ELSEVIER

European Journal of Operational Research 107 (1998) 272–288

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

A branch and bound algorithm for the resource-constrained project scheduling problem [1]

Peter Brucker *, Sigrid Knust, Arno Schoo, Olaf Thiele

Universität Osnabrück, Fachbereich Mathematik/Informatik, Albrechtstrasse 28, 49069 Osnabrück, Germany

Received 1 May 1996; received in revised form 1 April 1997

Abstract

A branch and bound algorithm is presented for the resource-constrained project scheduling problem (RCPSP). Given are n activities which have to be processed without preemptions. During the processing period of an activity constant amounts of renewable resources are needed where the available capacity of each resource type is limited. Furthermore, finish–start precedence relations between the activities are given. The objective is to determine a schedule with minimal makespan. The branching scheme starts from a graph representing a set of conjunctions (the classical finish–start precedence constraints) and disjunctions (induced by the resource constraints). Then it either introduces disjunctive constraints between pairs of activities or places these activities in parallel. Concepts of immediate selection are developed in connection with this branching scheme. Immediate Selection allows to add conjunctions as well as further disjunctions and parallelity relations. Computational results based on the test data of Kolisch et al. (Kolisch, R., Sprecher, A., Drexl, A., 1995. Management Science 41, 1693–1703.) and Kolisch and Sprecher (Kolisch, R., Sprecher, A., 1997. PSPLIB – A project scheduling problem library, EJOR 96, pp. 205–216.) are reported.

Keywords: Branch and bound method; Immediate selection; Resource-constrained project scheduling problem; Disjunctive graph model

1. Introduction

The *resource-constrained project scheduling problem* (RCPSP) may be formulated as follows: Given are n activities $1,\ldots,n$ and r renewable resources $R_1,\ldots,R_r$. A constant amount of b_k units of resource R_k is available at any time. Activity i must be processed for p_i time units, where preemption is not allowed. During this time period a constant amount of r_{ik} units of resource R_k is occupied. The values b_k, p_i and r_{ik} are supposed to be non-negative integers. Furthermore, there are finish–start precedence relations defined between the activities. The objective is to determine starting times S_i for the activities $i=1,\ldots,n$ in such a way that:

* Corresponding author. E-mail: Peter.Brucker@mathematik.uni-osnabrueck.de.

[1] Supported by the Deutsche Forschungsgemeinschaft, Project 'Komplexe Maschinen-Schedulingprobleme'.

0377-2217/98/$19.00
PII S0377-2217(97)00335-4

- at each time t the total resource demand is less than or equal to the resource availability for each resource type,
- the given precedence constraints are fulfilled, and
- the makespan $C_{\max} = \max_{i=1}^{n} C_i$, where $C_i := S_i + p_i$, is minimized.

A schedule is called *active* if the starting times of the activities are such that no activity can be started earlier without delaying some other activity or violating the constraints. If not stated otherwise we assume that all schedules to be considered are active.

The problem is presented by an activity-on-the-node network with dummy start and end nodes 0 and $n+1$, respectively and $p_0 = p_{n+1} = 0$. We assume $S_0 = 0$ and identify S_{n+1} with the $C_{\max}$-value.

As a generalization of the job-shop scheduling problem the RCPSP is NP-hard in the strong sense (see Blażewicz et al., 1983). Therefore, the branch and bound technique is the most common way to deal with this problem in order to get optimal solutions. Some branch and bound algorithms can be found in Stinson et al. (1978) and Christofides et al. (1987). They use partial schedules that correspond to the nodes of the search tree. Demeulemeester et al. (1994) have shown that the branching scheme of Christofides et al. (1987) is conceptually wrong and may generate non-optimal solutions. Additionally, Demeulemeester and Herroelen (1992) have developed a new powerful depth-first branching scheme with new dominance criteria and bounding rules. A clever extension of the corresponding code can be found in Demeulemeester and Herroelen (1995). Based on a new mathematical formulation Mingozzi et al. (1994) formulated several new lower bounds and presented a branch and bound scheme using lists of feasible subsets to represent partial schedules. A "weighted node packing-bound" was used to improve the speed of their algorithm. Other approaches incorporate graphtheoretical concepts. Bell and Park (1990) implemented a so-called A^*-algorithm using graphs with directed arcs, whereas the branch and bound algorithm of Bartusch et al. (1988) relies on the disjunctive graph model. A complete different attempt was made by Carlier and Latapie (1991). Based on the calculation of upper bounds which provides execution intervals for the activities branching consists of splitting these intervals. Lower bounds are derived from an m-machine relaxation.

In our branch and bound algorithm the nodes of the enumeration tree correspond to sets of feasible schedules which are defined by conjunctions, disjunctions and relations placing activities in parallel. This model provides a suitable representation that allows to derive immediate selection concepts. It is closely related to the schedule graph model proposed by Krämer (1995) for instances of the multi-processor task scheduling problem.

This paper is organized as follows. In Section 2 we lay the foundations of our algorithm. A relation model formulation is given that is used for the representation of schedules. The organization of our branch and bound algorithm and the branching scheme are described. In Section 3 we illustrate different approaches of immediate selection which are very useful to keep the search tree small. Upper and lower bound calculations are described in Sections 4 and 5, respectively. Computational results for various problem instances are presented in Section 6. Finally, in Section 7 some concluding remarks can be found.

2. A general description of the algorithm

In this section we first introduce the notion of schedule schemes which is used to represent sets of feasible schedules. The schedule schemes correspond to the nodes of the enumeration tree of our branch and bound algorithm. Similar concepts have been introduced by Krämer (1995). Then we will give a general description of the branch and bound algorithm.

Let V be the set of all activities $0, 1, \ldots, n, n+1$ and define

$$A := \{(i,j) \mid i,j \in V, i \neq j\}.$$

A *schedule scheme* (C, D, N, F) consists of four disjoint relations $C, D, N, F \subseteq A$. It has the following properties:

- C is antisymmetric (i.e. $(i,j) \in C$ implies $(j,i) \notin C$),

- D, N, F are symmetric (i.e. if (i,j) belongs to one of these sets then (j,i) belongs to the same set),
- for each $(i,j) \in A$ we have either $(i,j) \in D \cup N \cup F$ or one of the conditions $(i,j) \in C$, or $(j,i) \in C$ holds.

The relations in C are called *conjunctions* whereas the set D represents *disjunctions*. We use the symbols $i \to j$ and $i - j$, respectively. The relations in N are called *parallelity relations* and finally, F contains *flexibility relations*. They are denoted by $i \| j$ and $i \sim j$, respectively. Activities i, j with $i \| j$ must be processed in parallel for at least one time unit whereas there is no restriction on activities with $i \sim j$. Due to symmetry we identify $i - j$ with $j - i$, $i \| j$ with $j \| i$, and $i \sim j$ with $j \sim i$. Two activities i and j are called *incompatible* if $i - j$, or $i \to j$, or $j \to i$ holds.

The schedule scheme (C, D, N, F) *represents* the set $\mathscr{S}(C, D, N, F)$ of all schedules with:

- $C_i \leqslant S_j$ (activity j is scheduled after i) for all $i \to j \in C$,
- $C_i \leqslant S_j$ or $C_j \leqslant S_i$ for all $i - j \in D$ (the processing intervals of activities i and j do not overlap),
- $S_i < C_j$ and $S_j < C_i$ (the processing intervals of activities i and j are intersecting) if $i \| j \in N$.

We restrict the set $\mathscr{S}(C, D, N, F)$ to schedules where no activity can be left shifted without violating some fixed relation (conjunctions, disjunctions, and parallelity relations). Note, that a schedule scheme may represent infeasible schedules as well as feasible ones because the resource constraints may be violated. The subset of feasible schedules that are represented is denoted by $\mathscr{S}_f(C, D, N, F)$. Furthermore, $\mathrm{LB}(S)$ denotes a lower bound for the set of feasible solutions represented by the schedule scheme $S = (C, D, N, F)$.

Let C_0 be the set of conjunctions which are defined by the original precedence relations and D_0 an initial set of disjunctions that is induced by the resource constraints for pairs of activities (i.e. we set $i - j \in D_0$ if and only if $r_{ik} + r_{jk} > b_k$ for some $k = 1, \ldots, r$). Then $(C_0, D_0, \emptyset, F)$ with $F = \{i \sim j \mid i \to j \notin C_0,\ j \to i \notin C_0,\ i - j \notin D_0\}$ represents all feasible active schedules.

