

Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling*

Birger Franck, Klaus Neumann, and Christoph Schwindt

Institut für Wirtschaftstheorie und Operations Research, University of Karlsruhe, 76128 Karlsruhe, Germany (e-mail: {franck,neumann,schwindt}@wior.uni-karlsruhe.de)

Received: July 26, 2000 / Accepted: May 15, 2001

Abstract. We present heuristic procedures for approximately solving large project scheduling problems with general temporal and resource constraints. In particular, we propose several truncated branch-and-bound techniques, priority-rule methods, and schedule-improvement procedures of types tabu search and genetic algorithm. A detailed experimental performance analysis compares the different heuristics devised and shows that large problem instances with up to 1000 activities and several resources can efficiently be solved with sufficient accuracy.

Key words: Resource-constrained project scheduling – Truncated branch-and-bound – Priority-rule methods – Tabu search – Genetic algorithms

1 Introduction

We deal with the problem of minimizing the duration of a project subject to general temporal and resource constraints. The temporal constraints are given by prescribed minimum and maximum time lags between the activities of the project. The resource constraints refer to limited renewable resources such as manpower and machines, which are necessary for carrying out the project activities. Aside from "classical" applications such as scheduling projects in construction industry, software development, etc., the above problem arises in single-item and make-to-order-production (cf. Franck et al., 1997; Neumann and Schwindt, 1997), operations management in process industries (cf. Schwindt and Trautmann, 2000), etc.

* The authors would like to acknowledge the helpful comments of two anonymous referees. This research has been supported in part by the Deutsche Forschungsgemeinschaft (Grant Ne 137/4).

Correspondence to: C. Schwindt

Branch-and-bound algorithms for our problem have been proposed by Bartusch et al. (1988), De Reyck and Herroelen (1998), Schwindt (1998b), Fest et al. (1999), and Dorndorf et al. (2000). Whereas the approach by Dorndorf et al. relies on a binary, time-oriented branching scheme using several constraint propagation techniques, the other procedures are based on the relaxation of resource scarcity. The resource constraints are observed by successively introducing precedence constraints between activities (Bartusch et al. and De Reyck and Herroelen), release dates of activities (Fest et al.), or disjunctive precedence constraints between sets of activities (Schwindt). The latter approach is discussed in more detail in Section 3.

Truncated branch-and-bound methods have been proposed by Schwindt (1998b) for the first time. Priority-rule methods have been devised by Zhan (1994), Neumann and Zhan (1995), Brinkmann and Neumann (1996), Franck and Neumann (1998), and Franck (1999, Ch. 4). An overview of some of those techniques can be found in Brucker et al. (1999) and Neumann and Zimmermann (1999).

This paper contains advanced versions of those heuristic algorithms (some of which are not available to the English speaking public), with several new results included. Also, some additional solution methods, for example, decomposition approaches and schedule-improvement procedures are devised. Moreover, we present a detailed experimental performance analysis based on the project generator ProGen/max (Schwindt 1998a), which compares the heuristics discussed. It turns out that the best heuristics can solve large problem instances with up to 1000 project activities and several resources with sufficient accuracy in less than one minute.

2 Basic concepts

2.1 Temporal project scheduling

First we review some basic concepts and results from temporal project scheduling. Consider a project which consists of n real activities $1, \ldots, n$. Each activity is supposed to be carried out without interruption. We introduce the fictitious activities 0 and $n+1$, which represent the beginning and completion of the project, respectively. Thus, $V = \{0, 1, \ldots, n+1\}$ is the set of project activities. Let $p_i \in \mathbb{Z}_{\geq 0}$ be the *processing time* or *duration* and $S_i \geq 0$ be the *start time* of activity $i \in V$, where $S_0 := 0$. Then S_{n+1} equals the *project duration*.

We assign node i of a network to project activity i and identify activity i with node i $(i \in V)$. If there is a prescribed *minimum time lag* $d_{ij}^{min} \in \mathbb{Z}_{\geq 0}$ between the start of two activities i and j, $i \neq j$, i.e., $S_j - S_i \geq d_{ij}^{min}$, we introduce an arc $\langle i, j \rangle$ with weight $\delta_{ij} := d_{ij}^{min}$. If there is a *maximum time lag* $d_{ij}^{max} \in \mathbb{Z}_{\geq 0}$ between the start of i and j, i.e., $S_j - S_i \leq d_{ij}^{max}$, we introduce an arc $\langle j, i \rangle$ with weight $\delta_{ji} := -d_{ij}^{max}$. Let E be the arc set of the resulting *activity-on-node network* or *project network* N. The *temporal constraints* can then be stated as

$$S_j - S_i \geq \delta_{ij} \quad (\langle i, j \rangle \in E) \tag{1}$$

Network N generally contains cycles due to maximum time lags. By appropriate specification of the minimum and maximum time lags, N is determined uniquely and does not contain parallel arcs (Neumann and Schwindt, 1997).

$S = (S_i)_{i \in V}$ is called a *schedule*. Note that $S_i \geq 0$ $(i \in V)$ and $S_0 = 0$. A schedule S is called *time-feasible* if it satisfies (1). The set of time-feasible schedules is denoted by $\mathcal{S}_T$. It is well-known that $\mathcal{S}_T \neq \emptyset$ exactly if N does not contain any cycle of positive length (Bartusch et al., 1988). In what follows we assume that $\mathcal{S}_T \neq \emptyset$.

Let d_{ij} be the length of a longest path from node i to node j in N. If there is no such path, we set $d_{ij} := -\infty$. In case that $d_{ij} > 0$, $d_{ij}^{min} := d_{ij}$ represents a minimum time lag between the start of activities i and j, and if $d_{ij} < 0$, then $d_{ji}^{max} := -d_{ij}$ is a maximum time lag between the start of j and i. If $d_{ij} = 0$, the time lag may be either minimum or maximum. $ES_i := d_{0i}$ is the *earliest start time* and $LS_i := UB - d_{i,n+1}$ is the *latest start time* of activity $i \in V$, where UB is an upper bound on the shortest project duration, which is often prescribed in practice. $ES = (ES_i)_{i \in V}$ and $LS = (LS_i)_{i \in V}$ are called the *earliest* and *latest schedule*, respectively. By *temporal project scheduling* we mean the computation of d_{ij} $(i, j \in V)$ and ES_i and LS_i $(i \in V)$.

Some heuristics discussed later on require a *strict order* $\prec$ in the node set V of N. For $i, j \in V, i \neq j$, we define $i \prec j$ exactly if either (i) $d_{ij} > 0$ or (ii) $d_{ij} = 0$ and $d_{ji} < 0$. The set of immediate predecessors of node i with respect to $\prec$ is denoted by $Pred^{\prec}(i)$.

2.2 Resource-constrained project scheduling

Assume that a set $\mathcal{R}$ of renewable resources are required for carrying out the project activities. Let $R_k > 0$ be the capacity of resource k available and r_{ik} be the amount of resource k used by activity i, where $0 \leq r_{ik} \leq R_k$ $(i \in V,\ k \in \mathcal{R})$. Given a schedule S, $\mathcal{A}(S,t) := \{i \in V | S_i \leq t < S_i + p_i\}$ is the set of activities in progress at time t, also called the *active set*, and $r_k(S,t) := \sum_{i \in \mathcal{A}(S,t)} r_{ik}$ is the amount of resource k used at time t. The *resource constraints* are then as follows:

$$r_k(S,t) \leq R_k \quad (k \in \mathcal{R}; t \geq 0) \tag{2}$$

A schedule S is called *resource-feasible* if it satisfies (2). A schedule S which is both time- and resource-feasible is called *feasible*. The sets of resource-feasible and feasible schedules are denoted by $\mathcal{S}_R$ and $\mathcal{S}$, respectively. $\mathcal{S} = \mathcal{S}_T \cap \mathcal{S}_R$ represents the union of finitely many convex polyhedra and is generally disconnected. The decision problem whether or not $\mathcal{S} \neq \emptyset$ is NP-complete (Bartusch et al., 1988). The strongly NP-hard problem of minimizing the project duration subject to the temporal and resource constraints can be written as

$$\left.\begin{array}{ll} \text{Min.} & S_{n+1} \\ \text{s.t.} & S \in \mathcal{S} \end{array}\right\} \tag{3}$$

Problem (3) is denoted by $PS|temp|C_{max}$ in the three-field notation of project scheduling problems proposed by Brucker et al. (1999), where PS stands for resource-constrained project scheduling, $temp$ for general temporal constraints, and C_{max} for the makespan to be minimized. A different three-field notation has been proposed by Herroelen et al. (1999), where problem (3) is denoted by

$m, 1|gpr|C_{max}$. A feasible schedule S with minimum S_{n+1} is called *optimal*. An *upper bound* UB on the shortest project duration for problem (3) is $\overline{d} := \sum_{i \in V} \max(p_i, \max_{\langle i,j \rangle \in E} \delta_{ij})$.

A *cycle structure* C of project network N is a strong component that contains at least two nodes. For each cycle structure C of N viewed as a separate subproject and started at time zero, a problem of type (3) can be specified. Likewise, the concepts of *feasible* and *optimal subschedules* for the subproject corresponding to C can be introduced. We then state the following *decomposition theorem* (cf. Neumann and Zhan, 1995), which is important for decomposition methods discussed later on.

Theorem 1 *There is a feasible schedule for project network N if and only if for each cycle structure C of N, there is a feasible subschedule for C.*

2.3 Preprocessing

All heuristic methods proposed later on contain a *preprocessing phase*, which consists of introducing additional temporal constraints to resolve so-called resource conflicts caused by two-element forbidden sets. Given a schedule S, $F \subseteq \mathcal{A}(S,t)$ is called a *forbidden set* if $\sum_{i \in F} r_{ik} > R_k$ for some $k \in \mathcal{R}$ (*resource conflict*). The following *preprocessing theorem* (Brucker et al., 1998; De Reyck and Herroelen, 1998) shows how to resolve resource conflicts for forbidden sets $F = \{i, j\}$. The theorem is a straight extension of a 2-bound consistency test known from the field of disjunctive constraint propagation (see e.g. Carlier and Pinson, 1989, or Nuijten, 1994).

