11-1998

A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations

Bert DE REYCK
Singapore Management University, bdreyck@smu.edu.sg

Willy HERROELEN

Citation

DE REYCK, Bert and HERROELEN, Willy. A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. (1998). *European Journal of Operational Research*. 111, (1), 152-174.
Available at: https://ink.library.smu.edu.sg/lkcsb_research/6742

ELSEVIER

European Journal of Operational Research 111 (1998) 152–174

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations

Bert De Reyck, Willy Herroelen *

Department of Applied Economics, Katholieke Universiteit Leuven, Naamsestraat 69, 3000 Leuven, Belgium

Received 1 October 1996; accepted 1 July 1997

Abstract

We present an optimal solution procedure for the resource-constrained project scheduling problem (RCPSP) with generalized precedence relations (RCPSP-GPR) with the objective of minimizing the project makespan. The RCPSP-GPR extends the RCPSP to arbitrary minimal and maximal time lags between the starting and completion times of activities. The proposed procedure is suited for solving a general class of project scheduling problems and allows for arbitrary precedence constraints, activity ready times and deadlines, multiple renewable resource constraints with time-varying resource requirements and availabilities, several types of permissible and mandatory activity overlaps and multiple projects. It can be extended to other regular and non-regular measures of performance. Essentially, the procedure is a depth-first branch-and-bound algorithm in which the nodes in the search tree represent the original project network extended with extra precedence relations to resolve a number of resource conflicts. These conflicts are resolved using the concept of minimal delaying modes, which is an extension of the notion of minimal delaying alternatives for the RCPSP. Several bounds and dominance rules are used to fathom large portions of the search tree. Extensive computational experience is reported.

Keywords: Project management; Critical path methods; Branch-and-bound; Generalized precedence relations

1. Introduction

The Critical Path Method (CPM) (Kelley and Walker, 1959) and Program Evaluation and Review Technique (PERT) (Malcolm et al., 1959) are devoted to minimizing the makespan of a project under the assumption that required resources are available in sufficient amounts and that the technological precedence relations between any pair of activities i and j imply that activity i must be completed before activity j can be initiated. Over the years, the assumption of sufficiently available resources has been relaxed and many research efforts have been directed towards project scheduling with explicit consideration of resource requirements and constraints. More recent research has been directed at relaxing the strict precedence assumption of CPM/PERT. In accordance

* Corresponding author. Fax: +32 16 326 732; e-mail: willy.herroelen@econ.kuleuven.ac.be.

0377-2217/98/$19.00
PII S0377-2217(97)00305-6

with Elmaghraby and Kamburowski (1992), we will refer to the resulting types of precedence relations as generalized precedence relations (GPRs). We distinguish between four types of GPRs: start–start (SS), start–finish (SF), finish–start (FS) and finish–finish (FF). The resulting types of precedence relations are also often referred to as Metra Potential Method (MPM) precedence constraints (Kerbosh and Schell, 1975; Zhan, 1994), precedence diagramming relations (Moder et al., 1983), time windows (Bartusch et al., 1988), minimal and maximal time lags (Neumann and Schwindt, 1997; Neumann and Zhan, 1995; Brinkmann and Neumann, 1996; Schwindt, 1996a), and generalized precedence constraints (Wikum et al., 1994).

GPRs can specify a minimal or a maximal time lag between a pair of activities. A minimal time lag specifies that an activity can only start (finish) when the predecessor activity has already started (finished) for a certain time period. A maximal time lag specifies that an activity should be started (finished) at the latest a certain number of time periods beyond the start (finish) of another activity. GPRs can be used to model a wide variety of specific problem characteristics, including (Bartusch et al., 1988; De Reyck, 1995; Neumann and Schwindt, 1997) activity ready times and deadlines, activities that have to start or terminate simultaneously, non-delay execution of activities, several types of mandatory activity overlaps, fixed activity starting times, time-varying resource requirements and availabilities, time-windows for resources, inventory restrictions, set-up times, overlapping production activities (process batches, transfer batches) and assembly line zoning constraints. The first treatment of GPRs is due to Kerbosh and Schell (1975), based on the pioneering work of Roy (1962). Other studies include Crandall (1973), Elmaghraby (1977), Wiest (1981), Moder et al. (1983), Bartusch et al. (1988), Elmaghraby and Kamburowski (1992), Zhan (1994), De Reyck (1995), Neumann and Schwindt (1997), Neumann and Zhan (1995), Schwindt (1996a), Brinkmann and Neumann (1996), De Reyck and Herroelen (1996a, b), Schwindt and Neumann (1996) and Franck and Neumann (1996).

In this paper, we present an optimal solution procedure for the resource-constrained project scheduling problem with generalized precedence relations (further denoted as RCPSP-GPR). To the best of our knowledge, the only optimal solution procedure presented in the literature for the RCPSP-GPR is the branch-and-bound algorithm of Bartusch et al. (1988). Heuristics have been presented by Zhan (1994), Neumann and Zhan (1995), Brinkmann and Neumann (1996), Franck and Neumann (1996) and Schwindt and Neumann (1996).

The remainder of this paper is organized as follows. Section 2 elaborates on the concept of GPRs and clarifies the terminology and the project representation used. Section 3 continues with the temporal analysis of activity networks with GPRs. In Section 4, which discusses the resource analysis of such networks, a branch-and-bound procedure for the RCPSP-GPR is presented. Computational results are given in Section 5. Section 6 is reserved for our overall conclusions.

2. Generalized precedence relations

Assume a project represented in activity-on-the-node format by a directed graph $G = \{V, E\}$ in which V is the set of vertices or activities, and E is the set of edges or GPRs. The non-preemptable activities are numbered from 1 to n, where the dummy activities 1 and n mark the beginning and the end of the project. The duration of an activity is given by $d_i (1 \leqslant i \leqslant n)$, its starting time by $s_i (1 \leqslant i \leqslant n)$ and its finishing time by $f_i (1 \leqslant i \leqslant n)$. There are m *renewable* resource types, with $r_{ikx} (1 \leqslant i \leqslant n, 1 \leqslant k \leqslant m, 1 \leqslant x \leqslant d_i)$ the resource requirements of activity i with respect to resource type k in the xth period it is in progress and $a_{kt} (1 \leqslant k \leqslant m; 1 \leqslant t \leqslant T)$ the availability of resource type k in time period $]t-1, t]$ (T is an upper bound on the project length). If the resource requirements and availabilities are not time dependent, they are represented by $r_{ik} (1 \leqslant i \leqslant n, \quad 1 \leqslant k \leqslant m)$ and $a_k (1 \leqslant k \leqslant m)$ respectively. The minimal and maximal time lags between two activities i and j have the form:

$$s_i + \mathrm{SS}_{ij}^{\min} \leqslant s_j \leqslant s_i + \mathrm{SS}_{ij}^{\max},$$

$$s_i + \mathrm{SF}_{ij}^{\min} \leqslant f_j \leqslant s_i + \mathrm{SF}_{ij}^{\max},$$

$$f_i + \mathrm{FS}_{ij}^{\min} \leqslant s_j \leqslant f_i + \mathrm{FS}_{ij}^{\max},$$

$$f_i + \mathrm{FF}_{ij}^{\min} \leqslant f_j \leqslant f_i + \mathrm{FF}_{ij}^{\max},$$

where $\mathrm{SS}_{ij}^{\min}$ represents a minimal time lag between the start time of activity i and the start time of activity j (similar definitions apply for $\mathrm{SS}_{ij}^{\max}$, $\mathrm{FS}_{ij}^{\min}, \ldots$). The various time lags can be represented in a *standardized form* by transforming them to, for instance, minimal SS precedence relations, using the following transformation rules (Bartusch et al., 1988):

$$s_i + \mathrm{SS}_{ij}^{\min} \leqslant s_j \Rightarrow s_i + l_{ij} \leqslant s_j \quad \text{with} \quad l_{ij} = \mathrm{SS}_{ij}^{\min},$$

$$s_i + \mathrm{SS}_{ij}^{\max} \geqslant s_j \Rightarrow s_j + l_{ji} \leqslant s_i \quad \text{with} \quad l_{ji} = -\mathrm{SS}_{ij}^{\max},$$

$$s_i + \mathrm{SF}_{ij}^{\min} \leqslant f_j \Rightarrow s_i + l_{ij} \leqslant s_j \quad \text{with} \quad l_{ij} = \mathrm{SF}_{ij}^{\min} - d_j,$$

$$s_i + \mathrm{SF}_{ij}^{\max} \geqslant f_j \Rightarrow s_j + l_{ji} \leqslant s_i \quad \text{with} \quad l_{ji} = d_j - \mathrm{SF}_{ij}^{\max},$$

$$f_i + \mathrm{FS}_{ij}^{\min} \leqslant s_j \Rightarrow s_i + l_{ij} \leqslant s_j \quad \text{with} \quad l_{ij} = d_i + \mathrm{FS}_{ij}^{\min},$$

$$f_i + \mathrm{FS}_{ij}^{\max} \geq s_j \Rightarrow s_j + l_{ji} \leqslant s_i \quad \text{with} \quad l_{ji} = -d_i - \mathrm{FS}_{ij}^{\max},$$

$$f_i + \mathrm{FF}_{ij}^{\min} \leqslant f_j \Rightarrow s_i + l_{ij} \leqslant s_j \quad \text{with} \quad l_{ij} = d_i - d_j + \mathrm{FF}_{ij}^{\min},$$

$$f_i + \mathrm{FF}_{ij}^{\max} \geqslant f_j \Rightarrow s_j + l_{ji} \leqslant s_i \quad \text{with} \quad l_{ji} = d_j - d_i - \mathrm{FF}_{ij}^{\max}.$$

In this way, all GPRs are consolidated in the expression $s_i + l_{ij} \leqslant s_j$, where l_{ij} denotes a minimal SS time lag. If there is more than one time lag l_{ij} between two activities i and j, only the maximum time lag is retained. The interval $[s_i + l_{ij}, s_i - l_{ji}]$ is called the *time window* of s_j relative to s_i (a similar definition can be found in Bartusch et al., 1988). Applying these transformation rules to an activity network with GPRs results in a so-called *constraint digraph*, which is short for *digraph of temporal constraints*.