The root of the branching tree is associated with the scheduling scheme $(C_0, D_0, \emptyset, F)$. Branching is done by choosing some $i \sim j \in F$ and transforming this flexibility relation either into a disjunction $i - j$ (*creating a disjunction*) or into a parallelity relation $i \| j$ (*creating a parallelity relation*). This branching process is repeated until scheduling schemes with $F = \emptyset$ are reached creating a binary search tree. More specifically we have the following branching rule.

Branching rule: For a search tree node v associated with the schedule scheme $S_v = (C_v, D_v, N_v, F_v)$ with $F_v \neq \emptyset$ create two immediate successor nodes s_1 and s_2 representing the schedule schemes:

$$S_{s_1}(i \sim j) = (C_v, D_v \cup \{i - j\}, N_v, F_v \setminus \{i \sim j\}),$$

$$S_{s_2}(i \sim j) = (C_v, D_v, N_v \cup \{i \| j\}, F_v \setminus \{i \sim j\})$$

with a flexibility relation $i \sim j \in F_v$ selected according to some priority rule. Notice, that by this branching the set of feasible schedules corresponding with S_v is split into two disjoint sets.

The priority rule to choose $i \sim j \in F_v$ follows the lines of Krämer (1995). It relies on values w_{i-j} and $w_{i\|j}$ which may be considered as approximations for the lower bounds $\mathrm{LB}(S_{s_1}(i \sim j))$ and $\mathrm{LB}(S_{s_2}(i \sim j))$ respectively. We choose a flexibility relation $i \sim j \in F$ such that the sum $w_{i-j} + w_{i\|j}$ is maximized. If this priority rule fails to determine a unique flexibility relation we specify $i \sim j \in F_v$ randomly. In Section 5 we will describe how to calculate the weights w_{i-j} and $w_{i\|j}$.

Next, we show that for a scheduling scheme $(C, D, N, \emptyset)$ we can either recognize that there exists no feasible schedule which corresponds with $(C, D, N, \emptyset)$ or we can calculate a schedule dominating all schedules which correspond with $(C, D, N, \emptyset)$. Furthermore, both can be done in polynomial time. Thus, vertices v of the enumeration tree with $F_v = \emptyset$ can be treated as leaves.

A schedule scheme $(C', \emptyset, N, \emptyset)$ is called a *transitive orientation* of $(C, D, N, \emptyset)$ if:

- $(C', \emptyset, N, \emptyset)$ is derived from $(C, D, N, \emptyset)$ by fixing all disjunctions $i - j \in D$;
- C' is transitive, i.e. for all $i \to j,\ j \to k \in C'$ we have $i \to k \in C'$.

Obviously, a transitive orientation is acyclic, i.e. the corresponding graph (V, C') is acyclic. Note, that the set of solutions $\mathscr{S}(C, \emptyset, N, \emptyset)$ represented by a transitive orientation contains at most one element because only active schedules are consid-

ered. Of course such a schedule may be infeasible with respect to the resource constraints.

Given a transitive orientation $(C,\emptyset,N,\emptyset)$ of some schedule scheme we construct a schedule by the following procedure. Consider the directed graph (V,C) associated with C. Calculate for each activity $i \in V$ the length r_i of a longest path from the start activity 0 to i (the length of a path to i equals the sum of all processing times of vertices in this path, i excluded). Schedule each activity $i \in V$ at time r_i. The schedule defined by these starting times $S_i = r_i$ is called *earliest start schedule* associated with $(C,\emptyset,N,\emptyset)$ and is denoted by $S_{es}(C)$. The earliest start schedule associated with $(C,\emptyset,N,\emptyset)$ may not be represented by $(C,\emptyset,N,\emptyset)$ because only the set C of conjunctions is considered to construct $S_{es}(C)$, i.e. some parallelity relations may be violated. Due to results of Golumbic (1980) and Möhring (1985) an earliest start schedule $S_{es}(C')$ associated with a transitive orientation $(C',\emptyset,N,\emptyset)$ of a schedule scheme $(C,D,N,\emptyset)$ has always the same makespan, regardless which transitive orientation is chosen.

The following theorem shows that a schedule scheme $(C,D,N,\emptyset)$ can be treated as a leaf of the enumeration tree.

Theorem 1. *Let $(C',\emptyset,N,\emptyset)$ be an arbitrary transitive orientation of a schedule scheme $(C,D,N,\emptyset)$ and let $S_{es}(C')$ be the corresponding earliest start schedule. If $S_{es}(C')$ is feasible it dominates all feasible schedules represented by $(C,D,N,\emptyset)$. Otherwise, the set $\mathcal{S}_f(C,D,N,\emptyset)$ of feasible solutions is empty.*

For a proof of this theorem, which is intuitively clear, we refer to Brucker et al. (1996).

Clearly, a feasible schedule always corresponds to a transitive orientation. Due to this observation and Theorem 1 we can proceed as follows when the set F of flexibility relations is empty. First we decide whether there exists a transitive orientation of $(C,D,N,\emptyset)$ and determine an arbitrary one in case of a positive decision. (A polynomial algorithm for this problem can be found in Korte and Möhring, 1985.) Given a transitive orientation $(C',\emptyset,N,\emptyset)$ we calculate the corresponding earliest start schedule $S_{es}(C')$. If $S_{es}(C')$ is feasible then we compare this schedule with the best schedule found so far. Otherwise, we can ignore $(C,D,N,\emptyset)$.

Finally, we describe the basic organization of our branch and bound method. The branching rule explained above is imbedded in a depth-first search algorithm. The immediate successors s_1 and s_2 of node v are examined according to non-decreasing lower bound values $LB(S_{s_v})$. However, before branching node v we apply some other procedures. First we apply concepts of immediate selection which are illustrated in Section 3. The purpose is to detect successor nodes of the current node v that can be excluded from consideration in order to find an optimal solution. Therefore, immediate selection is an important tool in our branch and bound algorithm. If the set F is not yet empty we continue with the calculation of bounds. We calculate a lower bound $LB(v)$. If this lower bound exceeds the makespan of the best solution found so far we may drop the current node v. Otherwise, we calculate heuristically a feasible solution providing an upper bound $UB(v)$. Each time an improvement is found the upper bound UB of the original problem is updated and a further comparison with the lower bound $LB(v)$ is indicated.

Due to Theorem 1 and the results in Section 3 this branch and bound scheme correctly finds an optimal solution in a finite number of steps.

3. Immediate selection

One of the objectives is to empty the set F of flexibility relations as soon as possible by eliminating relations from F. During branching this is established by "moving" a relation $i \sim j \in F$ either to N or to D. In this section we introduce some concepts for immediate selection which allow to move flexibility relations without branching. Additionally, immediate selection allows to move relations from D to C (fixing disjunctions).

Two main ideas for immediate selection are considered: The first one is to show that some move operation leads to infeasibility. Thus, the complementary move operation must be applied if a feasible schedule scheme should be reached. The corresponding criterion is referred to as feasibility

criterion. The second idea is to show that some modification of a schedule scheme leads to a set of solutions that is dominated by the current best solution. Thus, the complementary move operation must be applied if the current best solution should be improved. We call the corresponding criteria dominance criteria. We discuss different methods to implement these two ideas of immediate selection. Finally, we will discuss how to implement these concepts within the branch and bound method.

3.1. Feasibility criteria

In this section we illustrate different methods of immediate selection which lead to additional modifications of a schedule scheme. These modifications are forced by infeasibility, that is if the complementary modification is applied the resulting schedule scheme would not represent any feasible schedule.

3.1.1. Transitivity conjunctions and positive cycles

A feasible schedule S has the following basic properties:

- $i \to j$ holds if and only if $S_i + p_i \leqslant S_j$ or equivalently $S_j - S_i \geqslant p_i$ is satisfied.
- If $i \| j$ is given, then $S_j - S_i \geqslant -(p_j - 1)$ holds.

To prove the second property assume that $S_j - S_i < -(p_j - 1)$ is fulfilled. Then we have $S_j + p_j < S_i + 1$ which implies that $C_j \leqslant S_i$ holds because all data are integer causing a contradiction to $i \| j$.