Theorem 2 *If for two activities i and j, $i \neq j$, and some $k \in \mathcal{R}$, it holds that $r_{ik} + r_{jk} > R_k$ and (a) $LS_i < ES_j + p_j$ or (b) $d_{ij} > -p_j$, then we have $S_j \geq S_i + p_i$ for each $S \in \mathcal{S}$ with $S_{n+1} \leq UB$.*

UB is again any upper bound on the shortest project duration. An additional constraint $S_j \geq S_i + p_i$ corresponds to introducing a new arc $\langle i, j \rangle$ with weight p_i or, if there is already an arc $\langle i, j \rangle$ with weight δ_{ij}, a new arc weight $\max(\delta_{ij}, p_i)$ in project network N. The first part of preprocessing consists of introducing such new arcs or arc weights for all forbidden sets $\{i, j\}$ for which (a) or (b) in Theorem 2 hold. This results in a modified project network N' for which temporal project scheduling is performed (second part of preprocessing). Notice that for N', the values of d_{ij} and ES_i are generally larger and the values of LS_i are generally smaller than for N. It may happen that $d_{ij} + d_{ji} > 0$ for some $i, j \in V$, i.e., N' contains a cycle of positive length and hence there is no $S \in \mathcal{S}$ with $S_{n+1} \leq UB$. In the latter case we say that *preprocessing rejects* UB. Inequalities (a) or (b) may again hold in N' for some forbidden sets $\{i, j\}$. Thus, preprocessing generally consists of several preprocessing iterations until either no more temporal constraints $S_j \geq S_i + p_i$ or, equivalently, no new arcs or arc weights have to be introduced or preprocessing rejects UB.

2.4 Lower bounds

Lower bounds LB on the shortest project duration for problem (3) are needed for (truncated) branch-and-bound methods and experimental performance analysis (see Sections 3 and 6). For example, the shortest project duration $LB0 := ES_{n+1}$ of the *resource relaxation* of (3), i.e., we omit the resource constraints (2), represents such a lower bound. If we split the activities into subactivities of duration one and resource requirements one and delete the temporal constraints (1), we obtain the lower bound $LBR := \max_{k\in\mathcal{R}} \sum_{i\in V} r_{ik}p_i/R_k$. Lower bounds which result from relaxations of (3) are called *constructive lower bounds*. We speak of a *destructive lower bound* LBD if it can be shown that there is no $S \in \mathcal{S}$ with $S_{n+1} < LBD$. The computation of destructive lower bounds can be viewed as applying a so-called shaving technique to dummy activity $n+1$ (cf. Martin and Shmoys, 1996).

Destructive lower bounds LBD and corresponding upper bounds UB can be found as follows (Heilmann and Schwindt, 1997; Klein and Scholl, 1999). Since all temporal data δ_{ij} $(\langle i,j\rangle \in E)$ and p_i $(i \in V)$ are integers, there is always an optimal schedule which is integer-valued provided that $\mathcal{S} \neq \emptyset$. The algorithm begins with a lower bound LB_{start} and an upper bound UB_{start}, for example, $LB_{start} := \max(LB0, \lceil LBR\rceil)$ and $UB_{start} := \overline{d}$. In each iteration, we take a new upper bound $d := \lceil (LB + UB)/2\rceil$ on trial and perform preprocessing. If preprocessing rejects d, then $LB := d + 1$, otherwise we set $LB := \max(LB, ES_{n+1})$, where ES_{n+1} is the shortest project duration from last preprocessing iteration, and $UB := d - 1$. After that, we again put $d := \lceil (LB + UB)/2\rceil$, perform preprocessing and proceed analogously. We terminate as soon as $LB > UB$. This algorithm, which determines the least upper bound UB within set $\{LB_{start}, LB_{start}+1, \ldots, UB_{start}\}$ that is not rejected by preprocessing and a corresponding lower bound LBD using binary search, is formulated in Algorithm 1.

Algorithm 1 *Destructive lower bound*

$LB := LB_{start}$, $UB := UB_{start}$, $d := \lceil (LB + UB)/2\rceil$
While $LB \leq UB$ **do**
  Perform preprocessing
  If *preprocessing rejects* d **then**
    If $d = UB_{start}$ **then** *terminate (∗ there is no feasible schedule ∗)*
    else $LB := d + 1$, $d := \lceil (LB + UB)/2\rceil$
    end $(*$ *if* $*)$
  else
    $LB := \max(LB, ES_{n+1})$ $(*\, ES_{n+1}$ *from preprocessing* $*)$,
    $UB := d - 1$, $d := \lceil (LB + UB)/2\rceil$
  end $(*$ *if* $*)$
end $(*$ *while* $*)$
$LBD := LB$

The time complexity of Algorithm 1 is $O(\log \overline{d}\max(|\mathcal{R}||V|^2, |V|^4))$.

Finally, we present a lower bound which is often tighter than the preceding ones for large projects (Schwindt, 1998b). This lower bound is closely related to the concept of energetic reasoning or interval consistency (cf. e.g. Nuijten, 1994; Baptiste et al., 1999; Dorndorf et al., 1999). Given a schedule S^+, e.g. $S^+ = ES$,

$$w_k(S^+, t') := \sum_{\substack{h \in V \\ S_h^+ + p_h > t'}} r_{hk} \min(p_h, S_h^+ + p_h - t') \tag{4}$$

is the *workload* to be processed by resource $k \in \mathcal{R}$ in time interval $[t', \infty[$. $\lceil w_k(S^+, t')/R_k \rceil$ is a lower bound on the time required for processing workload $w_k(S^+, t')$ by resource k. As a consequence,

$$LBW(S^+) := \max_{k \in \mathcal{R}} \max_{i \in V} (S_i^+ + \lceil w_k(S^+, S_i^+)/R_k \rceil) \tag{5}$$

is a lower bound on the shortest project duration for all $S \in \mathcal{S}$ with $S \geq S^+$. $LBW(S^+)$ can be determined in $O(|\mathcal{R}||V| \log |V|)$ time.

3 Truncated branch-and-bound procedures

In this section we propose three different truncations of a branch-and-bound algorithm for $PS|temp|C_{max}$: an ε-approximate method, a filtered beam search procedure, and a decomposition method based on Theorem 1. First we touch upon the underlying branch-and-bound algorithm (for details we refer to Schwindt 1998b,c).

3.1 Enumeration

Let S^+ be the current schedule and UB be the current upper bound on the minimum project duration. Starting in the root node u of the enumeration tree with search space $\mathcal{S}(u) = \mathcal{S}_T$, we set $S^+ := ES$ and $UB := \overline{d} + 1$. We then try to restrict the search space by performing the preprocessing phase. If preprocessing rejects UB as valid upper bound, we have shown that $\mathcal{S} = \emptyset$, and the algorithm terminates. Otherwise, we proceed by testing whether S^+ is resource-feasible or not. If $S^+ \in \mathcal{S}_R$, S^+ is feasible, and we set $UB := S_{n+1}^+$ and perform backtracking. Otherwise, we consider the earliest point in time t at which a resource conflict occurs, i.e., $r_k(S^+, t) > R_k$ for some $k \in \mathcal{R}$. Next, we compute all partitions $\{A, B\}$ of the forbidden active set $\mathcal{A}(S^+, t)$ such that A is an inclusion-maximal set which is not forbidden. Thus, B represents an inclusion-minimal set of activities whose delay resolves the resource conflict at time t. A is referred to as a *maximal feasible set*, whereas B is termed a *minimal delaying alternative* for $\mathcal{A}(S^+, t)$. The overlapping of activities $j \in B$ with all activities $i \in A$ is prevented by a *disjunctive precedence constraint* $A \to B$ between sets A and B saying that

$$\min_{j \in B} S_j \geq \min_{i \in A} (S_i + p_i) \tag{6}$$

which ensures that activities $j \in B$ cannot be started before the first completion of an activity $i \in A$. (6) is sufficient to resolve the resource conflict caused by the simultaneous execution of the activities from $\mathcal{A}(S^+, t)$. For each partition $\{A, B\}$ of $\mathcal{A}(S^+, t)$ we introduce an enumeration node v with search space $\mathcal{S}(v) := \mathcal{S}(u) \cap \{S \in \mathcal{S}_T \mid \min_{j \in B} S_j \geq \min_{i \in A}(S_i + p_i)\}$. Then, for each partition, we either determine an optimal solution S^v to the corresponding scheduling problem

$$\left.\begin{array}{ll} \text{Min.} & S_{n+1} \\ \text{s.t.} & S \in \mathcal{S}(v) \end{array}\right\} \tag{7}$$

or establish $\mathcal{S}(v) = \emptyset$ using a pseudo-polynomial fixed-point algorithm. If $\mathcal{S}(v) = \emptyset$ or $S^v_{n+1} \geq UB$, node v is removed from the enumeration tree. If all child nodes v of u have been deleted, we perform backtracking. Otherwise, we branch from one of the remaining nodes v which minimizes a convex combination

$$\lambda S^v_{n+1} + (1 - \lambda) \sum_{k \in \mathcal{R}} \frac{1}{R_k} \sum_{i \in V} (S^v_i - S^+_i) p_i r_{ik} \quad (0 < \lambda < 1) \tag{8}$$

of the project duration S^v_{n+1} and the shifted relative workload and set $u := v$, $S^+ := S^u$. Finally, we compute the corresponding lower bound $LB(u) := LBW(S^+)$ and fathom node u if $LB(u) \geq UB$. Proceeding with the check of resource-feasibility for S^+, these steps are repeated until the whole feasible region has been explored.

The algorithm for solving problem (7) relies on the existence of a unique minimal point S^v of $\mathcal{S}(v) \neq \emptyset$. Starting with the current schedule $S^v := S^+$, we check whether or not the disjunctive precedence constraints defined along the path from the root to node v are met. Assume that schedule S^v does not satisfy one of these constraints, say $A' \to B'$. Then for all $j \in B'$, each schedule $S \in \mathcal{S}(v)$ satisfies the inequalities $S_j \geq \min_{i \in A'}(S^v_i + p_i)$, and S^v_j can be increased up to $\min_{i \in A'}(S^v_i + p_i)$. Subsequently, we restore the time-feasibility of S^v by setting $S^v_h := \max[S^v_h, \max_{j \in B'}(S^v_j + d_{jh})]$ for all $h \in V$. The iterations are performed until either all disjunctive precedence constraints are satisfied or $S^v_{n+1} \geq UB$. In the latter case, either $\mathcal{S}(v) = \emptyset$ or $\min_{S \in \mathcal{S}(v)} S_{n+1} \geq UB$. Obviously, the number of iterations is bounded by $|V|\,UB \leq |V|\overline{d}$.