A path $\langle i_s, i_k, i_l, \ldots, i_t \rangle$ is called a *cycle* if $s = t$. With 'path' we mean a *directed* path, and with 'cycle' we mean a *directed* cycle. The *length* of a path (cycle) is defined as the sum of the lags associated with the arcs belonging to that path (cycle). To ensure that the dummy start and finish activities correspond to the beginning and the completion of the project, we assume that there exists at least one path with non-negative length from node 1 to every other node and at least one path from every node i to node n which is equal to or larger than d_i. If there are no such paths, we can insert arcs $(1,i)$ or (i,n) with weight zero or d_i respectively. $P(i) = \{j \mid (j,i) \in E\}$ is the set of all *immediate predecessors* of node i, $Q(i) = \{j \mid (i,j) \in E\}$ is the set of all its *immediate successors*. If there exists a path from i to j, then we call i a *predecessor* of j and j a *successor* of i. $P^*(i)$ and $Q^*(i)$ denote the set of (not necessarily *immediate*) predecessors and successors of node i respectively. If the length of the longest path from i to j is non-negative, i is called a *real* predecessor of j, and j is called a *real* successor of i. Otherwise it is a *fictitious* one. (These definitions differ slightly from the ones used by Zhan (1994) and Neumann and Zhan (1995).)

3. Temporal analysis

A schedule $S = (s_1, s_2, \ldots, s_n)$ is called *time feasible*, if the activity starting times satisfy all GPRs, i.e. if they satisfy the following conditions:

$$s_i \geqslant 0 \quad \forall i \in V, \tag{1}$$

$$s_i + l_{ij} \leqslant s_j \quad \forall (i,j) \in E, \tag{2}$$

where Eq. (1) ensures that no activity starts before the current time (time zero), and Eq. (2) denotes the GPRs in standardized form. The minimum starting times $(s_1, s_2, \ldots, s_n)$ satisfying both Eqs. (1) and (2) form the *early start schedule* $\mathrm{ESS} = (\mathrm{es}_1, \mathrm{es}_2, \ldots, \mathrm{es}_n)$ associated with the temporal constraints. The calculation of an ESS can be related to the test for existence of a time-feasible schedule. The earliest start of an activity i can be

calculated by finding the longest path from node 1 to node i. We also know that there exists a time-feasible schedule for G iff G has no cycle of positive length (Bartusch et al., 1988). Such cycles would unable us to compute activity starting times which satisfy Eqs. (1) and (2). Therefore if we calculate the matrix $D=[d_{ij}]$, where d_{ij} denotes the longest path length from node i to node j, a positive path length from any node i to itself indicates the existence of a cycle of positive length and, consequently, the non-existence of a time-feasible schedule. In the literature (Bartusch et al., 1988), the matrix D is often referred to as the *distance* matrix, with d_{ij} the maximal distance between activities i and j. We prefer the term *longest path* instead of distance, although the same notation D and d_{ij} is used. The calculation of D can be done by standard graph algorithms for longest paths in networks, for instance by the Floyd–Warshall algorithm (see Lawler, 1976). If we start with the matrix $D^{(1)}=[d_{ij}^{(1)}]$ with

$$d_{ij}^{(1)}=\begin{cases}0 & \text{if } i=j,\\ l_{ij} & \forall (i,j)\in E,\\ -\infty & \text{otherwise,}\end{cases}$$

we can compute $D=D^{(n+1)}$ according to the updating formula $d_{ij}^{(v)}=\max\{d_{ij}^{(v-1)}, d_{il}^{(v-1)}+d_{lj}^{(v-1)}\}$ $(i,j,l=1,2,\ldots,n)$. If $d_{ii}=0$ for all $i=1,2,\ldots,n$, there exists a time-feasible schedule. The ESS is given by the numbers in the upper row of D: $\text{ESS}=(d_{11}, d_{12},\ldots, d_{1n})$. Computing D takes $O(n^3)$ time.

4. Resource analysis

The RCPSP-GPR can be conceptually formulated as follows:

$$\text{minimize} \quad s_n \tag{3}$$

$$\text{subject to} \quad s_i + l_{ij} \leqslant s_j \quad \forall (i,j)\in E, \tag{4}$$

$$\sum_{i\in S(t)} r_{ik} \leqslant a_{kt} \quad k=1,2,\ldots,m, \quad t=1,2,\ldots,T, \tag{5}$$

$$s_1 = 0, \tag{6}$$

$$s_i \in \mathbb{N}, \quad i=1,2,\ldots,n, \tag{7}$$

where $\mathbb{N}$ denotes the set of natural numbers, $S(t)$ is the set of activities in progress in time period $]t-1,t]$ and T is an upper bound on the project duration, for instance $T=\sum_{i\in V}\max\{d_i, \max_{j\in Q(i)}\{l_{ij}\}\}$.

Note that it is not always possible to derive a feasible solution. The upper bound T indicates the maximal value for the project makespan if a feasible solution exists. The objective function given in Eq. (3) minimizes the project duration, given by the starting time (or completion time, since $d_n=0$) of the dummy activity n. The GPRs are denoted in standardized form by Eq. (4). Eq. (5) represents the resource constraints. The resource requirements and availabilities are assumed to be constant over time, although this assumption can easily be relaxed using GPRs without having to change the solution procedures. Time-varying resource requirements can be modelled by splitting up the activities in a number of subactivities with a different *constant* resource requirement for each of the resource types. The subactivities should then be connected with minimal and maximal zero-lag FS precedence relations which ensure a non-delay execution of all the subactivities of each activity. Time-varying resource availabilities can be handled by creating dummy activities which absorb a certain amount of each resource type for which a constant availability (equal to the maximum availability over time of that resource type) can then be assumed. These dummy activities should then be assigned a fixed starting time using a minimal and maximal time lag between dummy activity 1 and the dummy activity in question. Naturally, allowing for time-varying resource requirements and availabilities will undoubtedly lead to an increase in the complexity of the RCPSP-GPR. Eq. (6) forces the dummy start activity to begin at time zero and Eq. (7) ensures that the activities starting time assume non-negative integer values. Once started, activities run to completion. However, this non-preemption condition can easily be relaxed by splitting up the activities in unit-duration *subactivities* (Demeulemeester and Herroelen, 1996),

connected with minimal zero-lag FS precedence relations. Again, allowing for activity preemption will substantially complicate the RCPSP-GPR.

The RCPSP-GPR is known to be strongly NP-hard, and even the decision problem of testing whether an RCPSP-GPR instance has a feasible solution is NP-complete (Bartusch et al., 1988). To the best of our knowledge, the only optimal solution procedure presented in the literature for the RCPSP-GPR is the branch-and-bound algorithm of Bartusch et al. (1988). However, the computational experience obtained with this procedure is limited to a single bridge construction project and the computer code is no longer available (Möhring, 1996). Neumann and Zhan (1995) developed a priority-rule-based heuristic which allows us to solve RCPSP-GPR instances using a parallel search scheme (see also Zhan, 1994). Brinkmann and Neumann (1996) developed a serial heuristic for the RCPSP-GPR (called DIRECT) and a heuristic based on the (serial) scheduling of cycle structures (strongly connected components of a project network with GPRs) and the subsequent (serial) scheduling of the (acyclic) contracted project network (called CONTRACT). Franck and Neumann (1996) further enhanced the approach of Neumann and Zhan (1995) and validated the performance of the heuristic procedures. Schwindt and Neumann (1996) report computational experience with a branch-and-bound-based heuristic which first schedules all cycle structures in an RCPSP-GPR instance using the procedure described in this paper and subsequently solves the contracted acyclic project network using an extended version of the RCPSP procedure of Demeulemeester and Herroelen, (1992, 1997a). In the following section, we discuss the fundamentals of the new branch-and-bound procedure for the RCPSP-GPR.

4.1. The search tree

The nodes in the search tree represent the initial project network, described by the matrix $D=[d_{ij}]$, extended with extra zero-lag FS precedence relations to resolve a number of resource conflicts, which results in an *extended* matrix $D'=[d'_{ij}]$. Nodes which represent time-feasible but resource-infeasible project networks and which are not fathomed by any of the node fathoming rules described below lead to a new branching. Therefore each (undominated) node represents a time-feasible, but not necessarily resource-feasible project network. Resource conflicts are resolved using the concept of *minimal delaying alternatives*, i.e. minimal sets of activities which, when delayed, release enough resources to resolve the resource conflict and which do not contain any other delaying alternative as a subset. Each of these minimal delaying alternatives is delayed (enforced by extra zero-lag FS precedence relations $i \prec j$, implying $s_i + d_i \leqslant s_j$) by each of the remaining activities also belonging to the *conflict set* $S(t^*)$, the set of activities in progress in period $]t^*-1, t^*]$ (the period of the *first* resource conflict). Therefore, each minimal delaying alternative can give rise to several *minimal delaying modes*.

A similar delaying strategy was used by Demeulemeester and Herroelen (1992) for the RCPSP. As the RCPSP can be solved using semi-active timetabling (i.e. schedule activities as early as possible within the precedence and resource constraints) to construct partial schedules, activities belonging to the minimal delaying alternative can be delayed by the activity in $S(t^*)$ which terminates at the earliest time instant after the current decision point (further denoted as the *delaying activity*). In the RCPSP-GPR, this delaying strategy cannot be used because of the GPRs, which make it impossible to determine which activity in $S(t^*)$ should be used as the delaying activity, because we cannot predict in advance which activity in $S(t^*)$ will terminate the earliest in the feasible schedules that will be obtained by branching from the current project network. Demeulemeester and Herroelen (1997b) devised an adaptation of their RCPSP solution strategy to cope with problems in which only minimal time lags ($l_{ij} \geqslant 0, \forall i,j \in V$) are present. When minimal *and maximal* time lags are allowed, however, even the constructs used by these authors can no longer be used. A similar situation occurs when maximizing the net present value of a resource-constrained project network (Icmeli and Erengüç, 1996). In the RCPSP-GPR, we have to consider several possible *delaying*

modes for each delaying alternative, possibly one for each activity in $S(t^*)$ which is not an element of the delaying alternative.

Assume, for example, that in a certain period $]t^* - 1, t^*]$, four activities are in progress and cause a resource conflict: $S(t^*) = \{1,2,3,4\}$. Suppose that the minimal delaying alternatives are $\{1\}$, $\{2\}$ and $\{3, 4\}$, i.e. delaying activity 1, activity 2 or activities 3 and 4 simultaneously releases enough resources to resolve the resource conflict. For the RCPSP, the procedure of Demeulemeester and Herroelen (1992) would create three new nodes. In the first node, activity 1 is delayed by the earliest finishing activity (x) among activities 2, 3 and 4 ($x \prec 1$). In the second node, activity 2 is delayed by the earliest finishing activity (y) among activities 1, 3 and 4 ($y \prec 2$). Finally, activities 3 and 4 are delayed by activity 1 or 2 (z), depending on which activity finishes the earliest ($z \prec 3$ and $z \prec 4$). This results in three new nodes, as illustrated in Fig. 1.