Thus, if we define a *distance matrix* $D_S = (d_{ij})_{i,j \in V}$ associated with a schedule scheme $S = (C, D, N, F)$ by

$$d_{ij} = \begin{cases} 0 & \text{if } i = j, \\ p_i & \text{if } i \to j, \\ -(p_j - 1) & \text{if } i \| j, \\ -\infty & \text{otherwise,} \end{cases} \tag{1}$$

then for each pair (i, j) $(i, j = 0, \ldots, n+1)$ the value d_{ij} is a lower bound for the difference $S_j - S_i$. Note, that the relation

$$S_j - S_i \geqslant d_{ij} \tag{2}$$

has the following transitivity property:

$$\text{If } S_j - S_i \geqslant d_{ij} \quad \text{and} \quad S_k - S_j \geqslant d_{jk} \quad \text{then } S_k - S_i \geqslant d_{ij} + d_{jk}. \tag{3}$$

Recall that we have an upper bound UB for the $C_{\max}$-value at our disposal during the search process which is defined by the current best solution. We exploit this fact by adding the constraint

$$S_{n+1} \leqslant \mathrm{UB} - 1 \tag{4}$$

that has to be respected in order to achieve an improvement. Since we have $S_i \geqslant 0$ for each activity i we may derive

$$S_i + p_i \leqslant S_{n+1} \leqslant \mathrm{UB} - 1 \leqslant \mathrm{UB} - 1 + S_j \quad \text{or} \quad S_j - S_i \geqslant p_i - (\mathrm{UB} - 1)$$

for an arbitrary pair of activities i and j. Thus, in Eq. (1) we may replace $-\infty$ by $p_i - (\mathrm{UB} - 1)$.

Due to the transitivity property Eq. (3) we may replace the distance matrix $D_S = (d_{ij})_{i,j \in V}$ by its transitive closure $\overline{D}_S = (\overline{d}_{ij})_{i,j \in V}$. $\overline{D}_S$ can be calculated in $O(n^3)$ time by applying the *Floyd–Warshall algorithm* to D_S (see Papadimitriou and Steiglitz, 1982).

From $\overline{D}_S$ we may derive the following conclusions:

- If

$$\overline{d}_{ij} \geqslant p_i \tag{5}$$

holds, then we may fix the conjunction $i \to j$,

- if

$$\overline{d}_{ii} > 0 \tag{6}$$

holds for some activity i, then there exists no feasible schedule.

A relation $i \to j$ added to C by condition (5) is called a *transitivity conjunction*. Note, that condition (6) indicates a positive cycle of the graph (V, C).

3.1.2. Symmetric triples and extensions

The concepts presented in this section are extensions of dominance rules commonly applied in the field of resource-constrained scheduling and line balancing (see e.g. Johnson, 1988).

A triple of activities (i, j, k) is called *symmetric* if $k \| i$ and $k \| j$ hold and i, j and k cannot be processed simultaneously because of resource con-

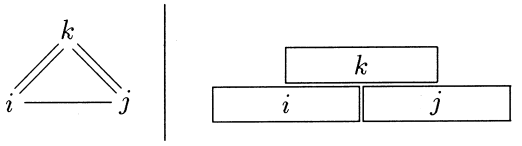


Fig. 1. Symmetric triple.

straints. Thus, we may introduce a disjunction $i - j \in D$. Fig. 1 illustrates a symmetric triple. A procedure with time complexity $O(rn|N|)$ that calculates all symmetric triples (i, j, k) and updates $D = D \cup \{i - j\}$ is easily implemented. This can be seen as follows: We scan each parallelity relation $k \| i \in N$ which takes $O(|N|)$ time. Each activity k of some relation $k \| i \in N$ has at most $O(n)$ further parallelity relations $k \| j \in N$. Finally, checking feasibility of (i, j, k) with respect to the resource capacities is done in $O(r)$ time.

We extend our procedure by incorporating additional conditions that force additional schedule scheme modifications in order to save feasibility. Provided with a symmetric triple (i, j, k) and an arbitrary other activity l six cases based on additional conditions may be described as follows:

1. Let the parallelity relation $l \| i$ be given and assume that j, k and l cannot be processed simultaneously. Then we can update $D = D \cup \{l - j\}$. For illustration see Fig. 2. Let additionally the precedence relation $i \to j$ or $j \to i$ be given. Then we may even fix the conjunction $l \to j \in C$ or $j \to l \in C$, respectively.
2. Suppose that the conditions $p_k - 1 \leqslant p_i$, $p_k - 1 \leqslant p_j$, and $p_k - 1 \leqslant p_l$ hold. Furthermore, assume that i, k and l as well as j, k and l cannot be processed simultaneously. Then we can update $D = D \cup \{k - l\}$ since parallel processing of k and l is excluded. For illustration see Fig. 3.

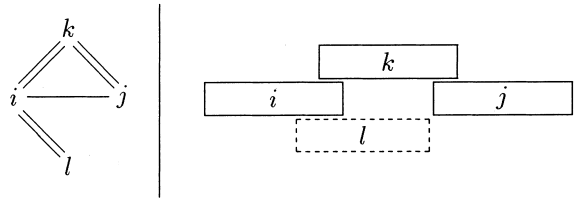


Fig. 2. Extension of a symmetric triple – Condition 1.

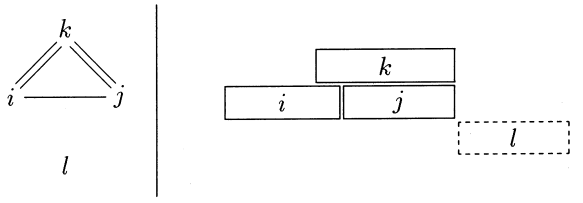


Fig. 3. Extension of a symmetric triple – Condition 2.

3. Suppose that the conditions $p_k - 1 \leqslant p_i$ and $p_k - 1 \leqslant p_l$ hold. Furthermore, assume that the precedence relations $l \to j$ and $i \to j$ are given and i, k and l cannot be processed in parallel. Then the additional conjunction $l \to k \in C$ can be fixed. For illustration see Fig. 4.
4. Suppose that the conditions $p_k - 1 \leqslant p_j$ and $p_k - 1 \leqslant p_l$ hold. Furthermore, assume that the precedence relations $i \to j$ and $i \to l$ are given and j, k and l cannot be processed in parallel. Then the additional conjunction $k \to l \in C$ can be fixed. For illustration see Fig. 5.
5. Suppose that the conditions $p_k - 1 \leqslant p_i$ and $p_k - 1 \leqslant p_j$ hold. Furthermore, assume that the precedence relations $l \to i$ and $l \to j$ ($i \to l$ and $j \to l$) are given. Then we may fix the conjunction $l \to k \in C$ ($k \to l \in C$). For illustration see Fig. 6.

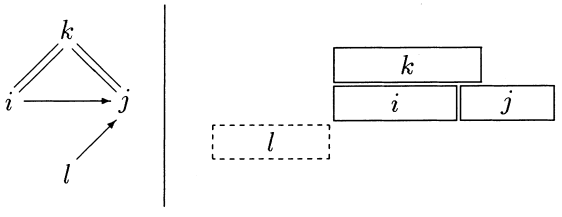


Fig. 4. Extension of a symmetric triple – Condition 3.

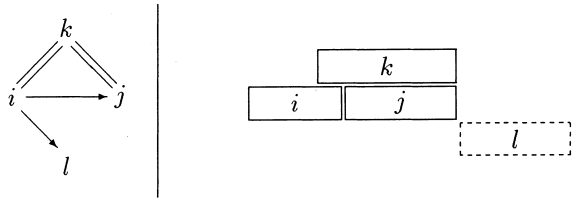


Fig. 5. Extension of a symmetric triple – Condition 4.

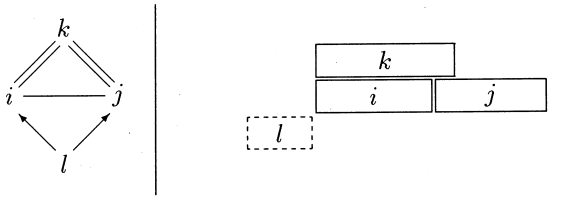


Fig. 6. Extension of a symmetric triple – Condition 5.

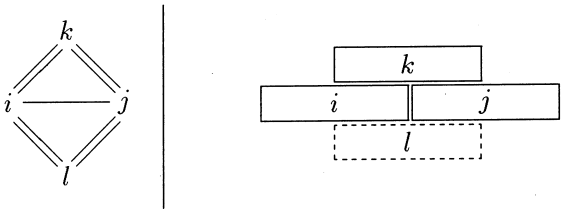


Fig. 7. Extension of a symmetric triple – Condition 6.