Notice that despite the pseudo-polynomial time complexity, the running time of this algorithm is independent of the scaling of the time lags. Moreover, the algorithm also solves (7) for arbitrary regular (i.e. componentwise nondecreasing) objective functions $f(S)$ instead of S_{n+1}.

Algorithm 2 summarizes the enumeration procedure.

Algorithm 2 *Branch-and-bound algorithm*

Step 1. Initialize current enumeration node u by the root and current schedule S^+ by ES.
Set search space $\mathcal{S}(u) := \mathcal{S}_T$ and upper bound $UB := \overline{d} + 1$. Perform preprocessing. If UB is rejected as valid upper bound, then terminate (there is no feasible schedule).

Step 2. If S^+ is resource-feasible, set $UB := S^+_{n+1}$ and backtrack. Otherwise, determine earliest time t at which a resource conflict occurs.

Step 3. For each partition $\{A, B\}$ of $\mathcal{A}(S^+, t)$ into maximal feasible set A and minimal delaying alternative B, define a child node v with $\mathcal{S}(v) := \mathcal{S}(u) \cap \{S \in \mathcal{S}_T \mid \min_{j \in B} S_j \geq \min_{i \in A}(S_i + p_i)\}$ and solve corresponding problem (7).
If $\mathcal{S}(v) = \emptyset$ or resulting project duration $S^v_{n+1} \geq UB$, fathom node v.

Step 4. If all child nodes v have been eliminated, backtrack. Otherwise set u equal to a child node v minimizing (8) and $S^+ := S^u$.
Compute lower bound value $LB(u) := LBW(S^+)$. If $LB(u) \geq UB$, fathom node u and backtrack. Otherwise, go to Step 2.

Computational experience shows that when dealing with large problem instances with more than 100 real activities, two modifications improve the efficiency of Algorithm 2. Since the number of minimal delaying alternatives generally grows exponentially in the cardinality of the active set, we replace minimal delaying alternatives by singletons if $|\mathcal{A}(S^+, t)| > 10$. Moreover, the time needed for computing a first feasible schedule can be considerably reduced by replacing (8) with the following branching criterion. Let $\overline{N}$ be the network resulting from the addition of all arcs $\langle g, h \rangle$ to N with $g \in A'$ and $h \in B'$ for any of the disjunctive precedence constraints $A' \rightarrow B'$ giving rise to search space $\mathcal{S}(v)$. As long as no feasible schedule has been found, we branch from a node v which minimizes the number of activities $i \in A$ that are reachable from some $j \in B$ in $\overline{N}$. Ties are broken according to decreasing values of (8). After having determined a feasible schedule S^+, we perform preprocessing with $UB = S^+_{n+1}$ and restart the enumeration, where we now branch according to criterion (8).

3.2 Approximation method

The branch-and-bound algorithm described in Subsection 3.1 can easily be truncated to an ε-approximate heuristic with $\varepsilon > 0$. To this end, in Algorithm 2 we initialize the upper bound UB by $(1+\varepsilon)\,\overline{d}+1$. In Step 4 nodes u are only branched from if they satisfy the inequality

$$(1 + \varepsilon)\, LB(u) < UB \tag{9}$$

Since $UB = (1+\varepsilon)\,\overline{d}+1$ and $LB(u) \leq \overline{d}$ for all nodes u with nonempty search space $\mathcal{S}(u)$, no node u with $\mathcal{S}(u) \neq \emptyset$ is fathomed until a first feasible schedule S^+ has been determined. As before, we then set $UB := S^+_{n+1}$.

Let f^+ and f^* denote the objective function value of the best schedule determined by applying condition (9) to the evaluation of nodes u and the minimum objective function value, respectively. Consider a node u whose search space $\mathcal{S}(u)$ contains an optimal schedule. Then $LB(u) \leq f^*$. If u is fathomed, we obtain

$$f^+ \leq UB \leq (1 + \varepsilon)\, LB(u) \leq (1 + \varepsilon)\, f^*$$

where UB denotes the current upper bound when eliminating node u from the search tree. In that case, the relative deviation $(f^+ - f^*)/f^*$ does not exceed ε. If we branch from node u, there either exists a fathomed descendant node v such

that $\mathcal{S}(v)$ contains an optimal schedule or an optimal schedule is generated, i.e. $(f^+ - f^*)/f^* = 0 < \varepsilon$. Thus, the above truncation of the branch-and-bound algorithm represents an ε-approximate method. Obviously, the time complexity of this approach is not polynomial. Note that as a generalization of PRECEDENCE CONSTRAINED SCHEDULING, there is no polynomial-time approximation scheme for $PS|temp|C_{max}$ (cf. Garey and Johnson, 1979).

3.3 Filtered beam search

The computation of optimal or ε-approximate schedules for large problem instances with hundreds of activities may be too time-consuming. In what follows, we sketch the truncation of the branch-and-bound algorithm to a filtered beam search procedure (cf. e.g. Pinedo, 1995). By φ and $\beta < \varphi$ we denote the integers corresponding to the filter width and the beam width, respectively. After the generation of all child nodes v of current node u in Step 3 of Algorithm 2, we perform one iteration of the fixed-point algorithm for solving problems (7) belonging to nodes v, where only the respective disjunctive precedence constraint $A \rightarrow B$ introduced in node v is taken into account. Subsequently, nodes v are ordered according to nondecreasing values of the resulting project durations. For the first φ child nodes v of u, an optimal solution S^v to (7) is determined, whereas the remaining child nodes of u are excluded from further consideration. Finally, β nodes v with smallest project durations S^v_{n+1} are added to the enumeration tree.

For hard problem instances even a beam width of $\beta = 2$ is too large. Let

$$\widetilde{\beta} := \lceil \overline{\beta}\widetilde{u} \rceil$$

denote the integer random variable where $\widetilde{u}$ is uniformly distributed in interval $]0, 1]$ and $\overline{\beta} \geq 1$ is the maximum beam width chosen. For the beam width β we then choose realizations of $\widetilde{\beta}$. If $\overline{\beta} \leq 2$, the expected value of $\widetilde{\beta}$ equals $2 - 1/\overline{\beta}$. The filter width φ and maximum beam width $\overline{\beta}$ are chosen depending on the number n of activities and the desired maximum number σ of leaves of the enumeration tree. In our implementation, the filter width φ is set equal to $\lceil \ln n \rceil$. Computational experiments with the branch-and-bound algorithm have shown that the depth of the enumeration tree can generally be bounded by $n \ln n$. Thus, we choose

$$\overline{\beta} := 1/(2 - \sigma^{1/(n \ln n)})$$

For example, if $\sigma = 10000$, we have $\overline{\beta} = 1.0206$ for $n = 100$ and $\overline{\beta} = 1.0013$ for $n = 1000$.

3.4 Decomposition approach

This subsection is concerned with a decomposition heuristic which is tailored to an efficient computation of feasible solutions based on Theorem 1.

First we apply the approximation method to the problem instances induced by the cycle structures C of project network N. If there is a cycle structure for which

no feasible subschedule can be determined, the original instance is unsolvable. Otherwise, based on the feasible subschedules S^C found for cycle structures C with node sets V^C, the activities $i \in V^C$ are ordered according to nondecreasing start times S_i^C. Between any two consecutive activities $i, j \in V^C$, a minimum time lag $d_{ij}^{min} := S_j^C - S_i^C \geq 0$ is introduced. Moreover, we add a maximum time lag $d_{ij}^{max} := S_j^C - S_i^C \geq 0$ between the start of the first activity $i \in V^C$ and the last activity $j \in V^C$ such that all activities belonging to cycle structure C are firmly tied. The introduction of those additional time lags corresponds to adding a cycle C' of length zero for each cycle structure C to network N, which results in a network $\widetilde{N}$.

A first feasible schedule for the original instance can be computed without backtracking by a modification of the filtered beam search algorithm solving the instance which corresponds to network $\widetilde{N}$. To this end, we perform the enumeration according to the following traversing strategy. Let $\overline{N}$ be the network resulting from the addition of arcs to N from nodes $i \in A$ to nodes $j \in B$ for all disjunctive precedence constraints $A \to B$ which have been introduced along the path from the root to current node u. Due to the additional minimum time lags d_{ij}^{min}, the activities of a forbidden set F whose nodes belong to one and the same cycle structure cannot be executed any longer simultaneously. Thus, the resource conflicts can always be resolved such that for all corresponding disjunctive precedence constraints $A \to B$, there is no $i \in A$ reachable in $\overline{N}$ from any $j \in B$, which implies $\mathcal{S}(u) \neq \emptyset$. After having found a feasible schedule, the enumeration is continued as stated in Subsection 3.3.

In what follows, the algorithm just described is referred to as *bb decomposition method 1*. We obtain a second heuristic called *bb decomposition method 2* if we omit the introduction of the "backward arcs" $\langle j, i \rangle$ connecting the last and the first activity j and i, respectively, of each cycle structure.

Algorithm 3 *Bb decomposition method 1*

Step 1. *Determine all cycle structures C of N and compute feasible subschedules S^C by the approximation method. If for some C, a feasible schedule cannot be found, terminate.*

Step 2. *Add a cycle C' of length zero to each cycle structure C such that activities $i \in V^C$ are firmly tied according to start times S_i^C.*

Step 3. *Apply the filtered beam search procedure to the resulting extended network $\widetilde{N}$. Branch only from nodes which belong to disjunctive precedence constraints $A \to B$ where no activity $i \in A$ is reachable from an activity $j \in B$, until a first feasible schedule has been found.*

4 Priority-rule methods

Priority-rule methods schedule the project activities successively according to certain priority rules, but do not necessarily provide a feasible schedule if $\mathcal{S} \neq \emptyset$. In what follows, we discuss two different types of priority-rule techniques. The *sequential* or *direct method* processes the activities or respectively nodes of the

project network N one after another without evaluating the cycle structures separately. The *pr decompositon methods 1* and *2*, where pr stands for priority rule, are the analogues to the bb decomposition methods 1 and 2 from Subsection 3.4 and again exploit the decomposition theorem (Theorem 1). The direct method and pr decomposition methods 1 and 2 are discussed in detail in Subsection 4.1.

