Technische Universität Berlin

FACHBEREICH 3
MATHEMATIK

RESOURCE-CONSTRAINED PROJECT SCHEDULING WITH TIME WINDOWS: A BRANCHING SCHEME BASED ON DYNAMIC RELEASE DATES

by

ANDREAS FEST ROLF H. MÖHRING FREDERIK STORK
MARC UETZ

No. 596/1998(revised1999)

Resource-Constrained Project Scheduling with Time Windows: A Branching Scheme Based on Dynamic Release Dates*

Andreas Fest, Rolf H. Möhring, Frederik Stork, and Marc Uetz

Technische Universität Berlin, Fachbereich Mathematik,
Sekr. MA 6–1, Straße des 17. Juni 136, D-10623 Berlin, Germany
{fest,moehring,stork,uetz}@math.tu-berlin.de

Abstract We propose a branch-and-bound algorithm for resource-constrained project scheduling where any two of jobs can be linked by arbitrary minimal and maximal time lags. The jobs have to be scheduled non-preemptively, and while in process, they require several limited resources. The objective is to find a feasible schedule which minimizes the project makespan. Different branch-and-bound algorithms have been previously proposed – either based on constraint propagation techniques, or based on the idea to branch over so-called resource conflicts which are resolved by introducing additional precedence constraints. Our approach also follows the latter principle. The new idea is to resolve resource conflicts only locally by a dynamic update of job release dates instead of introducing precedence constraints. This gives rise to a reduction of both computation time and memory requirements in every node of the enumeration tree, however, at the expense of a loss of information. Nevertheless, enriched by preprocessing, strong dominance rules, and a flexible search strategy, our computational results show that the algorithm performs better than previous branch-and-bound algorithms, and is competitive with a very recent constraint propagation approach as well as tailor-made heuristics, also for large-scale instances.

1 Introduction

Resource-constrained scheduling problems occur in many real-world applications such as civil engineering, supply chain planning, production planning, and many others. Given is a set of *activities* or *jobs* which have to be scheduled in order to minimize some objective function. The jobs are typically subject to both temporal and resource constraints. Temporal constraints are given by minimal and possibly also maximal time lags between the start times of certain jobs. This allows to model several peculiarities such as time-varying resource availabilities or requirements, so-called time windows for the processing times of jobs, as well as job synchronizing or minimal job overlaps. While in process, every job requires a certain amount of renewable resources (e.g., machines and/or personnel), but the availability of these resources is limited. The objective

* Research supported by the Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (bmb+f), grant No. 03-MO7TU1-3. Correspondence to: Marc Uetz.

most frequently addressed is to find a time- and resource-feasible schedule minimizing the *project makespan*, which is the time required to complete all jobs. We refer to Section 2 for a detailed account of the model and notation used throughout the paper. Following the classification scheme proposed by Brucker, Drexl, Möhring, Neumann, and Pesch (1999), the model under consideration is termed $PS|temp|C_{\max}$.

Several well known combinatorial optimization problems can be (polynomially) transformed into resource-constrained project scheduling problems, particularly machine and shop scheduling problems, but also vertex coloring of graphs (Schäffter 1997). As a consequence, the problem under consideration is not only NP-hard, but it is even hard to approximate, and already the feasibility problem for the problem $PS|temp|C_{\max}$ is NP-hard in the strong sense (a reduction of CLIQUE can be found in (Bartusch, Möhring, and Radermacher 1988)).

Nevertheless, perhaps due to the practical relevance, exact and heuristic algorithms for resource-constrained project scheduling are recently receiving a growing attention, particularly for the problem setting under consideration. This is reflected by several branch-and-bound algorithms which have been proposed in the last years, e.g., by De Reyck and Herroelen (1998) (see also (De Reyck, Herroelen, and Demeulemeester 1998)), Schwindt (1997, 1998), and Dorndorf, Pesch, and Phan Huy (1998). Other approaches include tailor-made heuristics, some of which have been summarized in a paper by Neumann and Zimmermann (1998). For a restricted model without maximal time lags, but with time-varying resource requirements of the jobs, LP-based heuristics as well as local search algorithms have been proposed by Cavalcante, De Souza, Savelsbergh, Wang, and Wolsey (1998). In fact, this restricted model originates in a chemical production process at BASF AG, Germany (Kallrath and Wilson 1997). For the same model, Heipcke and Colombani (1997, 1998) have introduced a constraint propagation algorithm, and a heuristic based on approximation algorithms for single-machine scheduling problems has been proposed by Savelsbergh, Uma, and Wein (1998).

In this paper, we focus on the general problem $PS|temp|C_{\max}$ involving arbitrary minimal and maximal time lags. For this problem, important order theoretic insights into the structure of optimum solutions have been obtained by Bartusch et al. (1988). They also proposed a branch-and-bound procedure, however, their implementation is no longer available. Partially based on ideas of their work, branch-and-bound algorithms have been evaluated more recently by De Reyck and Herroelen (1998) and Schwindt (1997, 1998). The underlying idea of these algorithms is that *time-feasible schedules* (i.e., no temporal constraint is violated) are enumerated by systematically resolving *resource conflicts* (i.e., times where more than the available resources are required), and we call these approaches *conflict-based*. The resource conflicts are resolved by introducing additional precedence relations between jobs, or sets of jobs, a concept which is based upon an order theoretic representation theorem of optimal schedules (Bartusch et al. 1988, Theorem 3.8).

The algorithm we propose is also in the tradition of conflict-based branch-and-bound procedures. The new concept compared to all previous algorithms of this type is that resource conflicts are resolved by a dynamic update of release dates instead of introducing precedence relations. Thus, our algorithm is not based on the order theoretic concept of Bartusch et al. (1988), but on a very simple dominance property instead (see Lemma 1 and Theorem 2 below). At a first glance, this technique has the drawback that resource conflicts are resolved only locally. More precisely, it is possible that, due to the existence of maximal time lags, identical resource conflicts have to be resolved repeatedly in distinct nodes of the enumeration tree although they are located on the same path from the root to a leaf. Nevertheless, subject to several ingredients which help to truncate large parts of the enumeration tree, our computational results show that the algorithm performs better than previous conflict-based algorithms on the well established ProGen/max test sets (Schwindt 1996), and it is competitive with a very recent constraint propagation approach by Dorndorf et al. (1998). Furthermore, a truncated version of our branch-and-bound algorithm shows to compete to the best known heuristics also for large scale instances with up to 500 jobs. Compared to previous conflict-based branch-and-bound approaches, the benefit is partly due to a very efficient update of time-feasible schedules once a resource conflict has been resolved – a result of the simple branching rule we propose. All conflict-based approaches must solve this subproblem in every node of the enumeration tree. Within our branching scheme, this update requires $O(n^2)$ time, where n is the number of jobs. Previous approaches require at least $O(n^3)$ at this point.

The organization of the paper is as follows. Section 2 introduces the model and notation. Section 3 deals with the general idea of our branching scheme and its correctness, and Section 4 gives a more detailed account of some ingredients that helped to speed up the computations. Our computational results are presented in Section 5, and we conclude with some remarks in Section 6.

2 Model and Notation

Let $V = \{0,\dots,n+1\}$ be the set of jobs j with corresponding integral processing times p_j, including dummy jobs 0 and $n+1$ for project start and end, respectively ($p_0 = p_{n+1} = 0$). We assume that all jobs must be scheduled non-preemptively. By $S = (S_0,\dots,S_{n+1})$ we denote a schedule, where S_j is the start time of job j. Let $D = (d_{ij})$, $i, j \in V$ be the matrix of temporal constraints. That is, a *time-feasible* schedule S has to

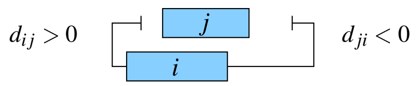

Figure1. Time window of job j relative to i

fulfill $S_j \geq S_i + d_{ij}$ for all $i, j \in V$. As depicted in Figure 1, $d_{ij} \geq 0$ ($d_{ji} < 0$) implies a minimal (maximal) positive time lag of S_j relative to S_i. If we assume without loss of generality that $S_0 = 0$, *release dates* and *due dates* for any job j are given by d_{0j} and $-d_{j0}$, respectively. Note that all temporal constraints refer to the start times S_j of jobs. The digraph $G(D) = (V,A)$ with node set V, arc set $A = \{(i,j) \mid d_{ij} > -\infty\}$ and arc weights d_{ij} for all $(i,j) \in A$ is called the *digraph of temporal constraints*.