For the RCPSP-GPR, the delay of activity 1 is established by adding a precedence relation between activities 2, 3 and 4 and activity 1. We therefore create three new nodes (instead of one), one with the precedence relation $2 \prec 1$, one with the precedence relation $3 \prec 1$ and one with the precedence relation $4 \prec 1$. Delaying activity 2 is accomplished by creating three new nodes with the extra precedence relations $1 \prec 2$, $3 \prec 2$ and $4 \prec 2$. Delaying activities 3 and 4 is accomplished by creating two new nodes with the extra precedence relations $1 \prec 3$ and $1 \prec 4$, and $2 \prec 3$ and $2 \prec 4$ respectively. In total, eight new nodes (minimal delaying modes) are created, as illustrated in Fig. 2.

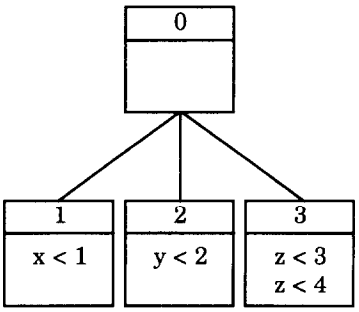

Fig. 1. Delaying strategy for the RCPSP of Demeulemeester and Herroelen (1992).

In general, the *delaying set* DS, i.e. the set of all minimal delaying alternatives, is equal to $\mathrm{DS} = \{D_d | D_d \subset S(t^*)$ and $\forall$ resource type k: $\sum_{i \in S(t^*)} r_{ik} - \sum_{i \in D_d} r_{ik} \leqslant a_k$ and $\forall\, D_{d'} \in \mathrm{DS} \setminus \{D_d\}$: $D_{d'} \not\subset D_d\}$. The set of minimal delaying modes equals: $M = \{M_m \mid M_m = \{k \prec D_d\},\ k \in S(t^*) \setminus D_d,\ D_d \in DS\}$. Activity k is called the *delaying activity*: $k \prec D_d$ implies that $k \prec l$ for all $l \in D_d$.

Theorem 1. *The delaying strategy which consists of delaying all minimal delaying alternatives D_d by each activity $k \in S(t^*) \setminus D_d$ will lead to the optimal solution of the RCPSP-GPR in a finite number of steps.*

Proof. See Appendix A.

Each minimal delaying mode is then examined for time-feasibility and evaluated by computing the critical path-based lower bound lb_0 (equal to the makespan of the ESS). Each time-feasible minimal delaying mode with a lower bound $\mathrm{lb}_0 < T$ is then considered for further branching, which occurs from the node with smallest lb_0. If the node represents a project network in which a resource conflict occurs, a new branching occurs. If it represents a feasible schedule, the upper bound T is updated and the procedure backtracks to the previous level in the search tree. Therefore, we have a depth-first search procedure, in which branching occurs until at a certain level in the tree, there are no delaying modes left to branch from. Then, the procedure backtracks to the previous level in the search tree and reconsiders the other delaying modes (not yet branched from) at that level. The procedure stops when it backtracks to level 0.

4.2. Node fathoming rules

Nodes are fathomed when they represent a time-infeasible project network or when lb_0 exceeds (or equals) T. Nodes which are not fathomed and still represent an infeasible project network are

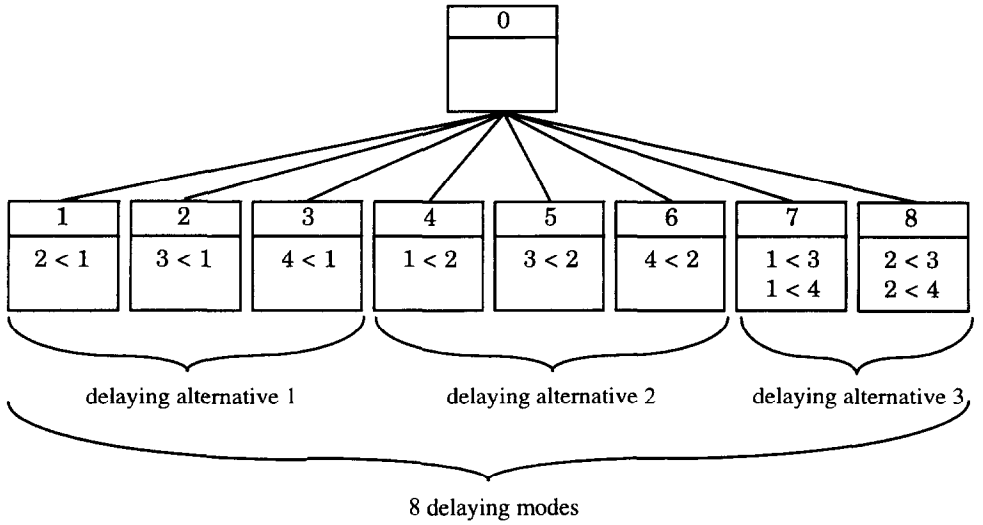

Fig. 2. Delaying strategy for the RCPSP-GPR.

considered for further branching. Four additional node fathoming rules (three dominance rules and a new lower bound) and a procedure which reduces the solution space and which can be executed as a preprocessing rule are added.

4.2.1. Redundant delaying alternatives

Because activity overlaps are allowed ($d_{ij} < d_i$), it is possible that in period $]t^* - 1, t^*]$ (the period of the first resource conflict), the set of activities in progress (the conflict set $S(t^*)$) contains an activity i together with a *real* successor j of activity i ($d_{ij} \geqslant 0$). Then, delaying activity i will also delay activity j. Consequently, the following theorem applies.

Theorem 2. *If there exists a minimal delaying alternative D_d with activity $i \in D_d$ but its real successor $j \notin D_d$ ($d_{ij} \geqslant 0$), we can extend D_d with activity j. If the resulting delaying alternative becomes non-minimal as a result of this operation, it may be eliminated from further consideration.*

Proof. Obvious.

4.2.2. Redundant delaying modes

Because of activity overlaps, it is possible that a certain minimal delaying alternative D_d gives rise to two delaying modes M_{m_1} and M_{m_2}, in which the delaying activities i ($i \prec D_d$) and j ($j \prec D_d$) are precedence related. Then, the following theorem applies.

Theorem 3. *When a minimal delaying alternative D_d gives rise to two delaying modes M_{m_1} and M_{m_2} with delaying activities i and j respectively, mode M_{m_2} is dominated by mode M_{m_1} iff $d_{ij} + d_j \geqslant d_i$.*

Proof. Obvious.

4.2.3. A time- and resource-based lower bound

Recently, Mingozzi et al. (1997) have developed five new lower bounds, lb_1, lb_2, lb_p, lb_x and lb_3, derived from different relaxations of a new mathematical formulation for the RCPSP. Bounds lb_1, lb_2, lb_x and lb_3 dominate the critical path-based lower bound lb_0 and all prove to be tighter than the critical sequence lower bound lb_s of Stinson et al. (1978) on the 110 RCPSP instances assembled by Patterson (1984) and the 480 randomly generated RCPSP instances of Kolisch et al. (1995). Mingozzi et al. (1997) compute lb_3 using a heuristic for the weighted node packing problem. Demeulemeester and Herroelen (1997a) have incorporated another version of lb_3 (further denoted as lb_3') in their procedure for the RCPSP. lb_3'

proved to be more powerful than lb_3 (when used in combination with lb_0), mainly because of its ease of computation. For each activity $i \in V$, its possible *companions*, i.e. the activities with which it can be scheduled in parallel, respecting both the precedence and resource constraints, are determined. All (unscheduled) activities i are then entered in a list L in non-decreasing order of the number of companions (non-increasing duration as tie-breaker). The following procedure then yields a lower bound, lb_3' (for the partial schedule under consideration):

$lb_3' = 0$ (or the earliest completion time of the activities in progress if a partial schedule is already determined)
while L not empty **do**
  take the first activity (activity i) in L
  $lb_3' = lb_3' + d_i$
  remove activity i and its companions from L
enddo

Computational results obtained by Demeulemeester and Herroelen (1997a) indicate that lb_3' indeed outperforms lb_0 and that incorporating lb_3' in their branch-and-bound procedure reduces the computational effort to solve the 110 problems of Patterson (1984) and the 480 problems of Kolisch et al. (1995). The procedure of Demeulemeester and Herroelen (1997a) for computing lb_3' can be extended to the RCPSP-GPR, by changing the calculation of the companions of the activities. In the RCPSP-GPR, activities i and j are companions if the resource requirements of both activities do not exceed the resource availability for any resource type, and if both $d_{ij} < d_i$ and $d_{ji} < d_j$.

In our implementation of lb_3, we have also adapted the (weighted node packing) heuristic. Instead of removing an activity j from the list L when a companion i is taken from the list, we only remove part of activity j from the list. The logic behind this reasoning relies on both a *duration* and *time lag argument*. The *duration argument* goes as follows: When an activity i is scheduled, a companion j can be scheduled in parallel with i. However, if $d_i < d_j$, only a part of activity j can be scheduled in parallel with i. Therefore, a part of activity j (with remaining duration $d_j^r = d_j - d_i$) can be left in L. Initially, all d_j^r are equal to d_j. The *time lag argument* goes as follows. We adjust the part of activity j which has to be removed when a companion i is taken from the list, by incorporating the precedence relations between i and j. Several different situations have to be considered when deciding how much to remove from an activity in L.

Table 1 shows the appropriate action (namely how much to remove from activity j when activity i is taken from L) depending on the longest path lengths between activities i and j. When we want to remove x units from activity j whereas only y ($y < x$) units are left, activity j is to be removed completely from L.

Because in the procedure for the RCPSP-GPR, time-infeasibilities will be detected before lb_3 is calculated, the calculation of lb_3 for the RCPSP-GPR (lb_3^g) can be summarized as follows:

$lb_3^g = 0$
Set all d_i^r equal to d_i
while L not empty **do**
  take the first activity (activity i) in L and remove it from L
  $lb_3^g = lb_3^g + d_i^r$
  for every companion j of i **do**
    if $d_{ij} > 0$ **then** $d_j^r = d_j^r - (d_i - d_{ij})$
    else if $d_{ji} > 0$ **then** $d_j^r = d_j^r - \min\{d_j - d_{ji}, d_i\}$
    else $d_j^r = d_j^r - d_i$
    endif
  if $d_j^r \leqslant 0$, remove activity j from L
  enddo
enddo

Theorem 4. *lb_3^g is a valid lower bound for the RCPSP-GPR.*

Proof. See Appendix A.

lb_3^g is used to fathom nodes for which $lb_3^g \geqslant T$. However, whereas lb_0 is calculated immediately upon the creation of a node, the calculation of lb_3^g is deferred until a decision has been made to actually branch from that node. The rationale behind this is that (a) lb_3^g is more difficult to compute

Table 1
The calculation of d_j^r

	$d_i \leqslant d_{ij}$	$0 < d_{ij} < d_i$	$-d_j < d_{ij} \leqslant 0$	$d_{ij} \leqslant -d_j$
$d_j \leqslant d_{ji}$	Infeasible	Infeasible	Infeasible	No companions
$0 < d_{ji} < d_j$	Infeasible	Infeasible	$d_j^r = d_j^r - \min\{d_j - d_{ji}, d_i\}$	$d_j^r = d_j^r - \min\{d_j - d_{ji}, d_i\}$
$-d_i < d_{ji} \leqslant 0$	Infeasible	$d_j^r = d_j^r - (d_i - d_{ij})$	$d_j^r = d_j^r - d_i$	$d_j^r = d_j^r - d_i$
$d_{ji} \leqslant -d_i$	No companions	$d_j^r = d_j^r - (d_i - d_{ij})$	$d_j^r = d_j^r - d_i$	$d_j^r = d_j^r - d_i$

than lb_0, and that (b) calculating lb_3^g implies calculating the entire matrix D. Supported by extensive computational tests, we defer the calculation of lb_3^g and D until the node is actually selected for branching. As a result, only lb_0 is used as a branching criterion.