To construct a feasible schedule (either for the original project network N in the direct method or for a cycle structure or the modified network $\widetilde{N}$ in the pr decomposition methods), a *serial schedule generation scheme* is used (see Kolisch and Hartmann, 1999). The activities are scheduled consecutively, where the activity to be scheduled next is selected using some priority rule. In Subsection 4.2, we propose five alternative priority rules. The serial schedule generation scheme is studied in detail in Subsection 4.3.

4.1 Direct and decomposition methods

The *direct* or *sequential method* processes the activities of project network N one after another without evaluating the cycle structures of N separately. Algorithm 4 gives a formulation of the direct method. To determine the cycle structures in Step 1, any algorithm for finding the strong components of a directed graph can be used (Even, 1979; Franck and Neumann, 1998).

Algorithm 4 *Direct method*

Step 1. *Determine all cycle structures of N.*

Step 2. *Compute $\overline{d}$ and perform temporal project scheduling with $UB = \overline{d}$.*
If $d_{ij} + d_{ji} > 0$ for some $i, j \in V$, terminate (there is no feasible schedule).
Perform preprocessing with $UB = \overline{d}$. If preprocessing rejects $\overline{d}$, terminate (there is no feasible schedule).

Step 3. *Determine sets $Pred^{\prec}(i)$ for all $i \in V$.*
Find feasible schedule for N by serial schedule generation scheme, where all activities of one and the same cycle structure are scheduled directly one after another.

We now turn to the decomposition methods. For each cycle structure C of N, a feasible subschedule is determined and a corresponding cycle C' of length zero is added to C as shown in Subsection 3.4. If the "backward arc" is eliminated from C', we obtain pr decomposition method 2 instead of method 1. Pr decomposition method 1 is formulated in Algorithm 5.

Algorithm 5 *Pr decomposition method 1*

Steps 1 and 2 as in the direct method (Algorithm 4).

Step 3. *Perform Steps 2 and 3 of the direct method for each cycle structure C of N instead for N. If for some C, a feasible schedule cannot be found, terminate.*

Step 4. *Add a cycle C' of length zero to each cycle structure C, which results in network $\widetilde{N}$.*

Step 5. *Perform Step 3 of the direct method for network $\widetilde{N}$ instead for N.*

To construct a feasible schedule (whether for network N in the direct method or for a cycle structure or the modified network $\widetilde{N}$ in pr decomposition methods), the *serial schedule generation scheme* is used, which will be discussed in Subsection 4.3.

4.2 Priority rules

Priority rules are used for successively scheduling the activities according to the serial schedule generation scheme. We consider three subsets of activity set V. The *completed set* $\mathcal{C}$ contains all activities that have already been scheduled. We start with $S_0 := 0$ and $\mathcal{C} = \{0\}$. Moreover, we need set $\overline{\mathcal{C}} := V \setminus \mathcal{C}$ and the *eligible set* $\mathcal{E} := \{j \in \overline{\mathcal{C}} | Pred^{\prec}(j) \subseteq \mathcal{C}\}$. The activity to be scheduled next is always that activity $j^* \in \mathcal{E}$ which has highest *priority* $\pi(j^*)$, where ties are broken on the basis of increasing activity numbers, that is,

$$j^* = \min\{j \in \mathcal{E} | \pi(j) = \text{ext}_{h \in \mathcal{E}} \pi(h)\}$$

with ext $\in \{\min, \max\}$. Notice that for ext $= \min$ (or $= \max$), the activity $j = j^*$ with highest priority has the smallest (or largest, respectively) priority-rule value $\pi(j)$. A large number of different *priority rules* have been examined in literature (see e.g. Kolisch, 1996b; Franck and Neumann, 1998). We only list some of them which have turned out to be best in an experimental performance analysis (cf. Section 6):

LST rule (smallest "latest start time" first): $\text{ext}_{h \in \mathcal{E}} \pi(h) = \min_{h \in \mathcal{E}} LS_h$

MST rule ("minimum slack time" first): $\text{ext}_{h \in \mathcal{E}} \pi(h) = \min_{h \in \mathcal{E}} TF_h$, where $TF_h := LS_h - ES_h$ is the total float or slack time of activity h

MTS rule ("most total successors" first): $\text{ext}_{h \in \mathcal{E}} \pi(h) = \max_{h \in \mathcal{E}} |Reach^{\prec}(h)|$, where $Reach^{\prec}(h)$ is the set of total (i.e. not necessarily immediate) successors of node h with respect to strict order $\prec$, that is, the *reachable set*

LPF rule ("longest path following" first): $\text{ext}_{h \in \mathcal{E}} \pi(h) = \max_{h \in \mathcal{E}} l(h)$, where the *level* $l(h)$ of node h is the maximum number of nodes on any longest path from h to $n + 1$

RSM rule ("resource scheduling method"): $\text{ext}_{h \in \mathcal{E}} \pi(h) = \min_{h \in \mathcal{E}} \max[0, \max_{g \in \mathcal{E} \setminus \{h\}} (ES_h + p_h - LS_g)]$. The RSM rule says that an activity which induces the smallest delay of every other activity from the eligible set is scheduled next, where we assume that each activity g must be delayed up to the end of activity h.

4.3 Schedule generation schemes

In Kolisch (1996a) and Kolisch and Hartmann (1999) a *serial* and a *parallel schedule generation scheme* for project scheduling with minimum time lags have been discussed. In what follows, we sketch the adaptation of the serial schedule generation scheme to our more general problem. To find the earliest possible start time

$t^* \geq ES_{j^*}$ of activity j^* to be scheduled next, we have to ensure that for each $k \in \mathcal{R}$ and all time points τ at which j^* is in execution, the amount of resource k needed for all $i \in \mathcal{C}$, denoted by $r_k(\tau)$, plus the amount used by j^* does not exceed R_k. If $t^* \leq LS_{j^*}$, then j^* can be started at time t^*, and we update ES_j and LS_j for all $j \in \overline{\mathcal{C}}$.

If $t^* > LS_{j^*}$, then t^* is not time-feasible, and we perform a so-called *unscheduling step*. In general, the latest start time LS_{j^*} results from a maximum time lag $d_{ij^*}^{max} = -d_{j^*i}$ between the start of some real activity $i \in \mathcal{C}$ and activity j^*, i.e., $LS_{j^*} = S_i - d_{j^*i}$. Let

$$\mathcal{U} := \{i \in \mathcal{C} | LS_{j^*} = S_i - d_{j^*i}\}$$

be the set of all those activities scheduled. To increase LS_{j^*}, we unschedule all activities $i \in \mathcal{U}$ and increase start times S_i by $t^* - LS_{j^*}$ for all $i \in \mathcal{U}$. In addition, we unschedule all activities $i \in \mathcal{C}$ with $S_i > \min_{h \in \mathcal{U}} S_h$, which due to the right-shift of the activities from set $\mathcal{U}$ may possibly be started earlier. If there is no real activity $i \in \mathcal{C}$ whose scheduling has led to a decrease of LS_{j^*}, i.e., $LS_{j^*} = -d_{j^*0}$ and thus $0 \in \mathcal{U}$, the schedule generation scheme is terminated because no feasible schedule can be found. It is recommended to prescribe the maximum number $\overline{u}$ of unscheduling steps, e.g. $\overline{u} = |V|$. The basic version of the serial schedule generation scheme is then as follows, where u is the number of unscheduling steps performed.

Algorithm 6 *Serial schedule generation scheme with unscheduling step*

$S_0 := 0, \mathcal{C} := \{0\}, \overline{\mathcal{C}} := V \setminus \{0\}, u := 0$
For *all* $k \in \mathcal{R}$ *initialize resource profiles* $r_k(\cdot)$
While $\overline{\mathcal{C}} \neq \emptyset$ **do**
  $\mathcal{E} := \{j \in \overline{\mathcal{C}} | Pred^{\prec}(j) \subseteq \mathcal{C}\}$
  For *all* $j \in \mathcal{E}$ *compute* $\pi(j)$
  $j^* := \min\{j \in \mathcal{E} | \pi(j) = \text{ext}_{h \in \mathcal{E}} \pi(h)\}$
  $t^* := \min\{t \geq ES_{j^*} | r_k(\tau) + r_{j^*k} \leq R_k$ *for* $\tau = t, \ldots, t + p_{j^*} - 1$ *and all* $k \in \mathcal{R}\}$
  If $t^* > LS_{j^*}$ **then** $u := u + 1$ *and* **Unschedule**$(j^*, t^* - LS_{j^*})$
  else $(*$ *schedule* j^* *at time* t^* $*)$
    $S_{j^*} := t^*, \mathcal{C} := \mathcal{C} \cup \{j^*\}, \overline{\mathcal{C}} := \overline{\mathcal{C}} \setminus \{j^*\}$
    For *all* $k \in \mathcal{R}$ *update resource profiles* $r_k(\cdot)$
    For *all* $j \in \overline{\mathcal{C}}$ **do** $(*$ *update* ES_j *and* LS_j $*)$
      $ES_j := \max(ES_j, S_{j^*} + d_{j^*j})$
      $LS_j := \min(LS_j, S_{j^*} - d_{jj^*})$
    end $(*$ *for* $*)$
  end $(*$ *else* $*)$
end $(*$ *while* $*)$

Unschedule(j^*, Δ)

$\mathcal{U} := \{i \in \mathcal{C} | LS_{j^*} = S_i - d_{j^*i}\}$
If $0 \in \mathcal{U}$ **or** $u > \overline{u}$ **then** *terminate* $(*$ *no feasible schedule is found* $*)$