It is well known that a time-feasible schedule exists, if and only if the digraph of temporal constraints does not contain a cycle of positive length (Bartusch et al. 1988).

By applying the Floyd-Warshall algorithm we calculate the longest paths between all pairs (i,j) in G, or detect cycles of positive length. Otherwise, throughout the paper we assume that $D=(d_{ij})$ is the *transitive closure* of the temporal constraints, more precisely, d_{ij} denotes the length of a longest path from i to j in G. The matrix D is also referred to as the *distance matrix*. A point-wise minimal time-feasible schedule is then given by $S_j := d_{0j}$, $j \in V$, and usually termed the *earliest start schedule* or $ES(D)$.

In addition to the temporal constraints, a finite set $\mathcal{R}$ of different, renewable resources is needed to complete the project. The availability of resource $k \in \mathcal{R}$ is given by a piecewise constant function $R_k(t)$, i.e., an amount of $R_k(t)$ of resource k is available at time t. Throughout the paper we identify t with the time period $[t, t+1)$. Every job j requires an amount of $r_{jk}(t)$ of resource k, $k \in \mathcal{R}$, during the tth time unit it is in process. However, for convenience we assume that $r_{jk}(t)$ as well as $R_k(t)$ are constant over time. This does not constitute any loss of generality, since jobs with a piecewise constant resource requirement can easily be replaced by several jobs which are "tied together" by means of temporal constraints, and a time-varying resource availability can be modeled by introducing dummy jobs with fixed start times which consume surplus resources.

For a given schedule S let $\mathcal{A}(S,t) := \{j \in V \mid S_j \leq t < S_j + p_j\}$ denote the set of jobs that is in progress at time t. If $\sum_{j\in\mathcal{A}(S,t)} r_{jk} > R_k$ for some $k \in \mathcal{R}$, we say that $\mathcal{A}(S,t)$ is a *forbidden set* (since it consumes more resources than available), and S causes a *resource conflict* at time t. A schedule is called *resource-feasible* if it does not cause any resource conflict for all k and t. It is called *feasible* if it is both resource and time-feasible. Assume that a schedule S causes a resource conflict at some time t, then a *minimal delaying alternative* $\mathcal{M} \to \mathcal{N}$ is a partition of $\mathcal{A}(S,t)$ into disjoint sets $\mathcal{M}$ and $\mathcal{N}$ such that $\sum_{j\in\mathcal{M}} r_{jk} \leq R_k$ for all $k \in \mathcal{R}$, and $\mathcal{M} \cup \{j\}$ is forbidden for all $j \in \mathcal{N}$. The intuition behind this notation is that delaying of all jobs $j \in \mathcal{N}$ will (at least temporarily) resolve the conflict at time t, and $\mathcal{N}$ is inclusion-minimal with this property.

If we let $r_k(S,t) := \sum_{j\in\mathcal{A}(S,t)} r_{jk}$ denote the resource consumption of resource k for a given schedule S at time t, the problem can be stated as follows:

$$
\begin{array}{lll}
\min & S_{n+1} & \\
\text{s.t.} & \left.\begin{array}{ll} S_j \geq S_i + d_{ij} & (i,j) \in A \\ S_0 = 0 & \end{array}\right\} & (1) \\
& r_k(S,t) \leq R_k \qquad k \in \mathcal{R},\ t = 0,1,\ldots,T, & (2)
\end{array}
$$

where T is an upper bound on the minimum project duration, and $D=(d_{ij})$ denotes the transitive closure of the temporal constraints. Here, (1) and (2) represent the temporal and the resource constraints, respectively.

3 The solution procedure

As already mentioned in the introduction, two different branch-and-bound approaches have recently received attention for the problem under consideration.

Within the *conflict-based* approach, time-feasible schedules are enumerated by resolving resource conflicts, which is done by introducing additional temporal constraints.

Every node of the enumeration tree represents a time-feasible, but possibly resource-infeasible schedule. Branching reflects all possibilities of resolving a conflict (i.e., all minimal delaying alternatives), and after a resource conflict has been resolved by having introduced new temporal constraints, a correspondingly updated time-feasible schedule has to be calculated. A feasible solution is found as soon as a schedule does not cause any resource conflict. Conflict-based approaches have previously been proposed by Bartusch et al. (1988), De Reyck and Herroelen (1998) as well as Schwindt (1997, 1998). While the first two approaches are based on introducing *precedence constraints* between jobs in order to resolve resource conflicts, Schwindt (1997, 1998) uses *disjunctive* precedence constraints between sets of jobs (that is, according to a minimal delaying alternative, some of the jobs of a resource conflict have to wait until at least one job completes). Since we also follow the conflict-based idea, we display the general framework in Algorithm 1.

Algorithm 1: Conflict-based branch-and-bound scheme

```
Input  : distance matrix D, resource constraints, upper bound T
Output : Schedule S* with minimal makespan C_max or "infeasible"

C_max ← T + 1; S*_{n+1} ← ∞;
perform preprocessing on D;                                   (Section 4.1)
if preprocessing reveals that D is infeasible then
    return "infeasible";
ActiveNodes ← {ES(D)};
while ActiveNodes ≠ ∅ do
    choose Node out of ActiveNodes;                           (Section 3.5)
    ActiveNodes ← ActiveNodes – Node;
    S ← time-feasible schedule corresponding to Node;
    if S is resource-feasible then
        if S_{n+1} < S*_{n+1} then S* ← S; C_max ← S_{n+1};
    else
        if Node is not dominated (Section 4.2) then
            A(S,t) ← first resource conflict of S;
            ActiveNodes ← ActiceNodes ∪ branch(S, A(S,t));    (Section 3.1)
if C_max = T + 1 then return "infeasible";
return S* and C_max;
```

The second approach is based on *constraint propagation* techniques, where every node of the enumeration tree represents feasible domains for the start times of jobs. Time and resource constraints are propagated by applying so called *consistency tests*, aiming at the reduction of those domains. Branching then represents some systematic way of partitioning the domains, and a feasible schedule is found as soon as all domains have cardinality one. We refer to Heipcke and Colombani (1997, 1998) and Dorndorf et al. (1998) for more details on constraint propagation approaches. We note, however, that the main difference between both approaches is basically another way of reducing the domains of start time variables.

3.1 Branching by dynamic release dates

Within the procedure we propose, every node of the enumeration tree represents a corresponding time-feasible schedule S. Unless this schedule is resource-feasible, it will cause at least one resource conflict. If we let t be the time of the earliest resource

Algorithm 2: Branching by dynamic release dates

```
Procedure: branch(S, A(S,t))
Input     : A time-feasible schedule S with resource conflict A(S,t), value of current
            best solution C_max
Output    : A set of new nodes (or time-feasible schedules, respectively)

Children = ∅;
if conflict A(S,t) has been resolved earlier on the same path then
    resolve conflict A(S,t) as before;                                  (Section 4.2)
    compute corresponding schedule S^new according to (4);              (Section 3.2)
    Children ← {S^new};
    return Children;
compute all min. delaying alternatives M → N for A(S,t);                (Section 3.6)
for all M → N do
    update d_0j for all j ∈ N according to (3), resp. (7);              (Sections 3.2, 3.4)
    if M → N is feasible (cf. Section 3.3) then
        compute schedule S^new according to (4);                        (Section 3.2)
        if S^new_{n+1} < C_max then
            perform consistency test and obtain
            a possibly improved lower bound LB;                         (Section 4.1)
            if LB < C_max then Children ← Children ∪ {S^new};
return Children;
```

conflict of S, then branching represents all possibilities of resolving this conflict by delaying the start times of certain jobs according to all minimal delaying alternatives for that conflict. The delaying itself is performed by augmenting the corresponding release dates and thus implicitly reducing the domains of the corresponding start-time variables. Based on the new set of release dates, it has to be checked if the resulting system of temporal constraints is feasible, and in this case a corresponding point-wise minimal time-feasible schedule S^{new} is calculated (see Section 3.2). Subject to an additional consistency test (see Section 4.1), the new node is eventually inserted into the enumeration tree if its lower bound does not exceed the current best known upper bound. Note that the new time-feasible schedule S^{new} is basically the only information which is stored in the corresponding node of the enumeration tree. The branching procedure is displayed in Algorithm 2.