4.2.4. A subset dominance rule

Each node in the search tree represents the initial project network extended with a set of (strict) precedence constraints to resolve resource conflicts. Therefore, it is possible that a certain node represents a project network which has been examined earlier at another node in the search tree. One way of checking whether two nodes represent the same project network is to check the added precedence constraints. Identical sets of precedence constraints lead to identical project networks. Moreover, the following theorem can then be applied.

Theorem 5. *If the set of added precedence constraints which leads to the project network (in the form of an extended matrix D) in node x contains as a subset another set of precedence constraints leading to the project network (extended matrix D') in a previously examined node y in another branch of the search tree, node x can be fathomed.*

Proof. See Appendix A.

This rule only applies when a node is compared to a previously examined node *in another path of the search tree*. This can be enforced by saving the information required during backtracking. The question remains which nodes have to be saved in order to test this rule. When a node x is dominated by a node y, it will also be dominated by a parent node z of y, unless node z is on the same path of the search tree as node x. Therefore, to check whether node x is dominated, we have to save the set of added precedence constraints of the nodes for which the parent node is on the same path as node x (but which themselves are not on the same path as node x).

4.2.5. Reducing the solution space using preprocessing

Before initiating the branch-and-bound procedure, the solution space can be reduced by simultaneously examining the GPRs and the resource requirements. If on the one hand, two activities i and j can never overlap due to the resource constraints, while on the other hand, the GPRs allow for an overlap, then the precedence relations can be tightened to avoid the overlap. This allows us to state the following theorem, which can be executed as a preprocessing rule.

Theorem 6. *If* $\exists i, j \in V$ *and resource type* k *for which* $r_{ik} + r_{jk} > a_k$ *and* $-d_j < d_{ij} < d_i$, *we can set* $l_{ij} = d_i$ *without changing the optimal solution of the RCPSP-GPR.*

Proof. See Appendix A.

4.3. The branch-and-bound algorithm

The detailed algorithmic steps of the proposed branch-and-bound algorithm are described below. The longest path length between two activities i and j is given by $d[p][i][j]$, where p denotes the level in the search tree. For each such level, a matrix $d[p]$ will have to be stored.

Step 1: Initialisation

Let $T = 9999$ be an upper bound on the project duration.
Set the level of the branch-and-bound tree $p = 0$.
Compute the constraint digraph cd (using the rules discussed in Section 2; $O(|E|)$).
Compute $d[0]$ at level 0 using the Floyd–Warshall algorithm ($O(n^3)$).
If the project is not time feasible (i.e. $\exists i \in V: d[0][i][i] > 0$), STOP.
Preprocessing: reduce the solution space by adjusting $d[0]$ ($O(n^2 m)$):
$\forall (i,j) | i,j \in V$ and $\exists$ resource type k: $r_{ik} + r_{jk} > a_k$ and $-d_j < d[0][i][j] > d_i$, set $l_{ij} = d_i$.
Recompute $d[0]$ using the Floyd–Warshall algorithm ($O(n^3)$).
If the project is not time feasible (i.e. $\exists i \in V$: $d[0][i][i] > 0$), STOP.
Compute the critical path-based lower bound $lb_0 = d[0][1][n]$ and go to Step 3.

Step 2: Temporal analysis

Compute $d[p]$, the extended matrix with longest paths at level p as follows ($O(n^2|D_d|)$):
$\forall i, j \in V$: $d[p][i][j] = d[p-1][i][j]$. $\forall i, j \in V$, $l \in D_d$: $d[p][i][j] = \max\{d[p][i][j],\ d[p-1][i][k] + d_k + d[p-1][l][j]\}$, k being the delaying activity.
If $T < 9999$, compute lb_3^g.
If $lb_3^g \geqslant T$, erase the delaying mode and go to Step 6.

Step 3: Resource analysis

Determine the *first* period $]t^* - 1, t^*]$ in which a resource conflict occurs, i.e. for which $\sum_{i \in S(t^*)} r_{ik} > a_k$ for some resource type k. $S(t^*)$, the set of activities in progress in period $]t^*-1, t^*]$, is called the *conflict set*.
If there is no conflict, let $T = d[p][1][n]$, erase all remaining delaying modes at level p and go to Step 7.
Store $d[p]$.

Step 4: Determine minimal delaying alternatives and minimal delaying modes

Increase the branch level of the search tree: $p = p + 1$.
Determine the minimal delaying set, i.e. the set of minimal delaying alternatives:

$$DS = \left\{ D_d | D_d \subset S(t^*) \text{ and } \forall \text{ resource type } k: \sum_{i \in S(t^*)} r_{ik} - \sum_{i \in D_d} r_{ik} \leqslant a_k \text{ and } \forall D_{d'} \in DS \setminus \{D_d\}: D_{d'} \not\subset D_d \right\}$$

Extend all minimal delaying alternatives using Theorem 2 and eliminate all non-minimal delaying alternatives. Determine the set of minimal delaying modes:

$$M = \{M_m | M_m = \{k \prec D_d\},\ k \in S(t^*),\ k \notin D_d,\ D_d \in DS\}.$$

Eliminate all delaying modes satisfying Theorem 3.

Step 5: Evaluate delaying modes

For all delaying modes M_m
{If the precedence constraints cannot be added, i.e. $\exists l \in D_d: k \prec l$ is infeasible, i.e. $d_k > -d[p-1][l][k]$ (k being the delaying activity), continue with the next delaying mode M_m.
Compute lb_0 as follows: Set $lb_0 = d[p-1][1][n]$. $\forall l \in D_d$ and delaying activity k:

$$lb_0 = \max\{lb_0,\ d[p-1][1][k] + d_k + d[p-1][l][n] \mid j \in D_d\}.$$

If $lb_0 \geqslant T$, continue with next delaying mode M_m.

If the set of added precedence constraints of a previously examined node is a subset of the corresponding set of the current node, continue with the next delaying mode M_m.
Set $\text{lb} = \text{lb}_0$.
Temporarily store the delaying mode and its lower bound lb.
}

Step 6: Branching
If no delaying modes are left to branch from at level p, go to Step 7.
Select the delaying mode M_m with the smallest lower bound lb (arbitrary tie-break).
If $lb \geqslant T$, erase all remaining delaying modes at level p and go to STEP 7.
Go to Step 2.

Step 7: Backtracking
Decrease the branch level of the search tree: $p = p - 1$.
If $p \leqslant 0$, STOP with the optimal solution with a makespan of T
(if $T = 9999$, then there exists no feasible solution).
Delete from the stack the information which has been previously saved on level $p + 1$ for dominance testing.
Save the necessary information for node dominance testing on the stack, i.e. the list of added precedence constraints of the node reached upon backtracking.
Erase $d[p]$ and go to Step 6.

5. Computational experience

The procedure has been programmed in Microsoft Visual C++ 2.0 under Windows NT for use on a Digital Venturis Pentium-60 personal computer with 16 Mb of internal memory. The code itself requires 109 Kb of memory, whereas 10 Mb are reserved for the storage of the search tree. Benchmark tests on 550 problems generated from the 110 RCPSP test instances assembled by Patterson (1984) are analyzed in De Reyck and Herroelen (1996a). Based on a full factorial experiment, they reveal that each of the proposed dominance rules and lower bounds leads to an increased performance of the procedure, both in terms of CPU-time as in terms of nodes in the search tree required to solve the problem instances to optimality. Furthermore, the results show that the percentage of maximal precedence relations, their tightness and the percentage of precedence relations that allow for activity overlaps have a significant impact on the computational effort. The higher the number of maximal time lags, the tighter they are and the higher the number of minimal time lags that allow for activity overlaps, the more effective the procedure.

Several tests have been performed on three different problem sets in order to validate the procedure against the serial and parallel heuristics developed by Franck and Neumann (1996). These heuristics improve upon the procedures developed by Neumann and Zhan (1995), Zhan (1994) and Brinkmann and Neumann (1996) and rank as the best currently available. All three data sets have been generated using the random problem generator ProGen/max developed by Schwindt (1996a) based on the problem generator ProGen for the RCPSP developed by Kolisch et al. (1995). ProGen/max uses two generating methods: DIRECT, which directly generates entire projects, and CONTRACT, which first generates cycle structures, upon which the (acyclic) contracted project network is generated. Several control parameters can be specified. The first problem set (Schwindt, 1996b) consists of 1080 instances, 540 generated using the DIRECT method and 540 using the CONTRACT method. The second set (Franck and Neumann, 1996) consists of 1440 problem instances generated using the DIRECT method. We used the DIRECT method to generate a third set of 7200 problem instances which allows for a more extensive testing of the impact of several control parameters.

Table 2
The parameter settings of the three problem sets

Control parameter	Set I	Set II	Set III
Number of activities	100	100	10; 20; 30; 50; 100
Activity durations	[5,15]	[5,15]	[1,10]
Number of resource types	5	[5,8]	[1,5]
Min/max number resources used per activity	1/5	1/8	1/5
Activity resource demand	[1,3]	[1,3]	[1,10]
Resource factor, RF (Pascoe, 1966)	0.50; 0.75; 1.00	0.25; 0.50; 0.75; 1.00	0.25; 0.50; 0.75; 1.00
Resource strength, RS (Kolisch et al., 1995)	0.20; 0.50; 0.70	0.20; 0.50; 0.75	0.25; 0.50; 0.75
Number of initial and terminal activities	[3,7]	[3,7]	[2,4]
Max number of initial/terminal activities [a]	2/2		
Max number of predecessors/successors	5/5	5/5	5/5
Max number of predecessors/successors [a]	3/3		
Order strength, OS (Mastor, 1970)	0.35; 0.50; 0.65	0.35; 0.50; 0.65	0.25; 0.50; 0.75
Order strength, OS [a]	0.50		
% Maximal time lags	[5%,15%]	[5%,15%]; [15%,25%]	0%; 10%; 20%; 30%
Number of cycle structures (Brinkmann and Neumann, 1996)	[2,5]; [6,9]	[2,7]; [8,13]	[0,10]
Min/max number of nodes per cycle structure	2/15	2/15	2/100
Coefficient of cycle structure density (Schwindt, 1996a)	0.3	0.3	0.3
Cycle structure tightness (Schwindt, 1996b)	0.5	0.5	0.5

[a] For the cycle structures (only for the CONTRACT method).