6. Let the parallelity relations $l\|i$ and $l\|j$ be given. Then the additional parallelity relation $l\|k$ can be fixed. For illustration see Fig. 7.

It is not difficult to prove that it is correct to fix relations as indicated and that all conditions can be fixed in $O(r^2n^2|N|)$ time (see Schoo, 1996).

3.2. Dominance criteria

In this section we illustrate how classical immediate selection procedures (see Carlier and Pinson, 1989; Brucker et al., 1994a) can be employed for the RCPSP. We first show how heads and tails can be used to fix additional conjunctions and parallelity relations. Then the notion of cliques which is well known from the job-shop problem is generalized and concepts of immediate selection are applied within this general framework.

3.2.1. Additional conjunctions and parallelity relations

A *head* r_i of an activity i is a lower bound for the starting time of i. Symmetrically, a *tail* q_i of an activity i is a lower bound for the time period between the completion of activity i and the optimal makespan. A procedure for calculating heads and tails is presented in Brucker et al. (1994a). This procedure has time complexity $O(|C|)$ and calculates heads and tails according to the formulas:

$$r_i := \max\{r_j + p_j \mid j \to i \in C\}, \tag{7}$$

$$q_i := \max\{p_j + q_j \mid i \to j \in C\}, \tag{8}$$

respectively. These recursive definitions require initializations which are given by $r_0 = q_{n+1} = 0$.

In order to obtain an improvement of the current best upper bound each activity i has to be processed before $UB - q_i$. Due to the fact that all data are integer we have a deadline $d_i = UB - q_i - 1$. Thus, i has to be processed within the time window $[r_i, d_i]$. Let a disjunction $i - j \in D$ be given. If the condition

$$p_i + p_j > d_i - r_j \tag{9}$$

holds, j cannot be processed before i and thus we may fix the additional conjunction $i \to j \in C$.

Now let $D_S = (d_{ij})$ be the distance matrix defined by Eq. (1) where $-\infty$ is replaced by $p_i - (UB - 1)$. If the condition

$$d_{ij} \geqslant -(p_j - 1) \tag{10}$$

is fulfilled, then $S_j - S_i \geqslant d_{ij} \geqslant -(p_j - 1)$ or equivalently $C_j > S_i$ holds. Thus, if Eq. (10) holds for the activities of a disjunction $i - j \in D$ we may fix the additional conjunction $i \to j \in C$ which is called a *direct conjunction*. In Schoo (1996) it is shown that each conjunction fixed by condition (9) is also embraced by condition (10). A procedure with time complexity $O(|D|)$ for fixing additional conjunctions due to Eq. (10) is easily implemented.

Condition (10) is also useful to determine additional parallelity relations. If both conditions

$$d_{ij} \geqslant -(p_j - 1) \quad \text{and} \quad d_{ji} \geqslant -(p_i - 1) \tag{11}$$

are fulfilled we clearly may introduce an additional parallelity relation $i\|j \in N$. Since each pair of activities i and j has to be considered the computational effort is $O(n^2)$.

3.2.2. Clique conjunctions

Provided with heads and tails we may apply classical concepts of immediate selection. The basic idea of these concepts is to improve heads

and tails of activities belonging to a clique in the undirected graph $G_S = (V, E_S)$ associated with a schedule scheme $S = (C, D, N, F)$. The set of edges E_S consists of all pairs of incompatible activities, i.e. $[i, j] \in E_S$ if and only if $i \to j \in C$, or $j \to i \in C$, or $i - j \in D$. Heads and tails can be derived from the distance matrix $\overline{D}_S$. The improved heads and tails are retransformed into distances. This may lead to additional conjunctions due to conditions (5) or (10), respectively.

To improve heads and tails of activities belonging to a clique we use a well known procedure which has been applied in connection with the job-shop problem (see e.g. Brucker, 1995, p. 202). We denote this procedure *Earliest Possible Completion Schedule* (EPCS).

It remains to describe how to calculate suitable cliques of activities that can be conducted to algorithm EPCS. Let $\mathscr{I}(G_S)$ denote the set of all cliques in the graph G_S induced by the schedule scheme $S = (C, D, N, F)$. We could apply EPCS to each clique in $\mathscr{I}(G_S)$. However, since the calculation of $\mathscr{I}(G_S)$ costs exponential computation time we determine only a subset $\{I_1, \ldots, I_q\} \subset \mathscr{I}(G_S)$ of cliques with the following properties:

- Each clique I_k is *maximal*, i.e.

$$I_k \not\subset I \quad \forall I \in \mathscr{I}(G_S) \setminus \{I_k\}, \tag{12}$$

- and

$$I_1 \cup \cdots \cup I_q = \{1, \ldots, n\}. \tag{13}$$

The calculation of the subset $\{I_1, \ldots, I_q\}$ is done by introducing the sets $E_i = \{j \mid i \| j \in N \text{ or } i \sim j \in F\}$ and the weights $\alpha_i := \mid E_i \mid$ for each activity i. These sets and weights are easily calculated during the determination of the initial sets C_0 and D_0. They are updated at each branching, backtracking, or immediate selection step in constant time.

The procedure *Process Cliques* presented below scans the activities according to a permutation π defined by $\alpha_{\pi(1)} \leqslant \cdots \leqslant \alpha_{\pi(n)}$. All activities that are contained in at least one clique which is already generated are marked by an array *mark* of boolean values. If a currently considered activity i is not yet marked *Calculate Clique* (i) calculates a new clique $I(i)$ proceeding as follows:

We start with $I(i) = \{i\}$ and mark activity i. Then all activities $j \neq i$ are scanned according to the permutation π in order to augment the clique $I(i)$. If an activity j is not contained in the forbidden set $\bigcup_{k \in I(i)} E_k$ it is added to $I(i)$, marked (if it is not marked yet), and the forbidden set has to be updated. Obviously, the procedure provides a clique that is maximal in the sense of condition (12). Since updating the forbidden set is done in less than $O(\max_i \alpha_i)$ the computational effort is $O(n \max_i \alpha_i)$.

The calculation of cliques is illustrated by the following example with seven activities. The sets E_i are given by the arcs incident to node i in the graph shown in Fig. 8. Node i is marked with its degree-weight $\alpha_i := \mid E_i \mid$. We scan the activities according to the permutation $\pi = (1, 2, 6, 5, 7, 3, 4)$. Then four cliques are generated which are shown in Fig. 9. The arrow indicates the first activity i of clique $I(i)$ and the marked nodes are filled.

By procedure Process Cliques a set of maximal cliques $I(i)$ that fulfill condition (13) is calculated. The permutation π is chosen to obtain large cliques and therefore to keep the number of cliques small. Additionally, the algorithm EPCS is applied to each clique generated. The boolean variable *success* indicates whether any direct conjunction could be fixed as a consequence of improved distances. This is checked by procedure *Fix Direct Conjunctions*. *Process Cliques* will be repeated within a procedure *Immediate Selection* until we have *success* = FALSE, i.e. no further conjunction has been fixed (see Section 3.3).

```
PROCEDURE Process Cliques
1.  FOR k := 1 TO n DO mark(k): = FALSE
2.  FOR k := 1 TO n DO BEGIN
3.        i := π(k);
4.        IF not mark(i) THEN BEGIN
5.          I(i): = Calculate Clique (i);
6.          calculate heads and tails of all j ∈ I(i);
7.          calculate improved heads r'_j by algo-
            rithm EPCS (I(i));
8.          FOR ALL j ∈ I(i) DO
9.            d_0j := r^p rime_j;
10.         calculate improved tails q'_j by algo-
            rithm EPCS (I(i));
11.         FOR ALL j ∈ I(i) DO BEGIN
12.           d_j,n+1 := q'_j + p_j;
```

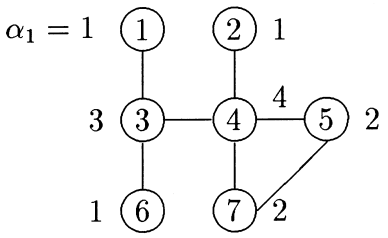


Fig. 8. $E(i)$-graph.