3.2 Calculation of time-feasible schedules

Consider a node corresponding to a time-feasible schedule S, and assume that $\mathcal{A}(S,t)$ is forbidden for some time t. Then, as depicted in Figure 2, a child node is generated for every minimal delaying alternative $\mathcal{M} \to \mathcal{N}$ for $\mathcal{A}(S,t)$ by updating the release dates d_{0j} according to

$$d_{0j}^{new} := \min_{i \in \mathcal{M}} \{S_i + p_i\} \quad \text{for all } j \in \mathcal{N}. \tag{3}$$

This will (temporarily) resolve the resource conflict at time t and either results in time-infeasibility or a new minimal time-feasible schedule S^{new} can be calculated.

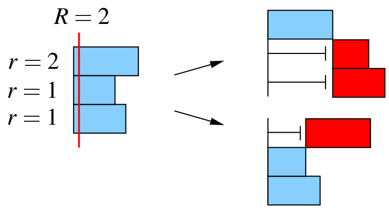

Figure2. Resolving a resource conflict via release dates according to all minimal delaying alternatives.

Note that, except for the values d_{0j}, $j \in \mathcal{N}$, the path lengths d_{ij} $(i \neq 0)$ remain constant in the course of the algorithm. We thereby allow that $d_{i0} + d_{0j}^{new} > d_{ij}$, i.e., some of the triangle inequalities of D may be violated. However, since we only increase the values d_{0j}, $j \in V$ the triangle inequalities $d_{ij} + d_{jk} \leq d_{ik}$ hold for all $i, j \neq 0$. One major advantage of our branching scheme is based on this invariant, which is expressed by the following theorem.

Theorem 1. *Consider a time-feasible but resource-infeasible schedule S, and a delaying alternative $\mathcal{M} \to \mathcal{N}$ to resolve a resource conflict $\mathcal{A}(S,t)$ of S. If the conflict is resolved by introducing release dates according to* (3), *the new set of temporal constraints is feasible if and only if $d_{0j}^{new} + d_{j0} \leq 0$ for all $j \in \mathcal{N}$, and a corresponding point-wise minimal time-feasible schedule can be computed in time $\Theta(|\mathcal{N}|n)$ by*

$$S_k^{new} = \max\{S_k, \max_{j \in \mathcal{N}} \{d_{0j}^{new} + d_{jk}\}\} \quad \textit{for all } k \in V. \tag{4}$$

Proof. Since the start times S^{new} given by (4) are obviously point-wise minimal, we only have to show that they are time-feasible if $d_{0j}^{new} + d_{j0} \leq 0$ for all $j \in \mathcal{N}$. Suppose this is not the case and there exists a pair $k, \ell \in V$ of jobs that violates the temporal constraints, i.e.

$$S_\ell^{new} < S_k^{new} + d_{k\ell}.$$

Let us first assume that $k \neq 0 \neq \ell$. Since S was time-feasible, and since $S_\ell \leq S_\ell^{new}$, we know that $S_k < S_k^{new}$ and thus there exists some delayed job $0 \neq j \in \mathcal{N}$ such that $S_k^{new} = d_{0j}^{new} + d_{jk}$. But now the triangle inequality yields

$$S_\ell^{new} < S_k^{new} + d_{k\ell} = d_{0j}^{new} + d_{jk} + d_{k\ell} \leq d_{0j}^{new} + d_{j\ell},$$

which is a contradiction to (4). Next consider the cases $k = 0$ and $\ell = 0$. If we assume that $S_\ell^{new} < d_{0\ell}^{(new)}$ for some $\ell \in V \setminus \{0\}$ (case $k = 0$), we again obtain a contradiction to (4). Furthermore, if $0 < S_k^{new} + d_{k0}$ for some $k \in V \setminus \{0\}$ (case $\ell = 0$), the same arguments as above yield a contradiction to the assumption that $d_{0j}^{new} + d_{j0} \leq 0$ for all $j \in \mathcal{N}$. □

Note that this is a major advantage over all previously proposed conflict-based branch-and-bound algorithms. Since for every new branch at most n jobs are delayed, the computation of the corresponding minimal time-feasible schedule S^{new} can be performed in $O(n^2)$ time. Within the procedures proposed by Bartusch et al. (1988) and De Reyck and Herroelen (1998), where precedence constraints are introduced instead of release dates, the analogous computation requires $O(n^3)$ time. This is due to the fact that every update of the distance matrix requires $O(n^2)$ time. Within the algorithm devised by Schwindt (1997, 1998) which is based on the idea of disjunctive precedence constraints, an even non-polynomial algorithm is proposed for the analogous problem (the running time depends on the current upper bound on the project makespan).

With respect to memory consumption, it follows directly from (3) and (4) that the information which has to be stored in every node of the enumeration tree is basically the vector of start times S^{new}. This is due to the fact that the values d_{ij} $(i \neq 0)$ remain invariant in the course of the algorithm. Thus the memory requirement for every node of the enumeration tree is drastically reduced in comparison to the procedures which use precedence constraints, where the entire distance matrix has to be stored.

3.3 Correctness of the branching scheme

Next we will show that our branching scheme in fact computes an optimal solution. Let some time-feasible schedule S be given, and let $\mathcal{A}(S,t)$ be a resource conflict of S. In accordance with Theorem 1, we call a delaying alternative $\mathcal{M} \to \mathcal{N}$ for $\mathcal{A}(S,t)$ *feasible* if it results in a time-feasible schedule, i.e., if $d_{0j}^{new} + d_{j0} \leq 0$ for all $j \in \mathcal{N}$. Furthermore, we introduce the notion of *domination*. We say that a time-feasible schedule S dominates another time-feasible schedule S', if $S_j \leq S'_j$ for all $j \in V$.

Lemma 1. *Consider a feasible schedule S^* and some time-feasible but resource-infeasible schedule S that dominates S^*. Then for any resource conflict $\mathcal{A}(S,t)$ of S there exists a feasible minimal delaying alternative $\mathcal{M} \to \mathcal{N}$ such that the resulting schedule S^{new} computed according to* (3) *and* (4) *dominates S^*.*

Proof. Let without loss of generality $\mathcal{A}(S,t) = \{1,\ldots,\ell\}$ be a forbidden set of jobs scheduled in parallel by schedule S at some time t, and assume that the order of start times of those jobs in schedule S^* is given by $S_1^* \leq \cdots \leq S_\ell^*$. Since S^* is resource-feasible and $\{1,\ldots,\ell\}$ is forbidden, we have

$$\min_{i=1,\ldots,k}\{S_i^* + p_i\} \leq S_{k+1}^*$$

for some $1 \leq k < \ell$, and we let k be minimal with this property. Then jobs $1,\ldots,k$ are scheduled simultaneously in S^* for some time, and consequently $\{1,\ldots,k\}$ is not forbidden.