The parameters used to generate the three problem sets are given in Table 2. The indication $[x,y]$ means that the corresponding value is randomly generated in the interval $[x,y]$, whereas x; y; z means that three settings for that parameter were used in a full factorial experiment. For each combination of parameter values, 10 instances have been generated. It should be observed that the parameter settings for the three data sets do not allow for the generation of problem instances which are not resource constrained, a characteristic which is not shared by the 480 RCPSP instances generated by Kolisch et al. (1995). 120 out of those 480 problem instances have a resource strength (Kolisch et al., 1995) equal to 1, and can therefore be solved by simply calculating the ESS.

5.1. Problem set I

Table 3 shows the computational results on problem set I. The branch-and-bound procedure is truncated after a specific amount of running time (1, 10 and 100 s). The results include the number of problems solved to optimality (for which the optimum was found *and verified*), the number of problems for which the optimal solution is obtained (but *not necessarily verified*), the number

Table 3
The results on problem set I

	F&N	1 s	10 s	100 s
Problems solved to optimality	196 (>18%)	543 (>50%)	592 (>54%)	609 (>56%)
Problems for which optimal solution is found	220 (>20%)	578 (>53%)	596 (>55%)	609 (>56%)
Problems for which best known solution is found	378 (35%)	606 (>56%)	652 (>60%)	682 (>63%)
Unsolved problems	21 (<2%)	205 (<19%)	86 (<8%)	68 (<7%)
Average deviation from lb	17.02%	5.99%	9.77%	10.00%
Average deviation from best known solution	7.20%	2.20%	2.54%	2.31%

of problems for which the best known solution is obtained, the number of unsolved problems (for which a feasible solution could not be determined and neither infeasibility of the instance could be proven), the average deviation from a lower bound and the average deviation from the best known solution. For heuristics, verification of optimality is only possible when the obtained solution is equal to a lower bound. Therefore in Table 3, also the number of times the optimal solution is obtained, but not necessarily verified, is given.

The lower bound lb used to compute the deviations, is the maximum of the critical path-based lower bound lb_0, the resource-based lower bound $lb_r = \max_{k=1}^{m}\{\lceil \sum_{i=1}^{n} d_i r_{ik}/a_k \rceil\}$ and lb_3^g (computed in the root node of the search tree after preprocessing). The column labelled F&N in Table 3 contains the results obtained by Franck and Neumann (1996), which are obtained by running a collection of 44 different heuristics which rank among the best currently available. The best known solution referred to in Table 3 is the best of the solutions obtained with various versions of the branch-and-bound algorithm running for up to 1 h per problem and with the heuristic (F&N) solutions, and can therefore be considered as near-optimal.

From Table 3 we can see that, despite the problem size and complexity, the branch-and-bound procedure manages to solve more than 50% of the problems to optimality within 1 s of computation time. However, increasing the allowed computation time from 1 to 100 s leads to an increase of only 12% in the problem instances solved to optimality (from 543 to 606). The average deviation from the best known solution (lower bound) never exceeds 2.54% (10.00%), whereas the F&N heuristics result in an average deviation of 7.20% (17.02%). The increasing average deviation from lb of the solutions obtained with the branch-and-bound algorithm when the time limit is increased is due to the fact that an increasing number of problems is solved. Hence, the results obtained when the algorithm is allowed to run for 1 s include solutions for less hard problems than those obtained with a limit of 10 and 100 s.

Less reassuring, however, is that, especially for small time limits, a relatively large number of problems remain unsolved. The F&N heuristics do a better job on this issue. This inspired us to develop another approach which is based on finding a feasible solution first, rather than going immediately for the optimal solution. In the original procedure, nodes are branched from in non-decreasing order of a lower bound. The rationale behind this (common) branching criterion is that nodes which entail a high chance of finding a very good solution are chosen first, in the hope that other nodes will be dominated by the obtained upper bound. However, when solving the RCPSP-GPR, each node in the search tree does not only contain information on the effect of the added precedence constraints on the best solution that can ever be obtained by branching from that node, but also on the effect of the added precedence constraints on the probability that a *feasible* solution can be obtained by branching from that node. We developed a new branching rule that also incorporates the latter information.

In the branch-and-bound procedure, a node (with a corresponding delaying activity k and delaying alternative D_d) is eliminated (because it can never lead to a feasible solution) if $\exists l \in D_d : k \prec l$ is infeasible, i.e. if $d_k > -d_{lk}$ for some $l \in D_d$. Thus, if $d_k + d_{lk} \leqslant 0$ for each $l \in D_d$ (no positive cycle in the project network), the delaying mode is considered for further branching, and the selection of the delaying mode to branch from is derived from the lower bound. Consider two delaying modes M_1 and M_2, each with one activity in the corresponding delaying alternative, with $d_{k_1} + d_{l_1 k_1} = -1$ for M_1 and $d_{k_2} + d_{l_2 k_2} = -20$ for M_2. Even if the lower bound of M_1 is smaller than the lower bound of M_2, branching from M_2 may be the smartest thing to do since there is a high probability that branching from M_1 will not lead to any feasible solution. The fact that $d_{k_1} + d_{l_1 k_1} = -1$ means that activity k_1, which was delayed by activity l_1, only has 1 time unit of slack within its time window with respect to activity l_1. Thus, when activity k_1 has to be delayed later on in the project, a positive cycle will probably result, leading to time-infeasibility of the corresponding project network. Therefore, if we want to find a feasible solution, it may be better to branch from the node for which the delayed

activities have a relatively high 'slack' in the time windows in which they can be scheduled. This leads to a new branching strategy, namely branching from the node with the highest slack with respect to the maximal time lags, i.e. in which the cycles created by delaying activities, if any, are as negative as possible. This slack value will be referred to as *time window slack* (TWS). If multiple activities are delayed, the minimal TWS value over all the delayed activities is used as the slack of the node, since this is probably the cycle that is going to create feasibility problems if additional activities are to be delayed.

We used this new branching rule in a new version of our algorithm (further denoted as the TWS *branching scheme*). When no feasible solution has been found yet, the procedure branches from the node with the highest TWS value. Upon finding a feasible solution, the branching criterion switches to the lower bound criterion as before. Using this approach, the number of unsolved problems decreases to 27 (<3%), 8 (<1%) and 6 (<1%) for the three time limit settings. Even when using the TWS branching scheme for an extended amount of time (10,000 s), no feasible solution can be obtained for the six remaining problem instances. Actually, for two out of those six instances, infeasibility of the problem can be proven. Therefore, we conjecture that the other four instances are also infeasible. Also the F&N heuristics cannot provide a feasible solution for those problems. The number of problems solved to optimality using the TWS branching scheme does not significantly differ from the original approach (it is even slightly higher). The average deviation from the best known solution (lower bound) increases somewhat, but never exceeds 4.5% (14%), thereby still outperforming the heuristics.

5.2. Problem set II

The results on problem set II are similar to the ones obtained for problem set I. Franck and Neumann (1996) report that, using the CONTRACT approach, a feasible solution was obtained for 1401 of the 1440 problem instances. The average deviation from the lower bound equals 14.3%. The average deviation from the lower bound obtained with the single best heuristic rule equals 16.6%. The authors state that better results can be obtained with the DIRECT approach, however, at the expense of increased computation times due to a possibly large amount of rescheduling steps needed to resolve time-infeasibilities. Table 4 indicates the results obtained with the truncated version of the branch-and-bound procedure. Again, with a time limit of only 1 s, more than 53% of the problem instances can be solved to optimality. The average deviation from the lower bound never exceeds 8.52%. Again, the best known solutions are obtained using different versions of the branch-and-bound algorithm running for up to 1 h per problem. The TWS branching scheme referred to above reduces the number of unsolved problems to 39 (<3%), 15 (±1%) and 15 (±1%) respectively. When running the TWS procedure for 10,000 s, a feasible solution could be obtained for 3 of the 15 remaining instances. For one additional instance, infeasibility could be proven.

5.3. Problem set III

The results obtained on problem set III are given in Table 5. From Table 5, we can observe that more than 77% of the problems can be solved to optimality within 1 s of computation time. If

Table 4
The results on problem set II

	1 s	10 s	100 s
Problems solved to optimality	766 (>53%)	891 (>61%)	910 (>63%)
Unsolved problems	267 (<19%)	66 (<5%)	43 (<3%)
Average deviation from lb	4.61%	8.52%	8.50%
Average deviation from best known solution	1.29%	1.35%	1.06%

Table 5
The results on problem set III

	1 s	10 s	100 s
Problems solved to optimality	5602 (>77%)	6017 (>83%)	6210 (>86%)
Unsolved problems	141 (<2%)	66 (<1%)	55 (<1%)
Average deviation from lb	4.69%	4.57%	4.39%
Avg. deviation from best known solution	0.71%	0.59%	0.33%

100 s of CPU-time are allowed, this percentage increases to 86%. However, as Fig. 3 clearly displays, the number of problems solved to optimality heavily depends on the problem size. For 1 s of computation time, the percentage of problems solved to optimality decreases from 100% for the 10-activity problem instances to 58% for the 100-activity problem instances. Nevertheless, the relatively high number of problems solved to optimality (even for the 100-activity set) seems very promising, and indicates that, even for large problem instances, the use of truncated branch-and-bound procedures should not be discarded. The average deviation from the best known solution (lower bound) never exceeds 0.71% (4.69%). Equipped with the TWS branching rule, only 3 (<0.1%) instances remain unsolved within 100 s. Using a time limit of 10,000 s, a feasible solution for one additional problem instance could be obtained.