Step 1 ($*$ *right-shift of activities* $i \in \mathcal{U}$ $*$)
For *all* $i \in \mathcal{U}$ **do** $ES_i := S_i + \Delta$, $\mathcal{C} := \mathcal{C} \setminus \{i\}$, $\overline{\mathcal{C}} := \overline{\mathcal{C}} \cup \{i\}$
Step 2 ($*$ *unschedule all activities* i *with* $S_i > \min_{h \in \mathcal{U}} S_h$ $*$)
For *all* $i \in \mathcal{C}$ *with* $S_i > \min_{h \in \mathcal{U}} S_h$ **do** $\mathcal{C} := \mathcal{C} \setminus \{i\}$, $\overline{\mathcal{C}} := \overline{\mathcal{C}} \cup \{i\}$
Step 3 ($*$ *Recalculate resource profiles* $*$)
For *all* $k \in \mathcal{R}$ *recalculate resource profiles* $r_k(\cdot)$
Step 4 ($*$ *compute* ES_j *and* LS_j *for all* $j \in \overline{\mathcal{C}}$ *again* $*$)
For *all* $j \in \overline{\mathcal{C}}$ **do**
  $ES_j := \max[d_{0j}, \max_{h \in \mathcal{U}}(ES_h + d_{hj})]$
  $LS_j := -d_{j0}$
  For *all* $i \in \mathcal{C}$ **do**
    $ES_j := \max(ES_j, S_i + d_{ij})$
    $LS_j := \min(LS_j, S_i - d_{ji})$
  end ($*$ *for* $*$)
end ($*$ *for* $*$)

Algorithm 6 does not guarantee that the activities of one and the same cycle structure are scheduled directly one after another as required for the direct method (cf. Step 3 in Algorithm 4). To ensure the latter, we redefine the eligible set as follows:

$$\mathcal{E} := \begin{cases} \{j \in \overline{\mathcal{C}} \cap V^C | Pred^{\prec_C}(j) \subseteq \mathcal{C}\}, & \text{if some but not all nodes from some cycle structure } C \text{ have been scheduled} \\ \{j \in \overline{\mathcal{C}} | Pred^{\prec_C}(j) \subseteq \mathcal{C}\}, & \text{otherwise} \end{cases} \tag{10}$$

$Pred^{\prec_C}(j)$ is the set of immediate predecessors of node j with respect to strict order $\prec_C$ which is defined as follows. Let

$$V(i) := \begin{cases} V^C, \text{ if there is a cycle structure } C \text{ with } i \in V^C \\ \{i\}, \text{ otherwise} \end{cases} \tag{11}$$

be the node set of the strong component node i belongs to. Then for $i, j \in V$, $i \neq j$, we define

$i \prec_C j$ exactly if $i \prec j$,
or $V(i) \neq V(j)$ as well as $g \prec h$ for some $g \in V(i), h \in V(j)$,
or $i \prec_C h$ as well as $h \prec_C j$ for some $h \in V$

Strict order $\prec_C$ means that each cycle structure can be viewed as a single (contracted) activity.

Let $q(|\mathcal{R}|, |V|)$ be a bivariate polynomial describing the time complexity of computing the priority $\pi(j)$ for some eligible activity $j \in \mathcal{E}$. The update of all resource profiles $r_k(\cdot)$ after the scheduling of activity j^* can be done in $O(|\mathcal{R}||\mathcal{C}|)$ time. The time complexity for one iteration of the serial schedule generation scheme if procedure **Unschedule** is not called is then $O(\max(|\mathcal{R}||V|, |V|q(|\mathcal{R}|, |V|))$. One execution of procedure **Unschedule** requires $O(|\mathcal{R}||V|^2)$ time. Before **Unschedule** is called for the first time or between two calls of **Unschedule**, at most n iterations of the generation scheme are performed. Thus, the time complexity of Algorithm 6 is $O(\overline{u}\,|V|^2 \max[|\mathcal{R}|, q(|\mathcal{R}|, |V|)] + |\mathcal{R}||V|^2)) = O(\overline{u}\,|V|^2 \max[|\mathcal{R}|, q(|\mathcal{R}|, |V|)])$.

5 Schedule-improvement procedures

In this section we discuss a genetic algorithm and a tabu search approach for improving a schedule determined by the direct method from Section 4. For a detailed description of the algorithms, we refer to Franck and Selle (1998) or Franck (1999, Ch. 6).

We represent a schedule S by an *activity list* $L = (i_0, i_1, \ldots, i_n, i_{n+1})$, where $(i_0, i_1, \ldots, i_n, i_{n+1})$ is a permutation of $(0, 1, \ldots, n, n+1)$ with $i_0 = 0$ and $i_{n+1} = n+1$. L can be transformed into schedule S by the serial schedule generation scheme, where $\mu < \nu$ implies that i_μ has a higher priority than i_ν.

The number of activity lists corresponding to one and the same schedule can be reduced by using the concept of so-called precedence-feasible activity lists (cf. Kolisch and Hartmann, 1999), which take strict order $\prec_C$ and the cycle structures of the project network into account. For activity list L, let $\prec_L$ be the linear order given by $i_\mu \prec_L i_\nu$ exactly if $\mu < \nu$.

Definition 1 *An activity list L is called precedence-feasible if*

(a) $i \prec_C j$ implies $i \prec_L j$ and

(b) for each cycle structure C and all activities $h, i, j \in V^C$, $i \prec_L h \prec_L j$ implies $h \in V^C$.

By replacing strict order $\prec$ or $\prec_C$ with linear order $\prec_L$ in Algorithm 6, the eligible set reduces to the singleton $\{j^*\}$, which contains activity j^* to be scheduled next. Thus, (10) can be rewritten as

$$\mathcal{E} := \{j \in \overline{\mathcal{C}} | Pred^{\prec_L}(j) \subseteq \mathcal{C}\}$$

5.1 Genetic algorithm

A genetic algorithm works on a *population* consisting of several schedules called *individuals*. An individual is represented by a precedence-feasible activity list, which is transformed into a corresponding schedule S. The "quality" of the schedule is given by the *fitness* of the individual. In each iteration of the genetic algorithm, we select two different individuals (called father and mother) from the population. We obtain two new individuals (two *children* called daughter and son) by applying some *crossover-operator*. Finally, the two children are subject to *mutation* with a certain probability, and we obtain a new *generation* of the population by replacing two individuals with the two children. These steps are iterated until one out of several stop criteria is met.

The *parallel genetic algorithm* proposed in what follows simultaneously works on several subpopulations of equal size (so-called *islands*, see e.g. Dorigo and Maniezzo, 1993). Each subpopulation evolves separately, until after mig iterations an individual migrates from one island to another. We choose the individual with the best fitness from a first island. That individual then replaces one individual with worst fitness on any other island if it is better than the latter. The migrating individual is always chosen from an island the smallest number of individuals have migrated from.

To determine the activity lists of the initial subpopulations, we generate different precedence-feasible activity lists based on priorities determined randomly (this type of encoding is referred to as random key representation, cf. Bean, 1994; for alternative representations, we refer to Hartmann and Kolisch, 1999, and Hartmann, 1999, Ch. 5). Computational experience (cf. Franck, 1999, Sect. 6.3.2) has shown that it is expedient to draw the random priority $\pi(i)$ for an activity i from interval $[0.8\pi'(i), 1.2\pi'(i)]$ where $\pi'(i)$ is the value of a specific priority rule (cf. Subsection 4.2). Using the priorities $\pi(i)$, an activity list is determined by choosing i_ν equal to the activity with highest priority among the activities for which all $\prec_C$-predecessors are contained in sublist $(i_0, i_1, \ldots, i_{\nu-1})$, $\nu = 1, \ldots, n$.

If it is impossible to transform an activity list into a feasible schedule by the direct method within $\overline{u}$ unscheduling steps, we allow time-infeasibility by scheduling some activities $i \in V$ at times $S_i > LS_i$. The fitness of activity list L with corresponding schedule S is given by

$$\widetilde{f}(L) := \begin{cases} \max[S_{n+1}, \widetilde{f}(L^*)] + \sum_{i \in V} (S_i - LS_i)^+, \text{ if } S \notin \mathcal{S}_T \\ S_{n+1}, \qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

where $\widetilde{f}(L^*)$ denotes the fitness of the best feasible individual L^* (i.e. the shortest project duration) found so far and $(z)^+ := \max(z, 0)$. If a feasible schedule has not been found yet, we set $\widetilde{f}(L^*) := \overline{d}$. Notice that an individual with best fitness is an individual for which the fitness $\widetilde{f}(L)$ is *minimum*.

The selection of two parents for a crossover is performed according to a *roulette-based tournament-selection strategy*. To this end, we randomly choose a certain number of individuals of the current population, where the probability for an individual to be chosen (figuratively speaking, the size of the individual's slot on a roulette wheel) is reciprocally proportional to its fitness. The best of the individuals chosen is selected to be the father. The selection of the mother is made analogously.

We distinguish between two types of crossover, which are based on crossover operators that have been introduced by Hartmann (1998) for the resource-constrained project scheduling problem without maximum time lags. Let F be the father and M the mother of daughter D. For the *one-point crossover*, we randomly draw a number q from set $\{0, \ldots, n+2\}$. The first q activities of the activity list of D are taken from the activity list of F, and the remaining $n + 2 - q$ activities are taken from the activity list of M. If the qth activity belongs to a cycle structure C and if there is at least one activity in V^C which is not represented in D before activities are taken from M, we say that cycle structure C is *cut*. In that case, we take activities of V^C which are not contained in D from mother M, where we start at the first position in the activity list of M. Then, we add the remaining activities from M to the still incomplete activity list of D starting a second time at the first position in activity list of M, that is, we add the missing activities according to their position in the activity list of M. Figure 1 illustrates the case of a one-point crossover cutting some cycle structure C.

For the *two-point crossover*, we randomly select two numbers q_1 and q_2 with $q_1 < q_2$ from set $\{0, \ldots, n+2\}$. The first q_1 activities of activity list of D are

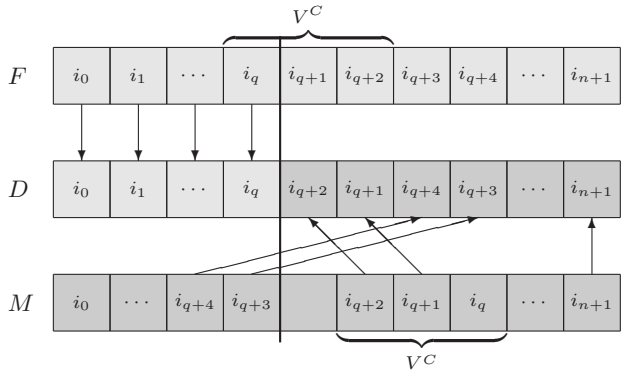

Fig. 1. One-point crossover

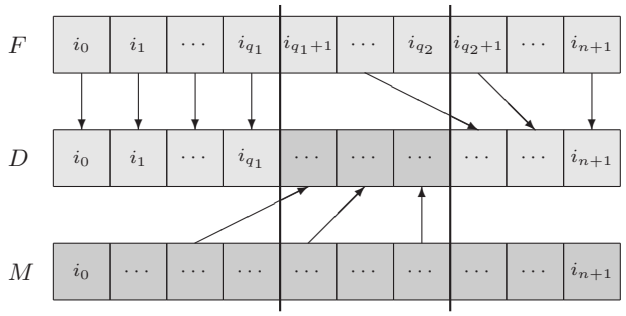

Fig. 2. Two-point crossover

taken from F, the next $q_2 - q_1$ activities are taken from M, and the remaining activities are again taken from F (cf. Fig. 2). If a cycle structures is cut, we proceed analogously to the one-point crossover.