We claim that any minimal delaying alternative $\mathcal{M} \to \mathcal{N}$ for conflict $\mathcal{A}(S,t)$ where $\{1,\dots,k\} \subseteq \mathcal{M}$ is feasible and (by applying (4)) leads to a time-feasible schedule S^{new} that dominates S^*. Let $\mathcal{M} \to \mathcal{N}$ be such a minimal delaying alternative, and suppose it were not feasible. Then, due to the definition of feasible minimal delaying alternatives, $d_{0j}^{new} + d_{j0} > 0$ for some $j \in \mathcal{N}$. But

$$d_{0j}^{new} = \min_{i\in\mathcal{M}}\{S_i + p_i\} \le \min_{i=1,\dots,k}\{S_i + p_i\} \le \min_{i=1,\dots,k}\{S_i^* + p_i\} \le S_{k+1}^*. \tag{5}$$

Since $j \in \mathcal{N}$ we have $S_{k+1}^* \le S_j^*$ due to the minimal choice of k. Due to (5) we now obtain $d_{0j}^{new} \le S_j^*$, but since we assumed that $d_{0j}^{new} + d_{j0} > 0$, this would imply $S_j^* + d_{j0} > 0$, which is a contradiction to S^* being feasible.

Thus $d_{0j}^{new} + d_{j0} \le 0$ for all $j \in \mathcal{N}$, and in particular $d_{0j}^{new} \le S_j^*$ for all $j \in \mathcal{N}$. We finally have to show that $S_j^{new} \le S_j^*$ for all $j \in V$. To this end, recall (4) and observe that for any start time S_k that has been changed we know that $S_k^{new} = d_{0j}^{new} + d_{jk}$ for some $j \in \mathcal{N}$. But now, since S^* is time-feasible, and since $d_{0j}^{new} \le S_j^*$ we obtain $S_k^* \ge S_j^* + d_{jk} \ge d_{0j}^{new} + d_{jk} = S_k^{new}$, which concludes the proof. □

Now observe that in every level of the enumeration tree at least one release date is increased by at least 1. Under the assumption that we are given an upper bound T on the minimal project duration, we have proven the following theorem.

Theorem 2. *The above branch-and-bound procedure based on a branching scheme according to* (3) *and* (4) *computes an optimal solution in a finite number of steps.*

3.4 Revised update of release dates

The above update step (3) can be further refined by anticipating temporal constraints between the sets $\mathcal{M}$ and $\mathcal{N}$ of a delaying alternative as follows. Consider a time-feasible schedule S where some forbidden set $\mathcal{A}(S,t)$ is processed in parallel at some time t. Let $\mathcal{M} \to \mathcal{N}$ be a delaying alternative for this conflict and suppose there exist jobs $i \in \mathcal{M}$ and $j \in \mathcal{N}$ such that $p_i > -d_{ji}$. Then job j has to start before completion of i in every time-feasible schedule S, i.e., $S_j < S_i + p_i$. In this case the completion time of i need not be considered when calculating the delay for the jobs $j \in \mathcal{N}$. More precisely, we let $\mathcal{M}' := \{i \in \mathcal{M} \mid p_i \le -d_{ji} \text{ for all } j \in \mathcal{N}\}$ and replace the update of the release dates as of (3) by

$$d_{0j}^{new} := \min_{i\in\mathcal{M}'}\{S_i + p_i\} \quad \text{for all } j \in \mathcal{N}. \tag{6}$$

Now we claim that this suffices for the correctness of the branching scheme.

Corollary 1. *The above branch-and-bound procedure based on a branching scheme according to* (6) *and* (4) *computes an optimal solution in a finite number of steps.*

Proof. Recall the proof of Lemma 1 and simply replace (5) by

$$\begin{aligned} d_{0j}^{new} &= \min_{i\in\mathcal{M}'}\{S_i + p_i\} \\ &\le \min_{i\in\{1,\dots,k\}\cap\mathcal{M}'}\{S_i + p_i\} \le \min_{i\in\{1,\dots,k\}\cap\mathcal{M}'}\{S_i^* + p_i\} \le S_{k+1}^*. \end{aligned}$$

To see that the last inequality holds, recall that $\min_{i=1,\ldots,k}\{S_i^* + p_i\} \leq S_{k+1}^*$ and observe that otherwise there exists some $i \in \{1,\ldots,k\} \setminus \mathcal{M}'$ with $S_i^* + p_i \leq S_{k+1}^*$. By definition of $\mathcal{M}'$ it follows that $p_i + d_{ji} > 0$ for some $j \in \mathcal{N}$, and thus, since S^* is supposed to be feasible, $S_j^* < S_i^* + p_i \leq S_{k+1}^*$. But due to the choice of k, and since $j \in \mathcal{N}$ we also have $S_{k+1}^* \leq S_j^*$, a contradiction. Thus, as in the proof of Lemma 1, we obtain $d_{0j}^{new} \leq S_j^*$ for all $j \in \mathcal{N}$, and proceed as before. □

Another improvement can be obtained by taking so-called *feasible domains* for the start times of jobs into consideration (see, e.g., (Dorndorf et al. 1998) and (Klein and Scholl 1999)). A feasible domain Δ_j for the start time of a job j can be calculated as follows. For a given upper bound T on the minimal project makespan, determine the earliest and latest start and completion times ES_j, LS_j, EC_j, and LC_j for every job j and set $\Delta_j := \{ES_j,\ldots,LS_j\}$. If for a job j the latest start time LS_j is smaller then the earliest completion time EC_j, the job must be in process during its so-called *core time* $CT_j := \{LS_j,\ldots,EC_j\}$. Now, for every job j and every time $t \in \{ES_j,\ldots,LC_j\}$, check if the job has a resource conflict with the jobs I_t, where $I_t := \{i \in V \mid t \in CT_i\}$ is the set of jobs which *must* be in process at time t. If this is the case, job j must not be executed at time t, and one can remove the start times $\{t - p_j + 1,\ldots,t\}$ from Δ_j. Finally, within our branching scheme we can obviously replace (6) by

$$d_{0j}^{new} := \min_{t \in \Delta_j}\{t \geq \min_{i \in \mathcal{M}'}\{S_i + p_i\}\} \quad \text{for all } j \in \mathcal{N}. \tag{7}$$

3.5 The search strategy

The search strategies we examined are based on the idea to use auxiliary functions to decide which node could be a promising candidate for branching. Such auxiliary functions are particularly important if the objective is the project makespan, since any criterion which is only based on the makespan of the actual schedule, that is, the critical path lower bound, is not sensitive enough to local schedule modifications, and thus in general does not serve as a good decision criterion. More precisely, the decisions are based on two parameters which are calculated for the corresponding time-feasible schedules S. On the one hand, these are the *gaps* between the start times S_j and latest start times $-d_{j0}$, and on the other hand the *tails* d_{jn+1}. To improve flexibility, we additionally use of a certain degree of randomness while choosing the nodes to be processed next.

Finally, we implemented a very flexible tree traversing strategy that processes a parameter driven number of DFS-like paths at a time. (When used as a heuristic, this strategy is also termed *beam search*.) A comparable tree traversing strategy has been used, e.g., by Klein and Scholl (1998).

3.6 Calculation of minimal delaying alternatives

Minimal delaying alternatives have to be computed in (almost) every node of the enumeration tree. For a time-feasible schedule S with a resource conflict at time t, in principle all subsets $\mathcal{M} \subset \mathcal{A}(S,t)$ are potential candidates for non-dominated and non-forbidden sets, yielding a minimal delaying alternative $\mathcal{M} \to \mathcal{A}(S,t) \setminus \mathcal{M}$. A procedure for computing these sets appears in Brucker, Knust, Schoo, and Thiele (1998). They propose a binary decision tree where every level j of the tree corresponds to the decision to include or exclude job j.

The procedure we implemented is basically the same, however, the necessary effort can be slightly reduced by considering the jobs in non-decreasing order of their consumption of some resource, preferably the resource with smallest ratio $R_k/r_k(S,t)$. Moreover, as also proposed, e.g., in (De Reyck and Herroelen 1998, Theorem 2), the given temporal constraints may serve to discard certain minimal delaying alternatives immediately within this subroutine. Consider, for instance, a minimal delaying alternative $\mathcal{M} \to \mathcal{N}$ and suppose there exist two jobs $j \in \mathcal{M}$ and $i \in \mathcal{N}$ such that $d_{ij} \geq 0$. Then it is easy to see that this delaying alternative is redundant and can be discarded.