13. *mark*(j) := TRUE;
END
END
END
14. *success*: = Fix Direct Conjunctions;
15. RETURN (*success*);

3.3. Updating distances and applying immediate selection

As a foundation of immediate selection concepts we need the distance matrix $\overline{D}_S$ that provides transitive distances. We apply an extended version of the Floyd–Warshall algorithm denoted as procedure *Calculate Transitive Distances* to the initialized matrix. Within this extension we fix transitivity conjunctions according to Eq. (5) and additional parallelity relations according to Eq. (11). Furthermore, we detect cycles according to Eq. (6). This extended version still runs in $O(n^3)$ time. The computational cost to update the distances in connection with a branching step or a single immediate selection decision can be reduced from $O(n^3)$ to $O(n^2)$ (see Bartusch et al., 1988).

Procedure *Calculate Transitive Distances* is illustrated by the example with five activities shown in Fig. 10(a).

The initial distance matrix D_S due to Eq. (1) is given by

$$D_S = \begin{pmatrix} 0 & 2 & -\infty & -\infty & -\infty \\ -\infty & 0 & 3 & -\infty & -1 \\ -\infty & -\infty & 0 & 4 & -\infty \\ -\infty & -\infty & -\infty & 0 & 5 \\ -\infty & -2 & -\infty & -\infty & 0 \end{pmatrix}.$$

In the Floyd–Warshall algorithm for $k = 0, \ldots, n+1$ the triple-operation

$$d_{ij} := \max\{d_{ij}, d_{ik} + d_{kj}\}$$

is applied to every matrix entry d_{ij} $(i, j \in V)$. We stop, if some diagonal element becomes positive indicating a cycle. After applying this procedure to D_S, we obtain the matrix

$$\overline{D}_S = \begin{pmatrix} 0 & 2 & 5 & 9 & 14 \\ -\infty & 0 & 3 & 7 & 12 \\ -\infty & -\infty & 0 & 4 & 9 \\ -\infty & -\infty & -\infty & 0 & 5 \\ -\infty & -2 & 1 & 5 & 10 \end{pmatrix}.$$

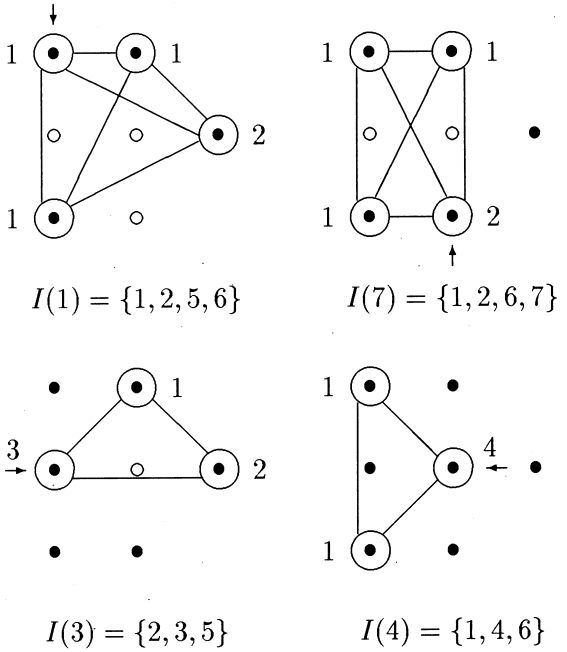


Fig. 9. Generated maximal cliques.

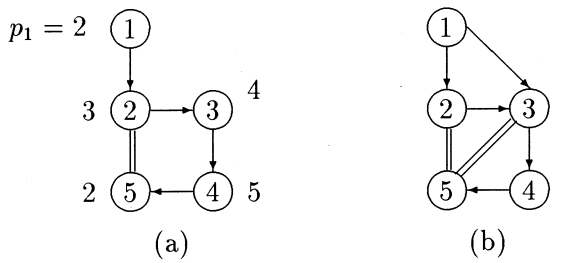


Fig. 10. Creating new relations.

Since $\overline{d}_{13} = 5 \geqslant p_1 = 2$ holds, we can fix the conjunction $1 \rightarrow 3$ according to Eq. (5). Since $\overline{d}_{35} = 9 \geqslant -(p_5 - 1) = -1$ and $\overline{d}_{53} = 1 \geqslant -(p_3 - 1) = -3$ are fulfilled, we have the additional parallelity relation $3\|5$ according to Eq. (4). However, we can stop because we detect a positive cycle according to Eq. (6) since $\overline{d}_{55} = 10 > 0$. No feasible schedule exists.

Below, we present the procedure *Immediate Selection* which collects immediate selection concepts described in the previous sections. The boolean variable *feasible* indicates whether a cycle of positive length is detected by procedure *Calculate Transitive Distances*. Note, that this procedure is only applied to an initialized distance matrix after a backtracking step. Each time a new conjunction or parallelity relation is fixed we update the distances in $O(n^2)$ time. The boolean variable *success* indicates whether additional direct conjunctions are fixed by procedure *Fix Direct Conjunctions*. Recall from Section 3.2.2 that this procedure is also called by procedure *Process Cliques* which is repeated until *success* = FALSE holds.

```
PROCEDURE Immediate Selection (S_v)
1. feasible: = Calculate Transitive Distances;
2. IF feasible THEN BEGIN
3.    Calculate Symmetric Triples;
4.    success: = Fix Direct Conjunctions;
5.    REPEAT
6.       success: = Process Cliques;
7.    UNTIL success = FALSE;
   END;
```

We already mentioned that immediate selection is an important tool to reduce the search tree of our branch and bound algorithm. It is evident that immediate selection decisions additionally may influence lower and upper bound calculations because the solution set becomes more restricted. Therefore, we apply the procedure *Immediate Selection* to each search tree node v before lower and upper bounds are calculated.

4. Calculation of upper bounds

In the root of the search tree we calculate a good initial solution by applying a rather time consuming tabu-search algorithm. Another heuristic is applied to provide an upper bound for each search tree node v different from the root node. Before we present the main ideas of the tabu-search heuristic we will describe the second heuristic.

For a given schedule scheme $S = (C, D, N, F)$ the heuristic calculates a "good" schedule that satisfies all conjunctions in C and disjunctions in D taking into account some parallelity relations from N (the decision problem whether there exists a feasible schedule that corresponds to a given schedule scheme has been shown to be NP-hard by Krämer, 1995). It is based on a priority rule which is dynamic in the sense that the priority of an activity differs from one state of the scheduling process to another.

Basic for this heuristic is the concept of parallelity components. A *parallelity component* is a connectivity component in the undirected graph induced by the parallelity relations of N. Let P_1 and P_2 be two different parallelity components. Then a precedence relation $P_1 \rightarrow P_2$ is introduced if and only if activities $i \in P_1$ and $j \in P_2$ exist such that $i \rightarrow j \in C$ holds (confer Fig. 11). These parallelity components and the precedences between

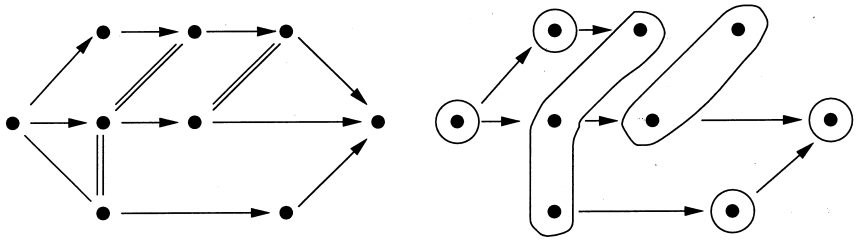


Fig. 11. Example for parallelity components ○ for a graph with vertices •.

components have a high impact on the priority of an activity.

The heuristic constructs a feasible schedule by adding one activity to the current partial schedule in each step. It can be described as follows:

1. Calculate the parallelity components and the corresponding precedence constraints between these components. Determine a static weight β_i for each activity i defined by $\beta_i := \sum_{k=1}^{r} r_{ik}/b_k$.
2. If no parallelity component without predecessor components exists, then we go to (5). Otherwise, one of the parallelity components without predecessor is chosen which contains a job with smallest $p_i + q_i$-value. The component P is now scheduled until it is empty in the following way:
3. Choose an activity $i \in P$ with greatest β_i-value. Calculate an earliest starting time r_i at which i can be started if it is added to the partial schedule. More precisely, let

$$h_i := \max\{\max_{j \to i}\{s_j + p_j | j \text{ planned}\},$$

$$\min_{k \parallel i}\{s_k - (p_i - 1) | k \text{ planned}\}.$$

   We calculate the earliest starting time $r_i \geqslant h_i$ such that activity i can be scheduled without violating the disjunctions and resource constraints and schedule i at time r_i. We eliminate i from P, update the resource profile of the partial schedule and repeat Step (3) until P is empty.
4. When P becomes empty, we eliminate this component with its precedences and go to (2).
5. A cycle between some parallelity components exists and we choose an activity i without unfinished conjunctive predecessor using the same criterion as in (3). We schedule i and update the parallelity components with its precedences. If now a component without predecessors exists, we go to (2). Otherwise, we repeat Step (5).