We used problem set III in an experiment to test the impact of several problem characteristics on the RCPSP-GPR complexity. Among the measures of the topological structure of an activity network, the *order strength* (OS), was found to be the most powerful measure for explaining the variations in the CPU-time required by the

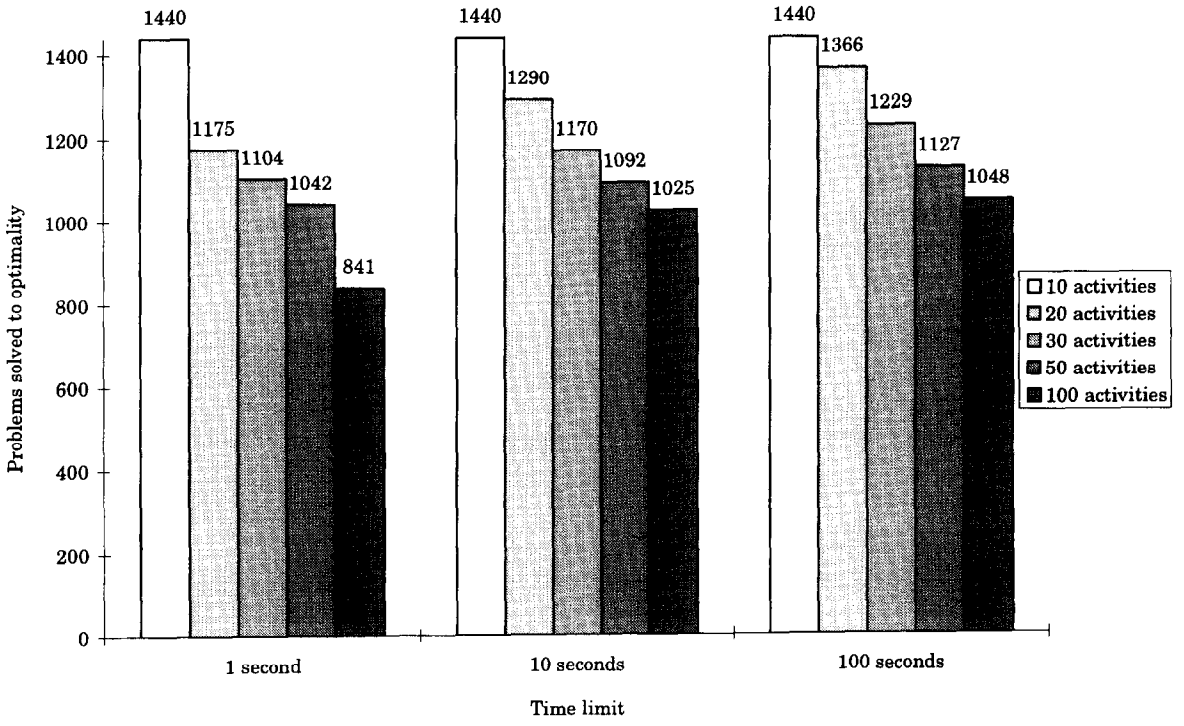

Fig. 3. The effect of problem size on the number of problems solved to optimality.

branch-and-bound procedure for solving different RCPSP-GPR instances. OS is defined as the number of precedence relations, including the transitive ones, divided by the theoretical maximum of such precedence relations, namely $n(n-1)/2$ (Mastor, 1970). In the case of GPRs, OS is defined for the acyclic network including the minimal time lags only (Schwindt, 1996a). OS has a negative impact on the computational complexity of the RCPSP-GPR. When OS increases, the number of problems solved to optimality generally increases, the number of unsolved problems decreases and the average deviations from the best known solutions also decrease. The effect of the percentage of maximal time lags is not monotonically increasing or decreasing. On the contrary, a bell-shaped curve seems to result. When maximal time lags are introduced, the number of problems solved to optimality increases, up to a certain point, beyond which the number of problems solved to optimality again decreases. Similarly, there is a non-linear effect of the percentage of maximal time lags on the number of unsolved problems.

We also tested the impact of two of the best known parameters for describing resource scarcity that have been proposed in the literature. The *resource factor* RF (Pascoe, 1966) reflects the average portion of resources requested per activity. If $RF = 1$, then each activity requests all resources. $RF = 0$ indicates that no activity requests any resource. The *resource strength*, RS, is defined by Kolisch et al. (1995) as $(a_k - r_k^{\min})/(r_k^{\max} - r_k^{\min})$, where a_k is the total availability of renewable resource type k, $r_k^{\max} \max_{i=1,\ldots,n} r_{ik}$ (the maximum resource requirement for each resource type), and $r_k^{\max}$ is the peak demand for resource type k in the ESS. Hence, with respect to one resource the smallest feasible resource availability is obtained for $RS = 0$. For $RS = 1$, the problem is no longer resource constrained.

The higher the RF, the harder the corresponding RCPSP-GPR. The number of problems solved to optimality decreases significantly, the number of unsolved problems increases substantially as does the average deviation from the best known solution. An opposite effect can be observed for RS. When RS increases, the number of problems solved to optimality increases dramatically. Moreover, RS seems to have a stronger impact on the computational complexity of the RCPSP-GPR than does RF. More details of the analysis of the impact of problem characteristics on the RCPSP-GPR complexity can be found in De Reyck and Herroelen (1996b).

6. Conclusions

This paper deals with the RCPSP-GPR with the objective of minimizing the project makespan. The RCPSP-GPR extends the RCPSP to arbitrary minimal and maximal time lags between the activities. This allows us to model a very general class of project scheduling problems including arbitrary precedence constraints, activity ready times and deadlines, multiple resource constraints with time-varying resource requirements and availabilities, activity and resource time windows and several types of mandatory activity overlaps. We presented a depth-first branch-and-bound procedure in which the nodes in the search tree represent the original project network extended with extra precedence relations which resolve a number of resource conflicts. Resource conflicts are resolved using the concept of minimal delaying modes. Several lower bounds and dominance rules are used to fathom large portions of the search tree.

Extensive computational experience obtained on several data sets indicates that all the proposed node fathoming rules lead to a significant reduction in the computation time and the number of nodes in the search tree. The procedure is capable of solving relatively large problem instances with up to 100 activities to optimality in a reasonable amount of time. The use of a truncated version of the procedure outperforms a set of the best heuristics available for the RCPSP-GPR. The procedure can be extended to various regular and non-regular objective functions. A regular objective function (which is to be minimized) is a non-decreasing function of the activity completion times. Consequently, when it is not impeded by precedence or resource constraints, it will not be advantageous to delay activities in order to improve the performance of the schedule. For a non-regular objective function, the condition above does not

hold. This implies that delaying activities may improve the performance of the schedule.

In the procedure, a precedence-based lower bound lb_0 for the project networks in each node of the search tree is calculated by computing the longest path length between the dummy start and the dummy end activity (equal to the makespan of the ESS). If such a node is chosen to branch from and the associated ESS turns out to be feasible, the obtained upper bound is equal to the lower bound lb_0. If we were to optimize any other *regular* measure of performance, we can simply replace the calculation of lb_0 by the regular performance measure under consideration. In other words, we can also use the ESS to compute the objective function value and use it as a lower and/or upper bound. Therefore, only two slight modifications are needed to extend the procedure to other regular measures of performance. First, we need to replace lb_0 by the new measure and use the resulting value as a lower bound and, if applicable, as an upper bound. Second, lb_3^g can no longer be used as a node fathoming rule, since it is based on minimizing the project makespan.

Practical applications of regular measures of performance often take the form of a cost function based on the activity completion times. Such cost functions may take the following form:

- Minimizing total project costs, where the project costs are determined by a weighted function of the tardiness of the activities with respect to pre-set due dates.
- Maximizing the net present value of the project, in which all activities are assigned a *positive* cash flow (a popular assumption under cost-plus contracts; see Herroelen et al., 1997).
- Minimizing the weighted sum of the completion times of several activities, where these activities represent the end of *sub*projects. In fact, we then have a multi-project scheduling problem for which the different projects have been combined to one single *super*project.

If we would optimize a non-regular measure of performance, we *cannot* use the ESS to compute the objective function value. Instead, the project network in each node of the search tree should be optimized using the non-regular objective function, while discarding the resource constraints. A well-known non-regular measure of performance which is gaining more popularity is the maximization of the net present value of the project, in which positive *and/or negative* cash flows are associated with each activity. In this case, the net present value of the project networks in each node should be maximized without taking the resource constraints into account. Algorithms for the unconstrained max-*npv* project scheduling problem can be found in Russell (1970), Grinold (1972), Elmaghraby and Herroelen (1990), Herroelen and Gallens (1993) and Herroelen et al. (1996). However, these algorithms allow only for zero-lag FS precedence constraints. Their extension to GPRs constitutes a viable area of future research.

Acknowledgements

We would like to express our gratitude towards Klaus Neumann and Christoph Schwindt from the University of Karlsruhe for providing us with the project generator ProGen/max and with the computational results for the heuristics that allowed for a comparative analysis. We would also like to thank three anonymous referees for their constructive comments.

Appendix A. Proofs

A.1. Proof of Theorem 1

We prove that (a) the delaying strategy based on (not necessarily *minimal*) delaying modes leads to the optimal solution, (b) it is sufficient to consider only *minimal* delaying modes, and (c) the procedure finds the optimal solution in a finite number of steps.

Lemma A.1. *The delaying strategy based on delaying modes leads to the optimal solution*

Proof. Consider the original project network associated with the *root node* (*node* 0) of the search tree. If this network is time infeasible, no feasible solution can be obtained. If the ESS for

this network has no resource conflict, it is optimal. Therefore, we assume that the project network in the root node of the search tree is time feasible but resource infeasible. Suppose that $S(t^*) = (i_1, i_2, \ldots, i_x)$ is the conflict set of activities in progress in period $]t^* - 1, t^*]$, the period in which the first resource conflict occurs. We can now use the following lemma.

Lemma A.2. *In each feasible solution that may result from resolving a resource conflict created by the conflict set $S(t^*)$, the precedence relation $i_k \prec i_l$ must be satisfied for at least one pair of activities $(k, l) \in S(t^*)$.*

Proof. See Lemma 3.6 in Bartusch et al. (1988).

Therefore, resolving a resource conflict at node 0 by branching into $x(x-1)$ nodes, each of which adds a different precedence constraint $i_k \prec i_l (k, l \in S(t^*))$, guarantees that $\Omega_0 = \bigcup_{k=1}^{x(x-1)} \Omega_k$, in which Ω_k represents the set of feasible solutions that can be obtained when branching from node k. Repeating this branching strategy throughout the search tree leads to the optimal solution.