3.7 Upper and lower bounds

An trivial upper bound is easily obtained by calculating $\sum_{j\in V} \max\{p_j, \max_{i\in V} d_{ji}\}$. To improve the initial upper bound, we additionally perform a depth first search heuristic before starting the branch-and-bound algorithm. A couple of paths in the enumeration tree are generated by resolving resource conflicts according to the orders given by standard priority lists. In most of the cases, the initial upper bound could be significantly improved by this heuristic, while the computation time is negligible.

One of the main ingredients of a branch-and-bound procedure is the computation of lower bounds. Several lower bounds have been proposed, compared, and evaluated for the basic model for resource-constrained project scheduling $PS|prec|C_{\max}$, where jobs are subject to ordinary precedence relations only. We refer to Stinson, Davis, and Khumawala (1978), Christofides, Alvarez-Valdes, and Tamarit (1987), Mingozzi, Maniezzo, Ricciardelli, and Bianco (1998), Klein and Scholl (1999) and Brucker and Kunst (1998). These bounds have been partially extended to the model with arbitrary time lags, e.g., by De Reyck and Herroelen (1998) and Heilmann and Schwindt (1997). Moreover, a Lagrangian approach which is based upon minimum–cut computations in auxiliary networks has been proposed by Möhring, Schulz, Stork, and Uetz (1999).

One of the fastest computable lower bounds is based on a single-machine relaxation (several authors, e.g., Mingozzi et al. (1998) refer to this bound as LB_3). The idea is to determine a set of jobs out of which no pair can be scheduled simultaneously, either due to temporal or resource constraints. Clearly, those jobs must be scheduled sequentially, and the sum of their processing times is a lower bound on the minimal project makespan. This bound can be improved by considering also fractions of jobs; see (De Reyck and Herroelen 1998, Theorem 4). It can be further refined by also considering the heads and tails of the involved jobs as proposed by Carlier (1982, Proposition 1).

However, our procedure is based on the principle to spend very little effort in every single node of the enumeration tree, at the expense of a comparatively large enumeration tree. Thus only fast computable lower bounds were considered, and we obtained the best results by only using the critical path lower bound, which is simply the makespan of the resource relaxation of the (sub-)problem under consideration. Within our procedure, this lower bound is obviously obtained in time $O(n)$ in every node of the enumeration tree when calculating the point-wise minimal time-feasible schedule as proposed in

(4). Nevertheless, we have integrated the above mentioned single-machine based lower bound into a preprocessing routine (see also Section 4.1).

4 Improving the performance

In this section we present some further ingredients which turned out to be computationally fruitful.

4.1 Preprocessing

In a preprocessing step, certain additional constraints (such as precedence relations or release dates) are fixed beforehand by arguing that they must be fulfilled by any optimal schedule.

As a typical example consider the following consistency test which is also described in Brucker, Hilbig, and Hurink (1999) for single machine scheduling, and in Schwindt (1997) and De Reyck and Herroelen (1998) for resource-constrained project scheduling. If a pair of jobs $\{i, j\}$ is forbidden, that is, i and j cannot be scheduled simultaneously, and either $-p_j < d_{ij}$, or $T < d_{0j} + p_j + d_{i,n+1}$ (for some global upper bound T on the project makespan), then j cannot precede i, since this would violate the temporal constraints. Therefore, i must precede j in every feasible schedule. Even if none of the conditions $-p_j < d_{ij}$ and $T < d_{0j} + p_j + d_{in+1}$ is fulfilled, we know that either i must precede j or vice versa, and this consideration, by taking the point-wise minimum over the resulting release dates, possibly leads to larger release dates for some jobs. In a preprocessing step we perform this rule for every forbidden set of cardinality two, and we obtain a possibly extended distance matrix D which is still valid for all optimal schedules. Clearly, this procedure can be invoked repeatedly until no further improvement is achieved. Additionally, an analogous consistency test can be applied in every node of the enumeration tree (where we have restricted the procedure to only a single iteration).

Another preprocessing step we implemented is to compute an earliest start time for any job j not only by a longest path calculation (that is, d_{0j}), but by calculating the above mentioned single-machine based lower bound for every single job j by considering the set of all (fractions of) jobs that must complete before j. If a subset of those (fractions of) jobs is identified such that no pair may be scheduled simultaneously, the sum of their processing times is clearly a lower bound for the earliest start time of job j.

4.2 Dominance rules

Generally speaking, dominance rules allow to discard redundant nodes of the enumeration tree by arguing that these are dominated by others. As already mentioned, the approach by dynamic release dates is *oblivious* in the sense that the same resource conflicts possibly have to be resolved repeatedly. In a sense, the dominance rules we have implemented help to recover a certain portion of the "history" that led to a specific node of the enumeration tree, which allows to identify redundant nodes.

First, based upon Lemma 1, any node x_S of the enumeration tree (corresponding to a time-feasible schedule S) can be discarded if there exists another node $x_{S'}$ on a different path of the enumeration tree such that S' dominates S, i.e., if $S'_j \leq S_j$ for all $j \in V$. This obviously follows since any feasible schedule S^* that could be obtained starting from x_S is also dominated by S'. This dominance rule can obviously be implemented by following the path from node x_S up to the root of the enumeration tree, and considering all immediate children of these nodes. At this point however, we have to take care of a phenomenon which could be termed *cross pruning*: Suppose that nodes $x_{S'}$ and x_S lie on different paths of the enumeration tree and dominate all child nodes of the other, respectively. Obviously, pruning of all child nodes could possibly lead to incorrect results.

But we can do even better if we store in every node of the enumeration tree how a resource conflict was resolved to generate that node. Then, before branching over some resource conflict, we check if the same conflict, or a subset thereof, has been resolved before by following the path up to the root node. If this is the case we need not branch but it suffices to make the same decision as before. To make this clear, consider the situation depicted in Figure 3. The resource conflict caused by jobs $\{1,2,3\}$ has previously been resolved on the path up to the root by delaying jobs 2 and 3. Thus it is not necessary to branch again over all minimal delaying alternatives for this conflict, but it suffices to consider the decision that was made before, i.e., delaying jobs 2 and 3. This is sufficient, since the other nodes corresponding to the remaining minimal delaying alternatives would in any case be dominated (as indicated by the two arcs in Figure 3). To implement this dominance rule, each node of the tree additionally has to incorporate a pointer to its father in the enumeration tree, as well as the information about how a resource conflict has been resolved.

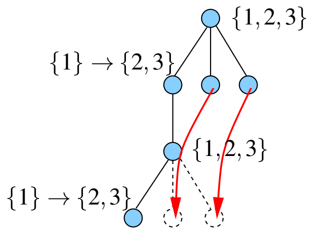

Figure3. A simple dominance rule by recalling past decisions

For both dominance rules, a certain amount of additional memory has to be reserved to store the required information, which particularly includes nodes where branching has already been performed. Our experiments showed, however, that on average the overhead was outweighed by the desired effect.

For the next dominance (or fathoming) rule, we need some preliminary definitions. For a minimal delaying alternative $\mathcal{M} \to \mathcal{N}$ of a resource conflict $\mathcal{A}(S,t)$, call a job $i \in \mathcal{M}$ *critical* (w.r.t. the current schedule S) if $i = \mathrm{argmin}_{\nu \in \mathcal{M}} \{S_\nu + p_\nu\}$, and let I be the set of all critical jobs with respect to S and $\mathcal{M} \to \mathcal{N}$. Denote by $\mathcal{M} \prec \mathcal{N}$ a corresponding *disjunctive* precedence constraint of the following form:

$$\text{There exists at least one } i \in I \text{ such that } S_i + p_i \leq S_j \text{ for all } j \in \mathcal{N}\ .$$

(If there is only a single critical job $i \in I$ for a delaying alternative $\mathcal{M} \to \mathcal{N}$, the corresponding disjunctive precedence constraint $\mathcal{M} \prec \mathcal{N}$ is simply a set of ordinary precedence constraints between job i and all $j \in \mathcal{N}$.) Now consider a path within our enumeration tree, and the corresponding series of delaying alternatives $\mathcal{M}_q \to \mathcal{N}_q$, $q = 1, \ldots, r$ with respective critical jobs I_q. Let us call such a path *non-contradicting*, if the corresponding set of disjunctive precedence constraints $\mathcal{M}_q \prec \mathcal{N}_q$, $q = 1, \ldots, r$, together with the temporal constraints given by the distance matrix D is feasible, i.e., the constraints do not induce any cycle of positive length. (Following this notation, the revised update step according to (6) and (4) guarantees that any single delaying alternative, respectively the corresponding update of release dates is not contradicting in itself.)