If for the one-point or two-point crossover we interchange the roles of F and M, we obtain a son. The children generated by crossover constitute precedence-feasible activity lists if the activity lists F and M are precedence-feasible (cf. Franck and Selle, 1998).

A *mutation* of an individual results from interchanging two consecutive activities i_μ and $i_{\mu+1}$ with $\mu = 1, \ldots, n$ of a child's activity list. For activities i_μ and $i_{\mu+1}$ it must hold that $i_\mu \not\prec_C i_{\mu+1}$ and either $i_{\mu+1} \in V(i_\mu)$ or $|V(i_\mu)| = |V(i_{\mu+1})| = 1$. Mutation occurs with a certain *mutation probability* p_{mut}.

For the evolution of the subpopulations, we use the $(\mu + \lambda)$-*strategy* (cf. Michalewicz, 1998, Sect. 8.1), which is also called *steady state strategy*. After the crossover, the child generated is accepted if and only if its fitness is better than the worst fitness of all individuals in the subpopulation at hand. The child then replaces one of the individuals with worst fitness. Thus, the size of all subpopulations is constant over time. The genetic algorithm terminates if at least one of the following five stop criteria is met: all individuals have the same fitness, a lower bound

on the shortest project duration has been attained ($\widetilde{f}(L^*) = \max\{LBR, LBD\}$), a prescribed maximum number $sched$ of schedules have been evaluated, a feasible schedule has not been found within a prescribed maximum number $notfeas$ of generations, or the best feasible schedule found has not been improved within a given number $notimp$ of generations.

In summary, the genetic algorithm is as follows.

Algorithm 7 *Genetic Algorithm*

Determine initial activity lists for the islands based on random priorities.
Compute the fitness of each initial activity list.
$gen := 0$.
For *each island* **do**
Select two parents.
Generate two children by one-point or two-point crossover.
Mutate the children with probability p_{mut}.
Compute the fitness of the children.
Replace the worst two individuals of the population by the children if the children have a better fitness.
$gen := gen + 1$.
If gen *is a multiple of* mig **then** *perform migration.*
Until *some stop criterion is met.*

5.2 Tabu search

Tabu search is a well-known heuristic approach for solving combinatorial optimization problems. In each *iteration* of tabu search, the best (feasible or infeasible) schedule of a *neighborhood* of a current schedule is selected. The neighborhood is generated using some *neighborhood operators*. To avoid cycling in the search process, we keep *tabu lists* of schedules or neighborhood operators. Elements contained in a tabu list are said to be *tabu*. A schedule which is tabu cannot be selected and a neighborhood operator which is tabu cannot be applied when generating the neighborhood.

The concept of *intensification* is used to explore subsets of the feasible region in more detail, whereas *diversification* allows the search process to investigate unvisited parts of the feasible region. For further details on tabu search we refer to Glover and Laguna (1997). In the following, we briefly sketch the neighborhood operators, diversification, intensification, and the stop criteria of a tabu search procedure for problem $PS|temp|C_{\max}$ which is based on an algorithm by Baar et al. (1998) for the resource-constrained project duration problem where there are only minimum time lags corresponding to precedence constraints between activities.

As for the genetic algorithm, a schedule is represented by a precedence-feasible activity list. We obtain a neighbor L' of an activity list L by selecting a pair (i, j) of activities from L to which we apply one of four different neighborhood operators. Let S be the schedule belonging to activity list L. To determine (i, j) we first scan V for activities j with start time $S_j > \widetilde{ES}_j := \max[d_{0j}, \max_{\langle h,j\rangle \in E}(S_h + \delta_{hj})]$.

For each such activity j, we then determine activities i with $i \not\prec_C j$ which (due to a resource conflict) delay activity j up to S_j, i.e., for which $S_i + p_i = S_j$ holds true.

If $i \prec_L j$, we apply the shift or swap operator, otherwise the back-shift or front-shift operator. In a first step, the operators change the position of activity i or j in the current activity list L. In a second step, the operators repair the new activity list in order to obtain a new precedence-feasible activity list L'. The four operators are illustrated in Figure 3.

First, we assume that $i \prec_L j$. The *shift operator* removes activity i and inserts it behind j in activity list L. If $V(i) \neq V(j)$, $V(i)$ is placed behind the last activity of $V(j)$, where the sequence of activities in $V(i)$ and $V(j)$ remains unchanged. The resulting list is repaired by appropriately shifting all activities l with $i \prec_C l$ and $i \prec_L l \prec_L j$ behind $V(i)$.

The *swap operator* interchanges the positions of i and j in activity list L if $V(i) = V(j)$, or the positions of $V(i)$ and $V(j)$ if $V(i) \neq V(j)$. Subsequently, activities l_1 with $i \prec_C l_1$ and $i \prec_L l_1 \prec_L j$ are shifted behind $V(i)$, and activities l_2 with $l_2 \prec_C j$ and $i \prec_L l_2 \prec_L j$ are placed in front of $V(j)$. Due to $i \not\prec_C j$ it holds that $l_1 \not\prec_C l_2$, and thus the repaired list L' is precedence-feasible.

Now let $j \prec_L i$. The *back-shift operator* removes the first activity h in L with $i \prec_L h$ and $i \not\prec_C h$ and inserts it immediately before i (and thus behind j) if $h \in V(i)$. If $V(h) \neq V(i)$, $V(h)$ is placed before the first activity of $V(i)$. Since h is the first activity after i in list L with $i \not\prec_C h$, it holds that $i \prec_C l$ and consequently $l \not\prec_C h$ for all l with $i \prec_L l \prec_L h$, and thus the resulting list L' is precedence-feasible.

The *front-shift operator* removes the last activity h with $h \prec_L j$ and $h \not\prec_C j$ and places it immediately behind j (and thus before i) if $h \in V(j)$ or transfers $V(h)$ behind the last activity of $V(j)$, otherwise. Similarly to the back-shift operator, the front-shift operator yields precedence-feasible lists L'.

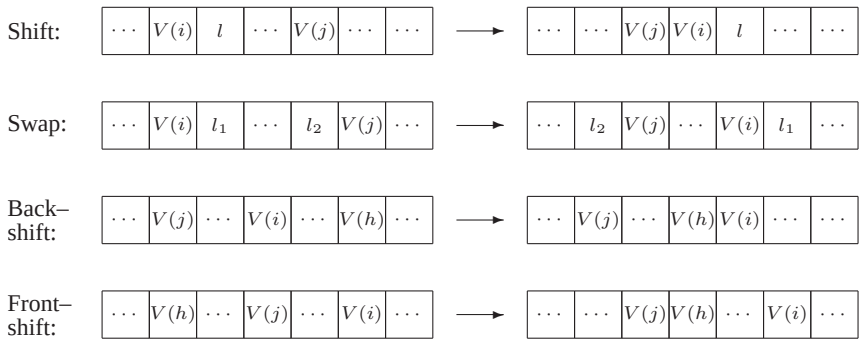

Fig. 3. Shift, swap, back-shift, and front-shift operators

The computation time needed for the transformation of an activity list into a schedule increases in the number of activities. To restrict the computational effort and to perform a sufficiently large number of iterations, we reduce the number of neighbors generated and investigated for large-size instances (say, for $n \geq 200$). To

this end, we select only a prescribed number of activities j with largest differences $S_j - \widetilde{ES}_j$ and determine only two activity pairs (i, j) for each of these activities j (*reduced neighborhood*).

We use three different tabu lists for storing activity lists explored and neighborhood operators applied. The first tabu list contains the activity lists of the last $tlsize_1$ iterations. The second tabu list contains shifts and swaps and the third contains back-shifts and front-shifts of the last $tlsize_2$ iterations. Each time we turn to a new activity list L, we delete an activity list from the tail (provided that the tabu list has already been filled) and add L at the head of the first tabu list. If list L has been generated by a shift or swap operator applied to activity pair (i, j), we delete two elements from the tail of the second tabu list and add the pairs (i, j) and (j, i) at the head. Otherwise, L has been generated by a front-shift or back-shift operator, and we pop two elements from the tail of the third tabu list and add the pairs (i, j) and (j, i) at the head.

If the project duration has not been reduced within the last $notimp$ iterations, we apply the following diversification. We select the activity pair (i, j) to which we apply the neighborhood operators according to a modified criterion: given activity j, we determine activities $i \in V$ with $i \not\prec_C j$ and $\widetilde{ES}_j < S_i + p_i \leq S_j$ (*extended neighborhood*). If this is not sufficient to reduce the project duration, we diversify the current schedule as follows. We generate a certain number (e.g. $\lceil n/20 \rceil$) of activity lists by randomly choosing pairs (i, j) with $i \prec_L j$ and $i \not\prec_C j$ and applying the shift or swap operator to current list L because those two operators induce the largest number of changes in an activity list. We then select a list with best corresponding schedule to be the current schedule.

After a prescribed maximum number $iter$ of iterations or after the evaluation of a given maximum number $sched$ of schedules (*stop criteria*), we perform intensification before stopping the algorithm, that is, we return to the best schedule found. It turns out to be expedient to empty the three tabu lists twice and to perform three iterations, where we start with the best activity list found so far. After the intensification phase, the tabu search algorithm is terminated.