The tabu-search heuristic makes use of heuristic H described above: Given a schedule scheme $S = (C, D, N, F)$ heuristic H provides a schedule $s = H(S)$. Based on this schedule a new schedule scheme $S' = (C', D', N', F')$ is calculated which provides the next schedule $s' = H(S')$, etc. For a more detailed description we refer to Baar et al. (1997).

5. Calculation of lower bounds

Several methods for calculating lower bounds for the RCPSP are available. We tested different methods in connection with our branch and bound algorithm. The main result of these tests was that the lower bound LB_2 from Mingozzi et al. (1994) turned out to be the best.

The corresponding bounding procedure is based on a linear program which relaxes the precedence constraints and allows preemption. To describe the linear program which calculates the lower bound associated with the schedule scheme (C, D, N, F) we introduce some notations.

A subset X of activities is called *feasible* if all $i \in X$ may be processed simultaneously, i.e. there are no conjunctions or disjunctions between any pair of activities $i, j \in X$ and all resource constraints are respected. Otherwise, X is called *infeasible*.

A feasible set X is called *dominated* if it is a proper subset $X \subset Y$ of some feasible set Y. Otherwise, a feasible set X is called *non-dominated*.

We denote all non-dominated feasible sets by $X_1, X_2, \ldots, X_q$. With each set X_j we associate an incidence vector $a_j \in \{0,1\}^n$ defined by

$$a_{ij} := \begin{cases} 1 & \text{if } i \in X_j, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, let x_j be a variable denoting the number of time units where all activities in X_j are processed simultaneously. Then an LP-formulation of the relaxation may be written as follows:

$$\min \quad \sum_{j=1}^{q} x_j \tag{14}$$

$$\text{s.t.} \quad \sum_{j=1}^{q} a_{ij} x_j \geqslant p_i \quad (i = 1, \ldots, n), \tag{15}$$

$$x_j \geqslant 0 \quad (j = 1, \ldots, q). \tag{16}$$

To show that this linear program yields a lower bound for the original problem consider an optimal schedule S_{opt}. Let $0 = t_0 < t_1 < \cdots < t_k = C_{\max}$ define time intervals $I_i = [t_i, t_{i+1}]$ $(i = 0, \ldots, k-1)$ of length $y_i = t_{i+1} - t_i$ such that a set Y_i of activities is processed during the whole time period I_i. Determine for each set Y_i a corre-

sponding set $X_{v(i)}$ with $Y_i \subseteq X_{v(i)}$. Furthermore, define $x_j = \sum_{v(i)=j} y_i$. Then $x = (x_j)$ is a feasible solution for the linear program (14)–(16) such that the sum $\sum_{j=1}^{q} x_j$ is equal to the solution value of S_{opt}.

A schedule for the relaxed problem can be derived quite easily from the solution vector $x^0 = (x_1^0, \ldots, x_q^0)$ if the slack variables

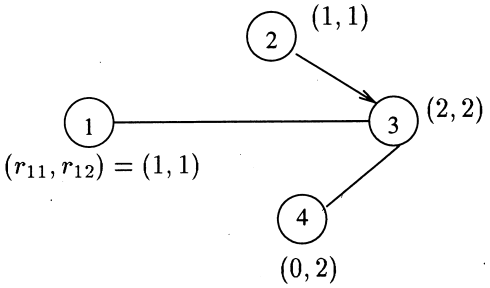
$$s_i^0 := -p_i + \sum_{j=1}^{q} a_{ij} x_j^0 \tag{17}$$

are given. If $s_i^0 > 0$ then all we have to do is to drop s_i^0 units of the processing parts of activity i at arbitrary places.

To calculate all incidence vectors which correspond to non-dominated feasible sets we enumerate all incidence vectors $a = (a_1, \ldots, a_n)$ which correspond to feasible sets and check for each a whether the corresponding set is dominated.

To accomplish this we define an enumeration tree T in which the vertices represent partial strings $(a_1, \ldots, a_l)$ $(1 \leqslant l \leqslant n)$. A vertex $(a_1, \ldots, a_l)$ of T has at most two sons. $(a_1, \ldots, a_l, 0)$ is one son. A second son $(a_1, \ldots, a_l, 1)$ exists only if the set which corresponds to $(a_1, \ldots, a_l, 1, 0, \ldots, 0)$ is feasible which can be checked easily. A set is feasible if and only if it is represented by a leaf of T. To enumerate all feasible sets we scan the tree T in lexicographic order, which is illustrated by the following example with four activities and two resources.

Example 1. Let $n = 4$, $r = 2$, $b_1 = 2$, $b_2 = 3$, $C = \{2 \to 3\}$ and the resource consumptions of the activities given by



Then $D_0 = \{i - j \mid r_{ik} + r_{jk} > b_k \text{for some } k \in \{1, 2\}\} = \{1 - 3, 3 - 4\}$. The calculation of the non-dominated sets is shown in Fig. 12. We obtain eight feasible subsets, only four of them are non-dominated (indicated by an arrow).

Unfortunately, the number of non-dominated feasible sets grows exponentially with the number n of activities. For $n = 60$ it was not possible to store all these columns. However, it is not necessary to store all columns. We can work with only a few non-dominated feasible sets at a time and generate new feasible sets only when they are really needed. Such an approach, which has already been suggested by Gilmore and Gomory (1961) is called delayed column-generation technique. We implemented a strategy (for details see Baar et al., 1997) where, outgoing from the current basis solution, we generate a fixed number of additional columns which may improve the objective value

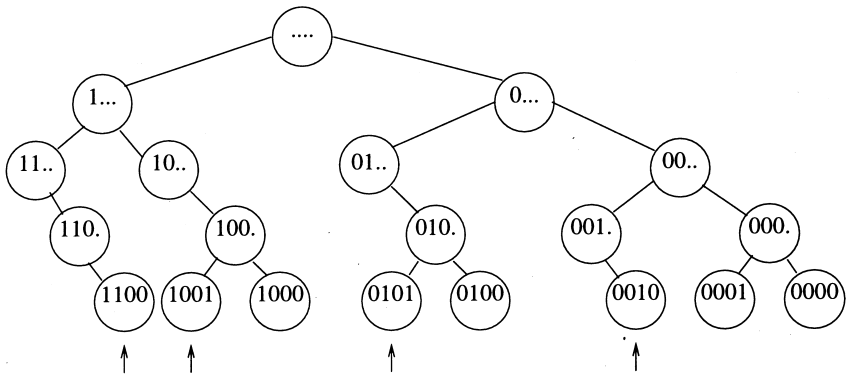


Fig. 12. Calculation of all non-dominated sets.

when entering the basis in the next simplex iteration. With these techniques we calculated LB_2 for available test data with up to 100 activities.

x^0 and the slack values s_i^0 are used to calculate the weights w_{i-j} and $w_{i\|j}$ which we used to decide which $i \sim j \in F$ defines the branching from (C, D, N, F). We set:

$$w_{k-l} := \max\{0, p_{k\|l} - (s_k^0 + s_l^0)\},$$

$$w_{k\|l} := \max_{i-j \in D_{k\|l}} w_{i-j},$$

where $p_{k\|l}$ is the sum of the parallel processing parts in the solution x^0 of the linear program, that is the sum of all x_j^0 where both activities k and l are contained in X_j, and $D_{k\|l}$ is the set of all new relations $i \to j$ induced by Eq. (5). For a (heuristic) justification of these weights we refer to Schoo (1996).

6. Computational results

In this section we present some computational results for the algorithm developed in the previous sections, which solves the RCPSP. In Section 6.1 the tested problem instances are described. Section 6.2 is devoted to the computational results.