Lemma 2. *Within the proposed branching scheme, it suffices to consider paths which are non-contradicting.*

Proof. From the representation theorem for optimal schedules (Bartusch et al. 1988, Theorem 3.8), we know that it suffices to consider optimal schedules S^* which can be obtained as earliest-start schedule of a *feasible extension* of the temporal constraints. Here, a feasible extension of the temporal constraints is an extension of the given temporal constraints by a set of precedence constraints which does not induce any cycle of positive length. Now, exactly as in the proof of Lemma 1, one can show inductively that non-contradicting paths are sufficient to generate all schedules S^* which correspond to feasible extensions of the temporal constraints, thus it suffices to restrict to non-contradicting paths; we omit the details. □

It is generally not trivial to detect non-contradicting paths, however, we implemented a third dominance rule which is based upon the above lemma, and which makes use of so-called *total idle times*. A total idle time is defined as a time interval where no job is in process, and there is no temporal constraint which enforces this constellation. More precisely, we say that there is a total idle time at time t, if there are two non-empty sets A and B such that any job $j \in A$ completes no later than t and any job $j \in B$ starts no earlier than $t+1$, and additionally $S_i + d_{ij} < S_j$ for all $i \in A$ and $j \in B$. The last condition states that, loosely speaking, there is no temporal constraint enforcing the idle time at $[t, t+1)$.

Lemma 3. *If a time feasible schedule S has a total idle time, the corresponding node of the enumeration tree can be discarded.*

Proof. We show that only a contradicting path leads to a total idle time, thus the claim follows by Lemma 2. Consider a node of the enumeration tree, let t be a total idle time in the corresponding schedule, and let A and B be the sets as defined above. Remember that $S_i + d_{ij} < S_j$ for all $i \in A$ and $j \in B$, and that the whole set B can be left-shifted by at least one time unit. Consequently, for any job $j \in B$, there exists at least one job $i \in B$ which caused that j was postponed to its current position (it is crucial here that $i \in B$). More precisely, the reason for j's current position is either a tight temporal constraint with a job $i \in B$, or a minimal delaying alternative $\mathcal{M} \to \mathcal{N}$ on the path from the root node down to the node under consideration such that $j \in \mathcal{N}$ (i.e., job j has been delayed). Note that in the latter case, the respective critical jobs $I \subseteq \mathcal{M}$ for the delaying alternative $\mathcal{M} \to \mathcal{N}$ are all contained in B. Thus, if one recursively continues this reasoning (i.e., for $j \in B$ pick (any of) the job(s) $i \in B$ which caused that j was moved to its current position), since B is finite, is not hard to see that one obtains a cycle of positive length among the jobs in B. This eventually shows that the disjunctive precedence constraints which correspond to the respective minimal delaying alternatives (from the root down to the node under consideration), together with the given temporal constraints, are infeasible. Consequently, the path under consideration is not non-contradicting. □

4.3 Bidirectional planning

A folklore trick for scheduling problems is to consider an equivalent instance where all temporal constraints have been reversed. For the makespan objective, the respective solutions can be transformed in either direction. Thus, when we put limits on the running time of our branch-and-bound procedure, we make use of this simple trick by spending half of the time for the original instance, and another half of the time for an instance with reversed temporal constraints, now using the makespan of the best found solution so far, if any, as an upper bound. In quite some cases, this helped to reduce the time required to verify optimality.

5 Computational Experiences

5.1 Computing Environment

Our experiments were conducted on a Sun Ultra 2 with 200 MHz clock pulse and 512 MB of memory operating under Solaris 2.6. The code has been written in C++ and is compiled with the EGCS g++ compiler version 2.8 using the -O3 optimization option.

5.2 Benchmark Instances

We have applied our algorithms to the widely accepted test beds of the ProGen/max library that has been created by Schwindt (1996). ProGen/max is an extension of the instance generator ProGen which has been developed by Kolisch and Sprecher (1996) and which is designed to create instances with ordinary precedence constraints only. We used a test set of 1080 instances each consisting of 100 jobs (Test set A), as well as a test set of 120 instances each consisting of 500 jobs (Test set B). In both test sets, the job processing times are chosen randomly between 5 and 15 and the number of different resources is 5. The instances are generated by systematically modifying four different control parameters, the *network complexity*, which reflects the average number of direct successors of an activity, the *resource factor*, which describes the average number of resources required in order to process an activity, the *resource strength*, which is a measure of the scarcity of the resources, and a parameter which controls the number of cycles in the digraph of temporal constraints. In order to generate the temporal constraints, ProGen/max makes use of two different methods; either an entire project network is created directly, or first (cyclic) subnetworks are generated and merged afterwards. For a detailed discussion of the characteristics of the instances we refer the reader to (Schwindt 1996, 1997). Moreover, we have also considered a small set of benchmark instances originating in a typical chemical production process as described in (Kallrath and Wilson 1997); the instances have been taken from (Cavalcante 1997). These instances do not involve maximal time lags, but time-varying resource requirements of jobs.

5.3 Experimental Results

Tables 1–3 show the results of some of the experiments we have performed. For each experiment, we have tested all instances of a test set subject to a limit on the running time for each individual instance. We then display the number of instances where a feasible solution was found, the number of instances where an optimal solution was found and optimality was verified, and the number of instances where infeasibility could be proved within the given time limit. The tables further display the number of instances where neither a feasible solution could be found nor infeasibility could be proved within the time limit. Finally, the tables contain the average deviation from the respective lower bounds, taken over all instances where a feasible solution was found. For comparison reasons, the lower bounds have been taken from an URL maintained by Schwindt (1999). Note that exactly 1059 out of the 1080 instances from Test set A, and 119 out of the 120 instances from Test set B have a feasible solution.

Improving the performance – Impact of the different ingredients. Table 1 refers to Test set A and compares the results of five of the experiments we have performed in order to measure the impact of the different ingredients proposed in Section 4. We display results for the case that all of those ingredients are switched on and off, respectively. Moreover, the table shows the results when *not* using (a) the dominance rules (Section 4.2), (b) the preprocessing (Section 4.1), and (c) the separation of the running time between original and a reversed instance (Section 4.3). For these experiments, we have allowed 100 seconds of computation time.

Obviously, all of the considered ingredients contribute considerably to the overall performance of the algorithm. In particular, as shown in Table 1, all ingredients improve the number of optimally solved instances, the dominance rules are crucial for finding feasible solutions for all of the 1059 feasible instances, and the preprocessing is responsible to prove infeasibility for all of the 21 infeasible instances (note that without preprocessing, 17 of these instances remain open). On average, the quality of the computed solutions is satisfactory (7% deviation of computed solution and the respective best known lower bound), however, there are a few instances with an extremely large deviation, such that the maximum deviation varies between 175–208% for these experiments.