Our delaying strategy, however, is based on delaying alternatives and delaying modes. Each delaying mode M_m for which the delaying alternative D_d consists of a single activity, corresponds to a precedence relation as specified in Lemma A.1. It remains to be shown that a precedence constraint $\{i_k \prec i_l\}$ imposed by Lemma A.1, which would not be identified by our procedure as a possible delaying mode, is dominated and can be omitted. The reason for our procedure not to generate the delaying mode $\{i_k \prec i_l\}$ can only be that conflict activity i_l does not release enough resources to resolve the resource conflict in period $]t^* - 1, t^*]$. In other words, adding a constraint $\{i_k \prec i_l\}$ at level 1 of the search tree is not enough to resolve the resource conflict in node 0, and, using Lemma A.1, it would be necessary to delay another activity $i_n \neq i_l \in S(t^*)$ by another activity $i_m \notin \{i_l, i_n\}$: $\{i_m \prec i_n\}$ at a higher level of the search tree. Suppose, without loss of generality, that at the second node of the search tree, the decision is made to delay activity $i_n \neq i_l \in S(t^*)$ by activity $i_m \notin \{i_l, i_n\}$: $\{i_m \prec i_n\}$.

Case 1: delaying activity i_n releases enough resources by itself to resolve the resource conflict.

In this case, our procedure would identify the constraint $\{i_m \prec i_n\}$ as a delaying mode at the first level of the search tree. Therefore, the set of feasible solutions that can be obtained by branching from the node on the second level of the search tree is a subset of the set of feasible solutions that can be obtained by our procedure when branching from the node $\{i_m \prec i_n\}$ at level 1 of the search tree.

Case 2: delaying activity i_n does not release enough resources by itself to resolve the conflict.

Case 2.1: $i_m \neq i_k$.

In this case, we delay i_l by i_k and i_n by i_m ($i_l \neq i_n$ and $i_k \neq i_m$).

Case 2.1.1: the resource conflict is resolved.

If the resource conflict is resolved by adding the delaying modes $\{i_k \prec i_l\}$ and $\{i_m \prec i_n\}$ to the project network, the corresponding feasible solution will also be obtained by the delaying mode $\{i_k \prec i_l, i_k \prec i_n\}$ or by delaying mode $\{i_m \prec i_l, i_m \prec i_n\}$. Suppose that in the feasible solution activity i_k finishes before activity i_m (or at the same time). Then this feasible solution will not be eliminated by relaxing the constraint $\{i_m \prec i_n\}$ by $\{i_k \prec i_n\}$. If, on the other hand, activity i_m finishes before activity i_k, then relaxing the constraint $\{i_k \prec i_l\}$ by $\{i_m \prec i_l\}$ will not eliminate this feasible solution. Moreover, the two relaxed problems consisting of delaying modes $\{i_k \prec i_l, i_k \prec i_n\}$ and $\{i_m \prec i_l, i_m \prec i_n\}$ will be identified by our procedure, since $\{i_l, i_n\}$ is a valid delaying alternative. Therefore, all the delaying modes $\{i_k \prec i_l, i_m \prec i_n\}$ imposed by Lemma A.1 are dominated by the two delaying modes $\{i_k \prec i_l, i_k \prec i_n\}$ and $\{i_m \prec i_l, i_m \prec i_n\}$ in our search procedure.

Case 2.1.2: the resource conflict is not resolved.

As the resource conflict is not resolved by the delaying modes $\{i_k \prec i_l\}$ and $\{i_m \prec i_n\}$, new delaying modes have to be added. Eventually, at a certain level of the search tree, a feasible solution will be obtained which was reached by adding a set of extra precedence relations $\{\{i_{k_1} \prec i_{l_1}\}, \{i_{k_2} \prec i_{l_2}\}, \ldots, \{i_{k_q} \prec i_{l_q}\}\}$, for which the delaying activities are not one and the same activity (at least two have to be different, since at level one and two of the search tree, the delaying activities were

different from one another: $i_m \neq i_k$). Then relaxing the constraints $\{i_{k_x} \prec i_{l_x}\}$ by $\{i_K \prec i_{l_x}\}$, in which i_K is the earliest finishing delaying activity in the feasible solution, will not eliminate the feasible solution attained. The corresponding delaying mode $\{\{i_K \prec i_{l_1}\}, \{i_K \prec i_{l_2}\}, \ldots, \{i_K \prec i_{l_q}\}\}$ will also be identified by our procedure on the first level of the search tree, since $\{i_{l_1}, i_{l_2}, \ldots, i_{l_q}\}$ constitutes a valid delaying alternative. Therefore, the delaying modes $\{\{i_{k_1} \prec i_{l_1}\}, \{i_{k_2} \prec i_{l_2}\}, \ldots, \{i_{k_q} \prec i_{l_q}\}\}$ imposed by Lemma A.1 are dominated by our delaying modes $\{\{i_K \prec i_{l_1}\}, \{i_K \prec i_{l_2}\}, \ldots, \{i_K \prec i_{l_q}\}\}$ on the first level of the search tree.

Case 2.2: $i_m = i_k$.

In this case, we would have delayed i_l and i_n by i_k $(i_l \neq i_n)$. If the corresponding ESS is feasible, the corresponding delaying modes will be identified by our procedure, since $\{i_l, i_n\}$ then constitutes a valid delaying alternative and i_k a valid delaying activity. If, however, the ESS would still not be time feasible, other precedence constraints would have to be added. Now we would run into Case 2.1 or 2.2, depending on the delaying activity, but now one level down in the search tree. As we have shown in both cases, the corresponding delaying modes will either also be identified by our procedure, or be dominated by a generated delaying mode. Repeating a similar argument for any node created in the search tree, leads to the proof of Lemma A.1.

Lemma A.3. *In order to resolve a resource conflict, it is sufficient to consider minimal delaying modes.*

Proof. According to Lemma A.1, a branching strategy based on delaying modes leads to the optimal solution. Lemma A.1 does not exclude non-minimal delaying modes, i.e. with a corresponding delaying alternative D_d that contains other delaying alternatives $D_{d'}$ as a subset. These non-minimal delaying alternatives D_d (and corresponding non-minimal delaying modes), however, need not be examined, since the set of feasible solutions we can obtain by branching from a node with a corresponding non-minimal delaying mode will be a proper subset of the set of feasible solutions obtained when branching from a node with a delaying mode in which D_d is replaced by $D_{d'}$. The project network created by the delaying mode corresponding to D_d is identical to the one created by the delaying mode corresponding to $D_{d'}$, except that one or more extra precedence relations are added. Therefore, the set of feasible solutions that can be obtained from the node corresponding to D_d is a proper subset of the set of feasible solutions that can be obtained from the node corresponding to $D_{d'}$. Therefore, delaying alternative D_d (and all corresponding delaying modes) can be eliminated.

Lemma A.4. *The delaying strategy based on Lemma A.3 leads to the optimal solution in a finite number of steps.*

Proof. At each branch of the search tree, we create a number of nodes equal to the number of minimal delaying modes. Clearly, the maximal number of activities in $S(t^*)$ is equal to n. Then, it can be shown that the maximal number of minimal delaying modes is equal to $n!$. Indeed, if the number of activities in each delaying alternative is equal to 1, the number of delaying alternatives is equal to n and the number of delaying modes equal to $n(n-1)$ because there are $n-1$ possible delaying activities for each delaying alternative. If the number of activities per delaying alternative is equal to 2, the maximal number of delaying modes would be equal to $n(n-1)(n-2)/2$, because there are $n!/(n-2)!2!$ delaying alternatives and $n-2$ possible delaying activities. In general, x activities per delaying alternative would give rise to maximally $n!/(n-x-1)!x!$ delaying modes. Clearly, the maximal number of minimal delaying modes is always smaller than $n!$, even if the number of activities varies among the delaying alternatives. Thus, the maximal number of nodes generated during each branching step equals $n!$. We know that the maximal number of zero-lag FS precedence relations that can be added to a project network (without affecting time-feasibility) equals $n(n-1)/2$. Therefore, the maximal number of levels in the search tree equals $n(n-1)/2$. Consequently, the maximal number of nodes generated in the search tree equals $\sum_{i=0}^{n(n-1)/2}(n!)^i$, the maximal number of leaf nodes being equal to $(n!)^{n(n-1)/2}$.

According to Lemma A.3, one of these nodes is bound to contain the optimal solution. Since the number of nodes in the search tree is finite, the optimal solution can be found in a finite number of steps. □

A.2. Proof of Theorem 4

If we split up each activity i of an RCPSP-GPR instance into unit-duration *subactivities* $(i_1, i_2, \ldots, i_p)$, connected with zero-lag minimal FS time lags, we obtain the preemptive version of the RCPSP-GPR (see also Demeulemeester and Herroelen, 1996 for the preemptive RCPSP), provided that the precedence relations between the activities are connected to the correct subactivity. However, if we also add zero-lag *maximal* FS time lags, we again obtain the original problem, since all the subactivities of a given activity have to be performed consecutively. If the project network is represented in its standardized form (as a constraint digraph), all precedence relations are of the SS type which can be represented in the unit-duration RCPSP-GPR by precedence relations between the first subactivities of each activity only ($i \prec j \Rightarrow s_{i_1} + d_i \leqslant s_{j_1}$). The zero-lag minimal and maximal FS precedence relations between the subactivities are then represented by two minimal SS relations equal to 1 and -1 respectively for each consecutive pair of subactivities ($s_{i_k} + 1 \leqslant s_{i_{k+1}}$ and $s_{i_{k+1}} - 1 \leqslant s_{i_k}$).

Since the two problems are identical, a lower bound for the unit-duration problem will also be a valid lower bound for the original problem. Now, we show that the exact value of lb_3^g is obtained by calculating lb_3' (Demeulemeester and Herroelen, 1997a) for the unit-duration problem, with the additional assumptions that (a) the order in which the subactivities are placed in the list L is determined by looking at the original problem (i.e. the number of companions is calculated by looking at the original activities, not the subactivities) and (b) (which logically follows from (a)) if a subactivity i_k of activity i is removed from L, so are all the other subactivities $i_l (l \neq k)$ of activity i (which can never be companions of i_k).

Since all the time lags between the subactivities of one and the same activity are equal to 1 (from i_k to i_{k+1}) and -1 (from i_{k+1} to i_k), we know that $i_k (1 \leqslant k \leqslant p)$ is a companion of

$$j_l (1 \leqslant l \leqslant q) \quad \text{if both } d_{ij} - (k-1) + (l-1) \leqslant 0 \text{ and } d_{ji} + (k-1) - (l-1) \leqslant 0 \tag{A.1}$$

i.e. the longest path length between i_k and j_l or between j_l and i_k may never exceed zero, otherwise they can never be scheduled in parallel.