6.1. Test data

We investigated the performance of our algorithm with respect to various benchmark instances generated by Kolisch et al. (1995) and Kolisch and Sprecher (1997). They defined some control parameters and developed the generator PROGEN that transposes the given odds into some problem data. The generator allows to vary the number of activities n as well as the number of resources r. The processing time and the resource demands of an activity are randomly chosen integers in the interval $[1, 10]$. In order to obtain different types of instances the following parameters were defined:

- A *complexity factor* $C \geqslant 1$ indicates the average number of immediate successors of an activity (the source and its successors as well as the sink and its predecessors are treated separately).
- A *resource factor* $\mathrm{RF} \in [0, 1]$ describes the average number of limited resources that are necessary to process an activity. $\mathrm{RF} = 1$ indicates a problem where each activity demands each resource, whereas $\mathrm{RF} = 0$ leads to a problem without resource demands and therefore without resource restrictions.
- A *capacity factor* $\mathrm{RS} \in [0, 1]$ is responsible for the resource capacities which depend on the generated resource demands. $\mathrm{RS} = 1$ indicates a problem where no resource conflict occurs. In case of $\mathrm{RS} = 0$ the capacity of a resource type equals the maximum resource demand of this type in order to guarantee feasibility.

Kolisch et al. (1995) and Kolisch and Sprecher (1997) specified a large number of parameter constellations and generated problem groups, each group consisting of 10 instances. For $n = 30, 60$ and $r = 4$ they obtained 48 groups in each case.

6.2. Computational results

The results presented in this section were obtained using a SUN/Sparc 20/801 workstation with operating system Solaris 2.5 and 64 MB general storage (80 MHz +1MB SC). The algorithms were coded in C. We used CPLEX as LP-solver to provide the lower bounds.

The computational results obtained by our implementation of the algorithm for the 480 PROGEN-instances with $n = 30$ and $r = 4$ are presented in Table 1. For each group the parameters are specified and the number "sol" of problems which could be solved to optimality within 1 h is indicated. For the problems which could be solved within the time-limit we determined the average computation time (min : sec) as well as the average number of visited search tree nodes. The column "nodeso" denotes the average numbers of visited notes until the optimum was found. For those problems that could not be solved to optimality in 1 h we present the average number of search tree nodes and in the column "nodesb" the average number of nodes until the best solution was found.

Four hundred and twenty five of the 480 test problems could be solved to optimality in at most

Table 1
PROGEN-instances with $n = 30$ and $r = 4$

Group	Parameters			Sol	Within time-limit			Time-limit	
	C	RF	RS		Time	Nodes	Nodeso	Nodes	Nodesb
j30_17	1.5	0.25	0.2	10	0 : 06	3	1	–	–
j30_18	1.5	0.25	0.5	10	0 : 01	1	1	–	–
j30_19	1.5	0.25	0.7	10	0 : 00	1	1	–	–
j30_20	1.5	0.25	1.0	10	0 : 00	1	1	–	–
j30_21	1.5	0.50	0.2	9	7 : 13	7562	1	36 808	1
j30_22	1.5	0.50	0.5	7	2 : 02	1131	6	55 680	1
j30_23	1.5	0.50	0.7	9	1 : 08	532	492	51 818	1
j30_24	1.5	0.50	1.0	10	0 : 00	1	1	–	–
j30_25	1.5	0.75	0.2	10	2 : 22	1476	155	–	–
j30_26	1.5	0.75	0.5	3	5 : 50	3812	1	52 528	9303
j30_27	1.5	0.75	0.7	7	2 : 10	1852	1835	47 056	1
j30_28	1.5	0.75	1.0	10	0 : 00	1	1	–	–
j30_29	1.5	1.00	0.2	8	10 : 21	8017	5198	51 524	26 348
j30_30	1.5	1.00	0.5	2	5 : 29	3050	1868	48 102	4992
j30_31	1.5	1.00	0.7	8	0 : 00	1	1	50 615	1
j30_32	1.5	1.00	1.0	10	0 : 00	1	1	–	–
j30_33	1.8	0.25	0.2	10	0 : 07	9	1	–	–
j30_34	1.8	0.25	0.5	10	0 : 03	1	1	–	–
j30_35	1.8	0.25	0.7	10	0 : 00	1	1	–	–
j30_36	1.8	0.25	1.0	10	0 : 00	1	1	–	–
j30_37	1.8	0.50	0.2	10	1 : 11	576	80	–	–
j30_38	1.8	0.50	0.5	8	2 : 03	1563	2	48 709	21 990
j30_39	1.8	0.50	0.7	10	0 : 00	1	1	–	–
j30_40	1.8	0.50	1.0	10	0 : 00	1	1	–	–
j30_41	1.8	0.75	0.2	10	2 : 13	1225	492	–	–
j30_42	1.8	0.75	0.5	9	2 : 24	1754	1698	52 596	1
j30_43	1.8	0.75	0.7	9	0 : 02	5	5	41 105	2
j30_44	1.8	0.75	1.0	10	0 : 00	1	1	–	–
j30_45	1.8	1.00	0.2	9	10 : 14	9585	7394	51 966	51 871
j30_46	1.8	1.00	0.5	3	12 : 09	9279	5171	51 431	6
j30_47	1.8	1.00	0.7	7	0 : 00	1	1	62 587	1
j30_48	1.8	1.00	1.0	10	0 : 00	1	1	–	–
j30_49	2.1	0.25	0.2	10	0 : 08	3	1	–	–
j30_50	2.1	0.25	0.5	10	0 : 00	1	1	–	–
j30_51	2.1	0.25	0.7	10	0 : 00	1	1	–	–
j30_52	2.1	0.25	1.0	10	0 : 00	1	1	–	–
j30_53	2.1	0.50	0.2	10	0 : 37	36	1	–	–
j30_54	2.1	0.50	0.5	9	4 : 51	5940	2	40 869	1
j30_55	2.1	0.50	0.7	10	0 : 02	1	1	–	–
j30_56	2.1	0.50	1.0	10	0 : 00	1	1	–	–
j30_57	2.1	0.75	0.2	10	0 : 53	194	35	–	–
j30_58	2.1	0.75	0.5	10	5 : 22	5628	4717	–	–
j30_59	2.1	0.75	0.7	7	2 : 23	2357	1637	58 740	1
j30_60	2.1	0.75	1.0	10	0 : 00	1	1	–	–
j30_61	2.1	1.00	0.2	10	0 : 33	81	37	–	–
j30_62	2.1	1.00	0.5	6	8 : 27	7205	6324	40 955	1
j30_63	2.1	1.00	0.7	5	0 : 00	1	1	46 321	12
j30_64	2.1	1.00	1.0	10	0 : 00	1	1	–	–