If we cannot transform an activity list L into a feasible schedule within $\overline{u}$ unscheduling steps by the direct method, we allow resource-infeasible schedules. The cost function $\widetilde{f}(L)$ of activity list L with corresponding schedule S (the analogue to the fitness in genetic algorithms) is given by

$$\widetilde{f}(L) := \begin{cases} \max[S_{n+1}, \widetilde{f}(L^*)] + \sum_{k \in \mathcal{R}} \int_0^{S_{n+1}} (r_k(S,t) - R_k)^+ dt, & \text{if } S \notin \mathcal{S}_R \\ S_{n+1}, & \text{otherwise} \end{cases}$$

where $\widetilde{f}(L^*)$ again denotes the project duration of the best feasible schedule found thus far (if no feasible schedule has been found yet, $\widetilde{f}(L^*)$ again equals $\overline{d}$).

The tabu search procedure is summarized in Algorithm 8.

Algorithm 8 *Tabu Search*

Find an initial schedule S by the direct method.
Determine a precedence-feasible activity list L corresponding to S.

Repeat

For all activities $j \in V$ with $\widetilde{ES}_j < S_j$, determine the activities $i \in V$ with $S_i + p_i = S_j$ and $i \not\prec_C j$ (if necessary, use the extended or reduced neighborhood).
Apply neighborhood operators to list L and all non-tabu activity pairs (i, j) determined.
For resulting activity lists L', compute corresponding schedules S' by the serial schedule generation scheme.
Select a non-tabu activity list L' with smallest $\widetilde{f}(L')$ as new current activity list L.
Update the three tabu lists.
If within the last $notimp$ iterations, the project duration of the best feasible schedule found has not been reduced, apply diversification.

Until *one of the two stop criteria is met.*
Perform intensification.

6 Experimental performance analysis

In this section we discuss the performance of the algorithms presented in Sections 3, 4, and 5. The different heuristics are evaluated on the basis of three criteria:

(a) the percentage of instances for which either a feasible solution could be found or which could be shown to be unsolvable (p_{feas}),
(b) the mean percentage deviation of the project duration found from the tightest of the lower bounds LBW and LBD (Δ_{LB}), which represents an upper bound on the mean error of the project duration computed.
(c) the mean computation time in seconds (t_{cpu}).

Whereas p_{feas} and t_{cpu} refer to all instances, Δ_{LB} corresponds to the average deviation from lower bound of those instances which could be solved to feasibility by all algorithms compared in the respective table.

The instances of the test set have been generated by the project generator ProGen/max (cf. Schwindt, 1998a) and are available via world-wide-web at `www.wior.uni-karlsruhe.de/rcpspmax/progenmax` (test sets `UBO`). ProGen/max allows the parameter-driven construction of instances of resource-constrained project scheduling problems. The three parameters which have the largest impact on the hardness of $PS|temp|C_{max}$ instances are the number n of activities, the order strength OS of the project network, and the resource strength RS of the project.

The order strength is a measure of the percentage of activity sequences which comply with the precedence relationships between activities given by strict order $\prec$. $OS = 0$ implies that observing the temporal constraints, all activities can be executed simultaneously, whereas $OS = 1$ means that between any two activities $i, j \in V$ there is a minimum time lag $d_{ij} \geq 0$ or $d_{ji} \geq 0$ such that all resource conflicts can be resolved in a unique way. As a rule, the lower the order strength of a project the harder is the problem of finding an optimal schedule.

The resource strength introduced by Kolisch et al. (1995) reflects the scarcity of resources. $RS = 0$ means that the resources only have the minimum capacity

necessary to process all activities one after the other. If $RS = 1$, the earliest schedule ES is resource-feasible and thus represents an optimal solution. Hence, projects with large RS can be viewed as easy instances whereas instances with small RS are hard. On the other hand, a resource strength $RS = 0$ means that many two-element forbidden sets can already be dealt with in the preprocessing phase. Moreover, the number of minimal delaying alternatives for a forbidden set F represents a bell-shaped function in RS, where the maximizer decreases with the cardinality of F (cf. Elmaghraby and Herroelen, 1980). Thus, projects with tight resource constraints are generally better tractable than projects whose resource capacities allow the overlapping of a larger number of activities.

The three parameters have been chosen as $n \in \{10, 20, 50, 100, 200, 500, 1000\}$, $OS \in \{0.25, 0.5, 0.75\}$, and $RS \in \{0.0, 0.25, 0.5\}$. Ten instances have been generated for each combination according to a full-factorial design. All 630 projects have five renewable resources, and 75% of the resource requirements r_{ik} ($i = 1, \ldots, n$, $k \in \mathcal{R}$) are positive, which corresponds to a resource factor of $RF = 0.75$.

533 instances possess a feasible solution. For the remaining 97 instances, unsolvability could be shown either by preprocessing or by the bb decomposition methods. Table 1 lists the percentage of solvable instances as a function of n.

Table 1. Percentage of solvable instances

$n = 10$	$n = 20$	$n = 50$	$n = 100$	$n = 200$	$n = 500$	$n = 1000$
81.11%	77.78%	81.11%	86.67%	88.89%	87.78%	88.89%

The tests have been performed on a 333 MHz clock pulse PII personal computer with 128 MB RAM running NT 4.0 as operating system. For the truncated branch-and-bound procedures, we have imposed a time limit of n seconds. The maximum number of calls to procedure **Unschedule** for one execution of the direct method has been fixed to $10\sqrt{n}$. For the genetic algorithm, the number of islands and the size of the subpopulations per island have been chosen as 5 and 50, respectively. The initial five subpopulations have arisen from biasing the values of the five priority rules as described in Subsection 5.1. Table 2 provides the settings for the remaining parameters of the genetic algorithm and tabu search procedure.

Table 2. Parameter settings for genetic algorithm | tabu search procedure

mig	p_{mut}	$sched$	$notfeas$	$notimp$	$tlsize_1$	$tlsize_2$	$sched$	$notimp$	$iter$
10	0.01	5000	20	100	20	$0.15 \cdot n$	5000	5	100

First we consider the different truncated branch-and-bound procedures. For both decomposition methods, half of the available computation time is allotted to the scheduling of the individual cycle structures, the remaining time is used for solving the problem on the extended network. As shown in Table 3, the decomposition methods determines a feasible schedule for all 533 solvable instances. However,

we pay for this reliability with a high deviation from lower bound compared to the approximation method and filtered beam search. In contrast to decomposition method 1, the extended network constructed by decomposition method 2 contains cycles of negative length, i.e., not all activities of a cycle structure are pulled tight. This flexibility leads to a significant decrease of Δ_{LB}. Due to the imposed time limits, the enumeration is generally incomplete when the algorithm stops. Nevertheless, the a priori restriction in size of the enumeration tree allows the filtered beam search to (briefly) visit different parts of the complete enumeration tree and thus to solve more instances to feasibility than the approximation method.

Table 3. Comparison of truncated branch-and-bound procedures

	Approx. m.	Bb decomp. 1	Bb decomp. 2	Filt. beam s.
p_{feas}	92.31%	100.00%	100.00%	94.75%
Δ_{LB}	10.63%	25.07%	17.51%	8.87%
t_{cpu}	93.05	61.66	54.55	125.47

Table 4 compares the performance of the different priority rules for the serial schedule generation scheme of the direct method. The LST rule outperforms the four remaining rules, as far as the number of problems solved to feasibility and the mean deviation from lower bound are concerned. This observation is in line with results obtained by Kolisch (1995) for the resource-constrained project scheduling problem without maximum time lags. Notice that in contrast to the approximation method and filtered beam search, the direct method with rules LST and RSM solves all instances to feasibility. This is mainly due to the fact that for large instances, activities can be unscheduled much faster by procedure **Unschedule** than by backtracking in a depth-first search tree.

Table 4. Comparison of priority rules for direct method

	LST	MST	MTS	LPF	RSM
p_{feas}	100.00%	99.81%	99.62%	99.25%	100.00%
Δ_{LB}	15.50%	18.46%	16.16%	16.72%	16.48%
t_{cpu}	8.99	8.93	9.01	8.89	9.06

The results for the direct method and the two priority-rule decomposition methods are given by Table 5. All instances are solved to feasibility by each of the three algorithms, where we use the five priority rules and select a best of the five schedules generated (in Kolisch and Hartmann, 1999, this approach is termed multi-pass priority-rule based scheduling). By using the multi-pass technique, the least mean deviation Δ_{LB} in Table 4 (corresponding to rule LST) is decreased from 16.1% to 14.6%. Since the running time of the priority-rule methods is small, it is generally recommended to combine several rules. In contrast to truncated branch-and-bound procedures, the decomposition approach does not prove useful for priority-rule methods since the direct method is faster and solves all instances to feasibility.

Analogously to Table 3, in particular pr decomposition method 1 results in a considerable increase in Δ_{LB}.

Table 5. Comparison of priority-rule methods

	Direct meth.	Pr decomp. 1	Pr decomp. 2
p_{feas}	100.00%	100.00%	100.00%
Δ_{LB}	14.47%	28.04%	21.49%
t_{cpu}	13.76	69.22	51.53

We now turn to the schedule-improvement procedures. Table 6 depicts the results for the genetic algorithm and tabu search heuristic. Both algorithms are quite time-consuming, however, they provide "better" schedules. The genetic algorithm performs slightly better than the tabu search procedure but requires more computing time.

Table 6. Comparison of schedule-improvement procedures

	Genetic algorithm	Tabu search
p_{feas}	100.00%	100.00%
Δ_{LB}	11.12%	12.61%
t_{cpu}	1894.75	1625.39

Finally, we examine how truncated branch-and-bound procedures, priority-rule methods, and schedule-improvement algorithms behave for solving small instances on the one hand and large instances on the other hand. Table 7 provides the results for the 360 projects with $n \leq 100$. Table 8 refers to the remaining 270 instances of the test set with $n \geq 200$. We compare the best truncated branch-and-bound procedure (filtered beam search), the best priority-rule method (direct method combining all five priority rules), the genetic algorithm, and the tabu search approach. When coping with small projects, filtered beam search and the genetic algorithm seem to be best. In a relatively short computing time, (almost) all instances are solved to feasibility, and Δ_{LB} is smaller than for the remaining algorithms. For large projects, the direct priority-rule method should be used. If enough computation time is available, schedule-improvement procedures provide substantially better schedules.