Running times. We next investigate the computation times that are required to obtain solutions for Test set A. In 6 different experiments, we allowed 3, 10, 30, 100, 300, and 1000 seconds of computation time. All of the above mentioned ingredients are used within these experiments, except that for running times between 3 and 30 seconds, the preprocessing is restricted to one second (for some of the instances, the preprocessing takes up to 15 seconds). We note that the procedure computes a feasible solution for

	all ingred. on	all ingred. off	(a) domin. off	(b) preproc. off	(c) bidirect. off
feasible	1059	1019	1028	1059	1058
optimal	768	690	742	707	739
infeasible	21	2	21	4	21
unknown	0	59	31	17	1
av. dev. in %	7.0	9.1	8.3	7.5	7.6

Table1. Termination status of our algorithm for the 1080 instances of Test set A for 100 seconds of computation time. The first two columns show the results for the case when all ingredients described in Section 4 are switched on or off, respectively, and the next three columns show the impact of *not* using dominance rules, preprocessing, and bidirectional planning. Note that exactly 1059 instances are feasible.

all of the feasible 1059 instances already within 10 seconds, however, for very large computation times (1000 seconds), the improvements are marginal.

	3 sec.	10 sec.	30 sec.	100 sec.	300 sec.	1000 sec.
feasible	996	1059	1059	1059	1059	1059
optimal	628	720	749	768	781	792
infeasible	20	20	20	21	21	21
unknown	64	1	1	0	0	0
av. dev. in %	10.9	9.4	7.7	7.0	6.5	6.1

Table2. Termination status of our algorithm (all ingredients enabled) for the 1080 instances of Test set A for different time limits.

Beam–search. To cope with the large-scale instances of Test set B (500 jobs), we have implemented a truncated (beam–search) variant of our procedure. For each node in the enumeration tree we only consider k minimal delaying alternatives. Among them, we always take the best w, $w < k$, nodes into further consideration. This selection is performed according to the computation of *gaps* and *tails* as described in Section 3.5. Among several experiments, we obtained the best results when setting $k = 10$ and $w = 5$. Based on these parameters settings, we have performed four experiments allowing 50, 100, 200, and 1000 seconds of computation time, respectively. As indicated in Table 3, this truncated version of our algorithm computes a feasible solution for all 119 feasible instances within less than 200 seconds, and optimality is verified for 70 instances within less than 50 seconds (by computing a solution which matches the critical path lower bound). Within these experiments, the maximum deviation of the computed solution from the critical path lower bound varies between 33–56%.

Comparison to previous branch-and-bound algorithms and heuristics. We compare the results of our algorithm to results that have been obtained by other branch-and-bound algorithms, as well as to a cycle-decomposition (priority-rule) heuristic which

	50 sec.	100 sec.	200 sec.	1000 sec.
feasible	94	117	119	119
optimal	70	70	70	70
infeasible	1	1	1	1
unknown	25	3	0	0
av. dev. in %	1.1	5.0	5.2	3.8

Table3. Termination status of the truncated version of our algorithm for different time limits for the 120 instances of Test set B. Note that exactly 119 instances are feasible.

is described in Neumann and Zimmermann (1998). Following the latter authors, the cycle-decomposition heuristic is among the most powerful priority-rule heuristics for the problem. All authors have used the same test sets, however, we would like to stress that such comparisons are to be taken with care, due to the different hard- and software architectures. Schwindt (1998) as well as Neumann and Zimmermann (1998) have used a Pentium PC with 200 MHz clock pulse, De Reyck and Herroelen (1998) a Pentium PC with 60 MHz, and Dorndorf et al. (1998) performed their experiments on a Pentium/Pro PC with 200 MHz clock pulse and used ILOG C++ libraries. All algorithms are written in C or C++.

Consider, e.g., the percentage of instances of Test set A solved to optimality. According to Table 2, our algorithm verifies optimality for 70% of the 1080 instances within 30 seconds of computation time. Within the same time limit, Schwindt (1998) could verify optimality for 62.5% of the instances, while De Reyck and Herroelen (1998) verified 56% within a time limit of 100 seconds. Dorndorf et al. (1998) could recently optimize 70.1% of the instances, also within 30 seconds of computation time. All of these algorithms – except the one by De Reyck and Herroelen (1998) – find a feasible solution for all 1059 feasible instances and prove infeasibility for the remaining 21 instances. The corresponding overall average deviations from the best known lower bounds (with respect to all 1059 feasible instances) are 7.7% (this paper), 7.0% (Schwindt 1998), and 4.6% (Dorndorf et al. 1998). De Reyck and Herroelen (1998) report an average deviation of 14%, but this value is based on a different set of instances, and perhaps worse lower bounds.

For Test set B, Neumann and Zimmermann (1998) as well as Dorndorf et al. (1998) have reported on computational results. Within 200 seconds, the cycle-decomposition heuristic of Neumann and Zimmermann (1998) solves 7 instances optimally, and all 119 instances feasible, at an overall average deviation of 5% from the critical path lower bounds, whereas a truncated branch-and-bound procedure based on (Schwindt 1998) (the results are documented in (Neumann and Zimmermann 1998)) solves 74 instances optimally, and 95 feasible, at an average deviation of 0.1%. Within the same time limit, Dorndorf et al. (1998) solve 78 of the instances optimally, and find a feasible solution for 116 instances, at an average deviation of 0.5%. Our algorithm computes a feasible solution for all 119 instances (within 200 seconds), thereof 70 are solved optimally, and the overall average deviation for all 119 instances is 5.2%.

The instances from (Cavalcante 1997) have been considered by Cavalcante et al. (1998) as well as (Heipcke and Colombani 1997, 1998); one of these instances also by

Savelsbergh, Uma, and Wein (1998). Although we were able to find a feasible solution for all instances within short time with our algorithm, the quality of the first found solutions could often not be improved substantially, even for large running times. For these instances, the best known solutions have been obtained via tabu search by Cavalcante et al. (1998).

6 Concluding Remarks

We have proposed a simple yet powerful branch-and-bound algorithm for resource-constrained project scheduling problems where any two of jobs can be linked by an arbitrary time window. The theoretical foundation for our approach is a very simple dominance property of earliest-start schedules, which is exactly reflected by the branching scheme we propose. In spite of the simplicity of the general algorithmic idea and, as a result, the comparatively large enumeration tree, our computational experiences suggest that the algorithm performs better than previous conflict-based branch-and-bound algorithms. This is perhaps due to the very efficient update of time-feasible schedules in every node of the enumeration tree, and the effectiveness of strong and simple dominance rules. Moreover, our algorithm competes with a recent constraint propagation approach as well as tailor-made heuristics, and a truncated version shows to be suited also for large-scale instances with up to 500 jobs.

References

Bartusch, M., R. H. Möhring, and F. J. Radermacher (1988). Scheduling project networks with resource constraints and time windows. *Annals of Operations Research 16*, 201–240.

Brucker, P., A. Drexl, R. H. Möhring, K. Neumann, and E. Pesch (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research 112*(1), 3–41.

Brucker, P., T. Hilbig, and J. Hurink (1999). A branch & bound algorithm for a single machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics 94*, 77–99.

Brucker, P., S. Knust, A. Schoo, and O. Thiele (1998). A branch & bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research 107*(2), 272–288.

Brucker, P. and S. Kunst (1998). A linear programming and constraint propagation-based lower bound for the RCPSP. Technical Report 204, Department of Mathematics, University of Osnabrück. Revised 1999.

Carlier, J. (1982). The one–machine sequencing problem. *European Journal of Operational Research 11*, 42–47.

Cavalcante, C. C. B. (1997). `http://www.dcc.unicamp.br/~cris/SPLC.html`.

Cavalcante, C. C. B., C. C. De Souza, M. W. P. Savelsbergh, Y. Wang, and L. A. Wolsey (1998). Scheduling projects with labor constraints. CORE Discussion Paper 9859, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

Christofides, N., R. Alvarez-Valdes, and J. M. Tamarit (1987). Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research 29*, 262–273.

De Reyck, B. and W. Herroelen (1998). A branch and bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research 111*(1), 152–174.

De Reyck, B., W. Herroelen, and E. Demeulemeester (1998). Algorithms for scheduling projects with generalized precedence relations. In J. Węglarz (Ed.), *Handbook on Recent Advances in Project Scheduling*. Kluwer Academic Publishers.

Dorndorf, U., E. Pesch, and T. Phan Huy (1998). A time oriented branch-and-bound algorithm for resource-constrained project scheduling with generalized precedence constraints. Technical report, University of Bonn, Germany.