Table 2
PROGEN-instances with $n = 60$ and $r = 4$

Group	Parameters			Ver	Within time-limit			Time-limit	
	C	RF	RS		Time	Nodes	Nodeso	Nodes	Nodesb
j60_1	1.5	0.25	0.2	8	0 : 32	32	32	7964	1
j60_2	1.5	0.25	0.5	8	0 : 01	1	1	8372	1
j60_3	1.5	0.25	0.7	9	0 : 01	1	1	10 086	1
j60_4	1.5	0.25	1.0	10	0 : 00	1	1	–	–
j60_5	1.5	0.50	0.2	0	–	–	–	5482	1
j60_6	1.5	0.50	0.5	6	0 : 17	21	21	5992	1
j60_7	1.5	0.50	0.7	10	0 : 00	1	1	–	–
j60_8	1.5	0.50	1.0	10	0 : 00	1	1	–	–
j60_9	1.5	0.75	0.2	0	–	–	–	4028	30
j60_10	1.5	0.75	0.5	7	0 : 00	1	1	5695	1
j60_11	1.5	0.75	0.7	10	0 : 00	1	1	–	–
j60_12	1.5	0.75	1.0	10	0 : 00	1	1	–	–
j60_13	1.5	1.00	0.2	0	–	–	–	4070	1
j60_14	1.5	1.00	0.5	5	0 : 04	1	1	3026	1
j60_15	1.5	1.00	0.7	10	0 : 00	1	1	–	–
j60_16	1.5	1.00	1.0	10	0 : 00	1	1	–	–
j60_17	1.8	0.25	0.2	8	1 : 28	76	38	8032	1
j60_18	1.8	0.25	0.5	10	0 : 01	1	1	–	–
j60_19	1.8	0.25	0.7	8	0 : 01	1	1	10 096	1
j60_20	1.8	0.25	1.0	10	0 : 00	1	1	–	–
j60_21	1.8	0.50	0.2	0	–	–	–	6679	1
j60_22	1.8	0.50	0.5	7	0 : 19	10	10	8440	71
j60_23	1.8	0.50	0.7	10	0 : 00	1	1	–	–
j60_24	1.8	0.50	1.0	10	0 : 00	1	1	–	–
j60_25	1.8	0.75	0.2	0	–	–	–	4700	1
j60_26	1.8	0.75	0.5	5	0 : 00	1	1	4997	1
j60_27	1.8	0.75	0.7	10	0 : 00	1	1	–	–
j60_28	1.8	0.75	1.0	10	0 : 00	1	1	–	–
j60_29	1.8	1.00	0.2	0	–	–	–	3502	1
j60_30	1.8	1.00	0.5	3	0 : 01	1	1	3646	1
j60_31	1.8	1.00	0.7	10	0 : 00	1	1	–	–
j60_32	1.8	1.00	1.0	10	0 : 00	1	1	–	–
j60_33	2.1	0.25	0.2	7	3 : 43	281	4	7950	94
j60_34	2.1	0.25	0.5	9	1 : 00	152	27	10 953	1
j60_35	2.1	0.25	0.7	8	0 : 01	1	1	10 922	1
j60_36	2.1	0.25	1.0	10	0 : 00	1	1	–	–
j60_37	2.1	0.50	0.2	0	–	–	–	6848	13
j60_38	2.1	0.50	0.5	5	0 : 01	1	1	5937	1
j60_39	2.1	0.50	0.7	9	0 : 00	1	1	7607	1
j60_40	2.1	0.50	1.0	10	0 : 00	1	1	–	–
j60_41	2.1	0.75	0.2	0	–	–	–	4598	1
j60_42	2.1	0.75	0.5	5	0 : 01	1	1	4797	21
j60_43	2.1	0.75	0.7	9	0 : 00	1	1	6087	1
j60_44	2.1	0.75	1.0	10	0 : 00	1	1	–	–
j60_45	2.1	1.00	0.2	0	–	–	–	4819	22
j60_46	2.1	1.00	0.5	1	0 : 00	1	1	4318	1
j60_47	2.1	1.00	0.7	9	0 : 02	1	1	5356	1
j60_48	2.1	1.00	1.0	10	0 : 00	1	1	–	–

1 h. For problems not solved in this time we compared the best solution value after 1 h with the optimum found by Demeulemeester and Herroelen (1995). The average percentage distance from the optimum was 1.2%, the maximal deviation 4.0%. This fact and the columns "nodeso" and "nodesb", respectively show that a truncated branch and bound procedure already obtains quite good results. The memory requirement for solving problems with 30 activities did not exceed 12 MB. Here, the lower bounds were calculated without column-generation.

Not only for problems with RS = 1.0 where no resource conflicts occur, the search tree degenerates to the root (nodes = 1). This shows that upper and lower bound calculation has a high quality.

Our algorithm needs more time if the factor RS is in the middle range (RS = 0.5). Then for many pairs of activities we have the choice of placing them in parallel or scheduling them one after the other. Therefore, F contains many flexibility relations and we have many possibilities for branching which is time consuming.

The computational results for the 480 PROGEN-instances with $n = 60$ and $r = 4$ are presented in Table 2. Not for all these instances the optimal values are known. Therefore we used the lower bounds (calculated with column-generation) to obtain bounds for the quality of our solutions. The number of problems for which optimality could be verified within 1 h is indicated in the column "ver".

Three hundred and twenty six of the 480 test problems could be verified in at most 1 h. The average percentage distance from the lower bounds was 4.8%, the maximal deviation 30.8%. The memory requirement for solving problems with 60 activities did not exceed 10 MB.

7. Concluding remarks

A branch and bound algorithm based on the representation of RCPSPs by schedule schemes has been developed. Classical concepts of immediate selection as well as newly derived conditions like symmetric triples have been incorporated. Lower bounds have been provided by solving linear programs which partially relax the precedence constraints and allow preemptions. Additionally, the LP-optimizer provided information which has been used for branching. To avoid storage problems when solving problems with more than 30 activities column-generation techniques have been implemented for solving the LP-relaxations. Initial upper bounds which are calculated by a tabu-search procedure contribute to the very good performance of the branch and bound algorithm.

Computational results have been presented for PROGEN-instances with 30 and 60 activities. Three hundred and twenty six of the 480 test problems with 60 activities have been solved within the time-limit of 1 h. Similar good results are derived for a first set of PROGEN-instances provided by Kolisch each with 90 activities. Storage requirement for the versions which use column-generations techniques did not exceed 10 MB.

There is still room for further improvements. The computational tests show that the algorithm does not perform well on problems with small capacity factor (up to 0.5). To improve the performance for such problems better lower bounds should be derived. One way to improve the lower bounds is to incorporate heads and tails into the LP-formulation. Along with these changes other branching strategies could be explored.

Acknowledgements

The authors are grateful for the fertile discussions of Andreas Krämer with the third author. They also thank the referees and Johann Hurink for their constructive comments.

References

Baar, T., Brucker, P., Knust, S., 1997. Tabu-search algorithms for the resource-constrained project scheduling problem, Osnabrücker Schriften zur Mathematik, Reihe P, Heft 192.

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

Bell, C.A., Park, K., 1990. Solving resource-constrained project scheduling problems by A^* search. Naval Research Logistics 37, 61–84.

Blażewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G., 1983. Scheduling projects subject to resource constraints: Classification and complexity. Discrete Applied Mathematics 5, 11–24.

Brucker, P., 1995. Scheduling algorithms, Springer, Berlin.

Brucker, P., Jurisch, B., Krämer, A., 1994a. The job-shop problem and immediate selection. Annals of Operations Research 50, 73–114.

Brucker, P., Schoo, A., Thiele, O., 1996. A branch and bound algorithm for the resource-constrained project scheduling problem, Osnabrücker Schriften zur Mathematik, Reihe P, Heft 178.

Carlier, J., Latapie, B., 1991. Une méthode arborescente pour résoudre les problèmes cumulatifs. RAIRO 25, 311–340.

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

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

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

Demeulemeester, E., Herroelen, W., 1995. New benchmark results for the resource-constrained project scheduling problem, Paper presented at the INFORMS Singapore International 1995 Meeting.

Demeulemeester, E., Herroelen, W., Simpson, W., Baroum, S., Patterson, J.H., Yang, K.K., 1994. On a paper by Christofides et al. for solving the multiple-resource constrained, simple project scheduling problem. European Journal of Operational Research 76, 218–228.

Gilmore, P.C., Gomory, R.E., 1961. A linear programming approach to the cutting-stock problem, OR 9, 849–859.

Golumbic, M.C., 1980. Algorithmic graph theory and perfect graphs, Academic Press, New York.

Johnson, R., 1988. Optimally balancing large assembly lines with FABLE, Management Science 34 (2) 240–253.

Kolisch, R., Sprecher, A., 1997. PSPLIB – A project scheduling problem library. European Journal of Operational Research 96, 205–216.

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

Korte, N., Möhring, R.H., 1985. Transitive orientation of graphs with side constraints. In: Noltemeier, H. (Ed.), Proceedings 11th International Workshop on Graph Theoretic Concepts in Computer Science WG '85, Trauner Verlag, Linz 1985, 143–160.

Krämer, A., 1995. Scheduling multiprocessor tasks on dedicated processors, Dissertation, Fachbereich Mathematik/Informatik, Universität Osnabrück.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L., 1994. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation, Technical report no. 32, Department of Mathematics, University of Bologna, Italy.

Möhring, R.H., 1985. Algorithmic aspects of comparability graphs and interval graphs. In: Rival, I. (Ed.), Graphs and Order: The Role of Graphs in the Theory of Ordered Sets and its Applications, NATO Advanced Science Institute Series, Series C., Mathematical and Physical Sciences, 41–101.

Papadimitriou, C.H., Steiglitz, K., 1982. Combinatorial optimization: Algorithms and complexity, Prentice-Hall, Englewood Cliffs, New Jersey.

Schoo, A., 1996. Untere Schranken und Immediate Selection für das Resource-Constrained-Project Scheduling Problem, Diplomarbeit, Fachbereich Mathematik/Informatik, Universität Osnabrück.

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