Table 7. Comparison between filtered beam search, direct method, genetic algorithm, and tabu search – "small" instances with $n \leq 100$

	Filt. beam s.	Direct meth.	Genetic algorithm	Tabu search
p_{feas}	99.66%	100.00%	100.00%	100.00%
Δ_{LB}	6.82%	10.72%	6.93%	8.48%
t_{cpu}	12.40	0.03	3.16	30.00

Table 8. Comparison between filtered beam search, direct method, genetic algorithm, and tabu search – "large" instances with $n \geq 200$

	Filt. beam s.	Direct meth.	Genetic algorithm	Tabu search
p_{feas}	88.70%	100.00%	100.00%	100.00%
Δ_{LB}	15.56%	11.85%	9.87%	11.03%
t_{cpu}	276.24	31.94	4411.77	3760.39

Eventually, we summarize the pros and cons of the different heuristic methods proposed. Truncated branch-and-bound methods are recommended when dealing with small projects, where filtered beam search performs best. For scheduling projects with hundreds of activities, priority rule methods should be used. Within less than one minute of computing time, the direct method combining several priority rules provides feasible schedules with relatively small deviation from optimum. This deviation can be considerably reduced by using schedule-improvement procedures, where the genetic algorithm seems to outperform the tabu search approach. Compared to the direct method, both the truncated branch-and-bound algorithms and schedule-improvement procedures are quite time-consuming. If in case of $\mathcal{S} \neq \emptyset$ we want the algorithm to compute a feasible schedule without fail, we recommend bb decomposition method 2 (without time limit). If in addition a performance guarantee ε has to be met, we have to use the approximation method (without time limit). In practice, this will only be possible for small projects or for projects with large order strength or resource strength.

Figure 4 shows the methods recommended depending on computation time t_{cpu} and mean deviation from lower bound Δ_{LB}.

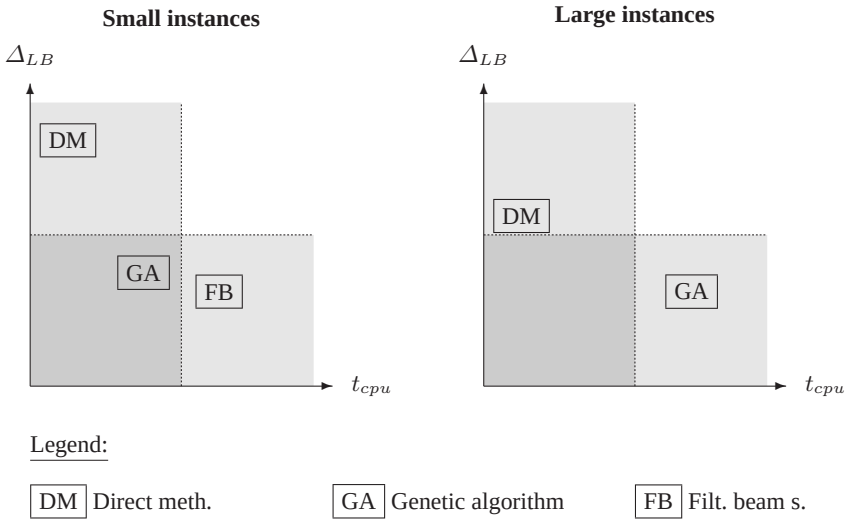

Fig. 4. Recommended methods depending on computation times and mean deviations from lower bound

7 Conclusions

Heuristic procedures for the resource-constrained project duration problem with general temporal constraints have been considered. In particular, truncated branch-and-bound techniques (approximation method, filtered beam search, and decomposition approach), priority-rule methods (direct and decomposition methods), and schedule-improvement procedures (genetic algorithm and tabu search) have been presented. An experimental performance analysis has shown that large problem instances with up to 1000 activities can be solved approximately in less than one minute.

An important area of future research is the development of efficient heuristic methods with similar performance for corresponding project scheduling problems with nonregular objective functions (for example, resource levelling, resource renting, and net present value problems, which have been briefly discussed in Brucker et al., 1999).

References

Baar T, Brucker P, Knust S (1998) Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem. In: Voss S, Martello S, Osman I, Roucairol C (eds.) Meta-heuristics: Advances and trends in local search paradigms for optimization, pp. 1–18. Kluwer, Boston

Baptiste P, Le Pape C, Nuijten W (1999) Satisfiability tests and time-bound adjustments for cumulative scheduling problems. Annals of Operations Research 92:305–333

Bartusch M, Möhring RH, Radermacher, FJ (1988) Scheduling project networks with resource constraints and time windows. Annals of Operations Research 16:201–240

Bean J (1994) Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 6:154–160

Brinkmann K, Neumann, K (1996) Heuristic procedures for resource-constrained project scheduling with minimal and maximal time-lags: The resource-levelling and minimum project duration problems. Journal of Decision Systems 5:129–155

Brucker P, Knust S, Schoo A, Thiele O (1998) A branch & bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 107:272–288

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112:3–41

Carlier J, Pinson E (1989) An algorithm for solving the job-shop problem. Management Science 35:164–176

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

De Reyck B (1998) Scheduling projects with generalized precedence relations – exact and heuristic procedures. Ph.D. Thesis, Catholic University of Leuven

De Reyck B, Herroelen W (1998) A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. European Journal of Operational Research 111:152–174

Dorigo M, Maniezzo V (1993) Parallel genetic algorithms: Introduction and overview of current research. In: Stender J (ed.) Parallel genetic algorithms: Theory and applications, pp. 151– 185. IOS Press, Amsterdam

Dorndorf U, Pesch E, Phan Huy T (1999) A survey of interval capacity consistency tests for time- and resource-constrained scheduling. In: Węglarz J (ed.) Project scheduling: Recent models, algorithms, and applications, pp. 214–238. Kluwer, Boston

Dorndorf U, Pesch E, Phan Huy T (2000) A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. Management Science 46:1365–1384

Elmaghraby SE, Herroelen WS (1980) On the measurement of complexity in activity networks. European Journal of Operational Research 5:223–234

Even S (1979) Graph algorithms. Pitman, London

Fest A, Möhring RH, Stork F, Uetz M (1999) Resource-constrained project scheduling with time windows: A branching scheme based on dynamic release dates. Technical Report 596, Department of Mathematics, Technical University of Berlin

Franck B (1999) Prioritätsregelverfahren für die ressourcenbeschränkte Projektplanung mit und ohne Kalender. Shaker, Aachen

Franck B, Neumann K (1998) Resource-constrained project scheduling with time windows: Structural questions and priority-rule methods. Technical Report WIOR-492, Institute for Economic Theory and Operations Research, University of Karlsruhe

Franck B, Neumann K, Schwindt C (1997) A capacity-oriented hierarchical approach to single-item and small-batch production planning using project-scheduling methods. OR Spektrum 19:77–85

Franck B, Selle T (1998) Metaheuristics for the resource-constrained project scheduling with schedule-dependent time windows. Technical Report WIOR-546, Institute for Economic Theory and Operations Research, University of Karlsruhe

Garey MR, Johnson DS (1979) Computers and intractability. Freeman, New York

Glover F, Laguna F (1997) Tabu search. Kluwer, Boston

Hartmann S (1998) A competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics 45:733–750

Hartmann S (1999) Project scheduling under limited resources: Models, methods, and applications. Lecture Notes in Economics and Mathematical Systems, Vol. 478. Springer, Berlin Heidelberg New York

Hartmann S, Kolisch R (2000) Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. European Journal of Operational Research 127:394–407

Heilmann R, Schwindt C (1997) Lower bounds for RCPSP/max. Technical Report WIOR-511, Institute for Economic Theory and Operations Research, University of Karlsruhe

Herroelen WS, Demeulemeester EL, De Reyck B (1999) A classification scheme for project scheduling. In: Węglarz J (ed.) Project scheduling: Recent models, algorithms, and applications, pp. 1–26. Kluwer, Boston

Icmeli O, Erengüç SS (1996) A branch and bound procedure for the resource constrained project scheduling problem with discounted cash flows. Management Science 42:1395–1408

Klein R, Scholl A (1999) Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. European Journal of Operational Research 112:322–346

Kolisch, R (1995) Project scheduling under resource constraints. Physica, Heidelberg

Kolisch, R (1996a) Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. European Journal of Operational Research 90:320–333

Kolisch, R (1996b) Efficient priority rules for the resource-constrained project scheduling problem. Journal of Operations Management 14:179–192

Kolisch R, Hartmann S (1999) Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In: Węglarz J (ed.) Project scheduling: Recent models, algorithms, and applications, pp. 147–178. Kluwer, Boston

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

Martin P, Shmoys DB (1996) A new approach to computing optimal schedules for the job-shop scheduling problem. Proceedings of the fifth IPCO conference, Vancouver, 1996

Michalewicz Z (1998) Genetic algorithms + Data structures = Evolution programs. Springer, Berlin Heidelberg New York

Neumann K, Schwindt C (1997) Activity-on-node networks with minimal and maximal time lags and their application to make-to-order production. OR Spektrum 19:205–217

Neumann K, Zhan J (1995) Heuristics for the minimum project-duration problem with minimal and maximal time-lags under fixed resource constraints. Journal of Intelligent Manufacturing 6:145–154

Neumann K, Zimmermann J (1999) Methods for the resource-constrained project scheduling with regular and nonregular objective functions and schedule-dependent time windows. In: Węglarz J (ed.) Project scheduling: Recent models, algorithms, and applications, pp. 261–287. Kluwer, Boston

Nuijten W (1994) Time and resource constrained scheduling: A constraint satisfaction approach. Ph.D. Thesis, Eindhoven University of Technology

Pinedo M (1995) Scheduling theory, algorithms, and systems. Prentice Hall, Englewood Cliffs

Schwindt C (1998a) Generation of resource-constrained project scheduling problems subject to temporal constraints. Technical Report WIOR-543, Institute for Economic Theory and Operations Research, University of Karlsruhe

Schwindt C (1998b) Verfahren zur Lösung des ressourcenbeschränkten Projektdauer-minimierungsproblems mit planungsabhängigen Zeitfenstern. Shaker, Aachen

Schwindt C (1998c) A branch-and-bound algorithm for the resource-constrained project duration problem subject to temporal constraints. Technical Report WIOR-544, Institute for Economic Theory and Operations Research, University of Karlsruhe

Schwindt C, Trautmann N (2000) Batch scheduling in process industries: An application of resource-constrained project scheduling. OR Spektrum 22:501–524

Zhan, J (1994) Heuristics for scheduling resource-constrained projects in MPM networks. European Journal of Operational Research 76:192–205