Heilmann, R. and C. Schwindt (1997). Lower bounds for RCPSP/max. Technical Report 511, WIOR, University of Karlsruhe, Germany.

Heipcke, S. and Y. Colombani (1997). A new constraint programming approach to large scale resource constrained scheduling. In *Third Workshop on Models and Algorithms for Planning and Scheduling Problems, Cambridge*.

Heipcke, S. and Y. Colombani (1998). A global constraint for scheduling under labour constraints. In *ECAI 98 Workshop on Non Binary Constraints*.

Kallrath, J. and J. M. Wilson (1997). *Business Optimisation using Mathematical Programming*. Macmillan Business, London, U.K.

Klein, R. and A. Scholl (1998). Scattered branch and bound – an adaptive search strategy applied to resource-constrained project scheduling. Technical Report 6/98, Schriften zur Quantitativen Betriebswirtschaftslehre, Darmstadt University of Technology, Germany.

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

Kolisch, R. and A. Sprecher (1996). PSPLIB - A project scheduling problem library. *European Journal of Operational Research 96*, 205–216.

Mingozzi, A., V. Maniezzo, S. Ricciardelli, and L. Bianco (1998). An exact algorithm for the multiple resource-constained project scheduling problem based on a new mathematical formulation. *Management Science 44*, 714–729.

Möhring, R. H., A. S. Schulz, F. Stork, and M. Uetz (1999). Resource-constrained project scheduling: Computing lower bounds by solving minimum cut problems. In J. Nešetřil (Ed.), *Algorithms – ESA'99*, Volume 1643 of *Lecture Notes in Computer Science*, pp. 139–150. Springer. Proceedings of the 7th Annual European Symposium on Algorithms, Prague.

Neumann, K. and J. Zimmermann (1998). Methods for resource-constrained project scheduling with regular and nonregular objective functions and schedule-dependent time windows. In J. Węglarz (Ed.), *Handbook on Recent Advances in Project Scheduling*. Kluwer Academic Publishers.

Savelsbergh, M. W. P., R. N. Uma, and J. Wein (1998). An experimental study of LP-based approximation algorithms for scheduling problems. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco*, pp. 453–462.

Schäffter, M. (1997). Scheduling with respect to forbidden sets. *Discrete Applied Mathematics 72*, 141–154.

Schwindt, C. (1996). Generation of resource constrained project scheduling problems with minimal and maximal time lags. Technical Report 489, WIOR, University of Karlsruhe, Germany.

Schwindt, C. (1997). *Verfahren zur Lösung des ressourcenbeschränkten Projektdauerminimierungsproblems mit planungsabhängigen Zeitfenstern*. Ph. D. thesis, WIOR, University of Karlsruhe, Germany.

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

Schwindt, C. (1999). `ftp://ftp.wior.uni-karlsruhe.de/pub/ProGen-max/pspMaxLib/RCPSPMax`.

Stinson, J. P., E. W. Davis, and B. M. Khumawala (1978). Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions 10*, 252–259.

Reports from the group

"Combinatorial Optimization and Graph Algorithms"

of the Department of Mathematics, TU Berlin

634/1999 *Karsten Weihe and Ulrik Brandes and Annegret Liebers and Matthias Müller–Hannemann and Dorothea Wagner and Thomas Willhalm:* Empirical Design of Geometric Algorithms

633/1999 *Matthias Müller–Hannemann and Karsten Weihe:* On the Discrete Core of Quadrilateral Mesh Refinement

632/1999 *Matthias Müller–Hannemann:* Shelling Hexahedral Complexes for Mesh Generation in CAD

631/1999 *Matthias Müller–Hannemann and Alexander Schwartz:* Implementing Weighted b-Matching Algorithms: Insights from a Computational Study

629/1999 *Martin Skutella:* Convex Quadratic Programming Relaxations for Network Scheduling Problems

628/1999 *Martin Skutella and Gerhard J. Woeginger:* A PTAS for minimizing the total weighted completion time on identical parallel machines

627/1998 *Jens Gustedt:* Specifying Characteristics of Digital Filters with FilterPro

620/1998 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* Resource Constrained Project Scheduling: Computing Lower Bounds by Solving Minimum Cut Problems

619/1998 *Rolf H. Möhring, Martin Oellrich, and Andreas S. Schulz:* Efficient Algorithms for the Minimum-Cost Embedding of Reliable Virtual Private Networks into Telecommunication Networks

618/1998 *Friedrich Eisenbrand and Andreas S. Schulz:* Bounds on the Chvátal Rank of Polytopes in the 0/1-Cube

617/1998 *Andreas S. Schulz and Robert Weismantel:* An Oracle-Polynomial Time Augmentation Algorithm for Integer Proramming

616/1998 *Alexander Bockmayr, Friedrich Eisenbrand, Mark Hartmann, and Andreas S. Schulz:* On the Chvátal Rank of Polytopes in the 0/1 Cube

615/1998 *Ekkehard Köhler and Matthias Kriesell:* Edge-Dominating Trails in AT-free Graphs

613/1998 *Frederik Stork:* A branch and bound algorithm for minimizing expected makespan in stochastic project networks with resource constraints

612/1998 *Rolf H. Möhring and Frederik Stork:* Linear preselective policies for stochastic project scheduling

609/1998 *Arfst Ludwig, Rolf H. Möhring, and Frederik Stork:* A computational study on bounding the makespan distribution in stochastic project networks

605/1998 *Friedrich Eisenbrand:* A Note on the Membership Problem for the Elementary Closure of a Polyhedron

596/1998 *Andreas Fest, Rolf H. Möhring, Frederik Stork, and Marc Uetz:* Resource Constrained Project Scheduling with Time Windows: A Branching Scheme Based on Dynamic Release Dates

595/1998 *Rolf H. Möhring Andreas S. Schulz, and Marc Uetz:* Approximation in Stochastic Scheduling: The Power of LP-based Priority Policies

591/1998 *Matthias Müller–Hannemann and Alexander Schwartz:* Implementing Weighted *b*-Matching Algorithms: Towards a Flexible Software Design

590/1998 *Stefan Felsner and Jens Gustedt and Michel Morvan:* Interval Reductions and Extensions of Orders: Bijections to Chains in Lattices

584/1998 *Alix Munier, Maurice Queyranne, and Andreas S. Schulz:* Approximation Bounds for a General Class of Precedence Constrained Parallel Machine Scheduling Problems

577/1998 *Martin Skutella:* Semidefinite Relaxations for Parallel Machine Scheduling

566/1997 *Jens Gustedt:* Minimum Spanning Trees for Minor-Closed Graph Classes in Parallel

565/1997 *Andreas S. Schulz, David B. Shmoys, and David P. Williamson:* Approximation Algorithms

561/1997 *Matthias Müller–Hannemann:* High Quality Quadrilateral Surface Meshing Without Template Restrictions: A New Approach Based on Network Flow Techniques

559/1997 *Matthias Müller–Hannemann and Karsten Weihe:* Improved Approximations for Minimum Cardinality Quadrangulations of Finite Element Meshes

554/1997 *Rolf H. Möhring and Matthias Müller–Hannemann:* Complexity and Modeling Aspects of Mesh Refinement into Quadrilaterals

551/1997 *Hans Bodlaender, Jens Gustedt and Jan Arne Telle:* Linear-Time Register Allocation for a Fixed Number of Registers and no Stack Variables

550/1997 *Karell Bertet, Jens Gustedt and Michel Morvan:* Weak-Order Extensions of an Order

549/1997 *Andreas S. Schulz and Martin Skutella:* Random-Based Scheduling: New Approximations and LP Lower Bounds

Reports may be requested from:

S. Marcus
Fachbereich Mathematik, MA 6–1
TU Berlin
Straße des 17. Juni 136
D-10623 Berlin – Germany

e-mail: Marcus@math.TU-Berlin.DE

Reports are available via anonymous ftp from:

ftp.math.tu-berlin.de
cd pub/Preprints/combi
file Report–<number>–<year>.ps.Z