Assume we compute lb_3' for the unit-duration RCPSP-GPR. When we remove subactivity i_k from L, we also have to remove its companions from L, i.e. each j_l for which condition (A.1) applies. By simplifying Eq. (A.1), we obtain

$$\text{remove} j_l \text{ if } \quad k + d_{ji} \leqslant l \leqslant k - d_{ij} \tag{A.2}$$

(note that $k + d_{ji} \leqslant k - d_{ij}$ because otherwise: $k + d_{ji} > k - d_{ij} \Rightarrow d_{ij} + d_{ji} > 0 \Rightarrow d_{ii} > 0$, which would result in a time-infeasibility). Consequently, when we remove all the subactivities i_k from L, we have to remove each subactivity j_l for which condition (A.2) applies for any subactivity i_k. The smallest value l for which (A.2) is true is equal to $\max\{1, 1 + d_{ji}\}$, whereas the largest value for l equals $\min\{d_j, d_i - d_{ij}\}$. Therefore, the number of subactivities from activity j to be removed equals: $\min\{d_j, d_i - d_{ij}\} - \max\{1, 1 + d_{ji}\} + 1$. However, the maximal number of subactivities of activity j to be removed can never exceed d_j. Therefore, the number of subactivities j_l to be removed from L if all subactivities i_k are removed from L equals

$$\min\{\min\{d_j, d_i - d_{ij}\} - \max\{1, 1 + d_{ji}\} + 1, d_j\}. \tag{A.3}$$

We will now examine each of the combinations given in Table 1 which represent time-feasible project networks and in which activities i and j are companions.

Case 1: $d_{ij} \leqslant 0$ and $0 < d_{ji} < d_j$ (row 2 and columns 3, 4 in Table 1).

Using Case 1, Eq. (A.3) can be simplified to: $\min\{\min\{d_j, d_i - d_{ij}\} - 1 - d_{ji} + 1, d_i\} = \min\{\min\{d_j, d_i - d_{ji}\} - d_{ji}, d_i\} = \min\{\min\{d_j - d_{ji}, d_i - d_{ij} - d_{ji}\}, d_i\} = \min\{d_j - d_{ji}, d_i - d_{ij} - d_{ji}, d_i\}$. We know that $d_i \leqslant d_i - d_{ij} - d_{ji}$, because otherwise we would get

$d_{ij} + d_{ji} > 0$ which leads to time-infeasibility. Therefore, the number of subactivities of activity j to be removed equals $\min\{d_j - d_{ji}, d_i\}$, which is equivalent to a reduction of the remaining duration of activity j with the same amount. This value corresponds to the values of cells (2,3) and (2,4) in Table 1. Because $d_{ji} < d_j$ this value can never become negative.

Case 2: $0 < d_{ij} < d_i$ and $d_{ji} \leqslant 0$ (column 2 and rows 3, 4 in Table 1).

Using Case 2, Eq. (A.3) can be simplified to: $\min\{\min\{d_j,\ d_i - d_{ij}\} - 1 + 1,\ d_i\} = \min\{\min\{d_j,\ d_i - d_{ij}\}, d_i\} = \min\{d_j, d_i - d_{ij}, d_i\} = \min\{d_j, d_i - d_{ij}\}$, which is the value for cells (3,2) and (4,2) in Table 1. Again, this value can never become negative. Notice also that the value given in Table 1 $(d_i - d_{ij})$ is slightly different, because there is no need to check whether the remaining duration of activity j becomes negative (if $d_j^r \leqslant 0$, it is completely removed from L).

Case 3: $d_{ij} \leqslant 0$ and $d_{ji} \leqslant 0$ (rows 3, 4 and columns 3, 4 in Table 1).

Using Case 3, Eq. (A.3) can be simplified to: $\min\{\min\{d_j,\ d_i - d_{ij}\} - 1 + 1,\ d_i\} = \min\{\min\{d_j,\ d_i - d_{ji}\},\ d_i\} = \min\{d_j,\ d_i - d_{ij},\ d_i\} = \min\{d_j,\ d_i\}$, which is the value for cells (3,3), (3,4), (4,3) and (4,4) in Table 1. Notice again the slight difference because of the non-negativity constraint of the remaining duration of activity j, which is not needed in our procedure.

For the sake of completeness, we will extend the proof to the time lag combinations which prohibit activities i and j to be companions (cells (4,1) and (1,4) in Table 1). A similar reasoning can be given for the time lag combinations which are not time feasible (cells (1,1), (1,2), (2,1), (2,2), (3,1) and (1,3) in Table 1).

Case 4: $d_{ij} \leqslant -d_j$ and $d_{ji} \geqslant d_j$ (row 1, column 4 in Table 1).

Using Case 4, Eq. (A.3) can be simplified to: $\min\{\min\{d_j, d_i - d_{ij}\} - 1 - d_{ji} + 1, d_i\} = \min\{\min\{d_j,\ d_i - d_{ij}\} - d_{ji}, d_i\} = \min\{d_j - d_{ji},\ d_i - d_{ij} - d_{ji}, d_i\}$. Because $d_{ji} \geqslant d_j$, we know that $d_j - d_{ji} \leqslant 0$. Therefore, the minimum will be smaller than zero, which is a logical result because then activities i and j will not be companions at all (as is given in cell (1,4) in Table 1). Subsequently, no subactivities j_l have to be removed from L, or, in the original problem, the remaining duration of activity j in L need not be reduced.

Case 5: $d_{ij} \geqslant -d_i$ and $d_{ji} \leqslant -d_j$ (row 1, column 4 in Table 1).

Similar argument as for Case 4. □

Proof of Theorem 5

If the set of added precedence constraints which leads to the project network in node x contains as a subset the set of added precedence constraints leading to the project network in a previously examined node y, the project network obtained in node x consists of the project network obtained in node y, extended with zero or more extra zero-lag finish-start precedence relations. Therefore, since the problem in node x is more constrained than the problem in node y, the set of feasible solutions which can be obtained when branching from node x is a subset of the set of feasible solutions which can be obtained when branching from node y ($\Omega_x \subset \Omega_y$). Therefore, the best possible solution that can be obtained when branching from node x can never be superior to the best possible solution that can be obtained when branching from node y.

We know that node y is already examined and that it stems from another part of the search tree than node x. Therefore, when we reach node x, because of the nature of the depth-first search procedure, node y is also backtracked upon. Therefore, we know that the best possible solution that can be obtained by branching from node y is already determined. Therefore, node x can be fathomed since no superior solution can be obtained by branching from it. □

Proof of Theorem 6

We know that $\exists\, i, j \in V$ and a resource type k for which:

$$r_{ik} + r_{jk} > a_k, \tag{A.4}$$

$$-d_j < d_{ij} < d_i. \tag{A.5}$$

Suppose that there exists a feasible solution for the RCPSP-GPR. In each feasible solution (therefore

also in the optimal solution), Eq. (A.4) guarantees that either $i \prec j$ or $j \prec i$. Suppose that

$$j \prec i \Rightarrow s_j + d_j \leqslant s_i. \tag{A.6}$$

We can derive from Eq. (A.5) that

$$d_{ij} > -d_j. \tag{A.7}$$

Combining the general expression $s_i + d_{ij} \leqslant s_j$ with Eq. (A.7), we get

$$s_i - d_j < s_j. \tag{A.8}$$

Substituting (A.6) into (A.8) yields: $s_j + d_j - d_j < s_j \Rightarrow s_j < s_j$, which is impossible. Therefore, in each feasible solution: $i \prec j \Rightarrow s_i + d_i \leqslant s_j$. Consequently, we can set $l_{ij} = d_i$.

Now suppose that there does not exist a feasible solution for the RCPSP-GPR. Then, adding the constraint $l_{ij} = d_i$ will not change the fact that no feasible solution is obtained, since its addition further constrains the problem, leading to a set of feasible solutions which is a subset of the original set. Since the original set was empty, so will the new set of feasible solutions. □

References

Bartusch, M., Möhring, R.H., Radermacher, F.J., 1988. Scheduling project networks with resource constraints and time windows. Ann. Oper. Res. 16, 201–240.

Brinkmann, K., Neumann, K., 1996. Heuristic procedures for resource-constrained project scheduling with minimal and maximal time lags: The minimum project-duration and resource-levelling problem. J. Dec. Syst. 5, 129–156.

Crandall, K.C., 1973. Project planning with precedence lead/lag factors. Proj. Mgmt. Quart. 4, 18–27.

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

Demeulemeester, E., Herroelen, W., 1996. A branch-and-bound procedure for the preemptive resource-constrained project scheduling problem. Eur. J. Oper. Res. 90, 334–348.

Demeulemeester, E., Herroelen, W., 1997a. New benchmark results for the resource-constrained project scheduling problem. Management Sci. 43 (to appear).

Demeulemeester, E., Herroelen, W., 1997b. A branch-and-bound procedure for the generalized resource-constrained project scheduling problem. Oper. Res. 45, 201–212.

De Reyck, B., 1995. Project scheduling under generalized precedence relations – a review: Parts 1 and 2. Research Reports 9517 and 9518, Department of Applied Economics, Katholieke Universiteit Leuven.

De Reyck, B., Herroelen, W., 1996a. A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. Research Report 9613, Department of Applied Economics, Katholieke Universiteit Leuven.

De Reyck, B., Herroelen, W., 1996b. Computational experience with a branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. Research Report 9628, Department of Applied Economics, Katholieke Universiteit Leuven.

Elmaghraby, S.E., 1977. Activity Networks: Project Planning and Control by Network Models. Wiley, New York.

Elmaghraby, S.E., Herroelen, W., 1990. The scheduling of activities to maximize the net present value of projects. Eur. J. Oper. Res. 49, 35–49.

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

Franck, B., Neumann, K., 1996. Priority-rule methods for the resource-constrained project scheduling problem with minimal and maximal time lags – an emprical analysis. In: Fifth International Workshop on Project Management and Scheduling. 11 – 13 April, Poznan, pp. 88–91.

Grinold, R.C., 1972. The payment scheduling problem. Naval Res. Logist. Quart. 19, 123–136.

Herroelen, W., Gallens, E., 1993. Computational experience with an optimal procedure for the scheduling of activities to maximize the net present value of projects. Eur. J. Oper. Res. 65, 274–277.

Herroelen, W., Demeulemeester, E., Van Dommelen, P., 1997. Project network models with discounted cash flows: A guided tour through recent developments. Eur. J. Oper. Res. 100, 97–121.

Herroelen, W., Demeulemeester, E., Van Dommelen, P., 1996. An optimal recursive search procedure for the deterministic unconstrained max-npv project scheduling problem. Research Report 9603, Department of Applied Economics, Katholieke Universiteit Leuven.

Icmeli, O., Erengüç, S.S., 1996. A branch-and-bound procedure for the resource-constrained project scheduling problem with discounted cash flows. Management Sci. 42, 1395–1408.

Kelley Jr., J.E., Walker, M.R., 1959. Critical path planning and scheduling. Eastern Joint Computing Conference 16, 160–172.

Kerbosh, J.A.G.M., Schell, H.J., 1975. Network planning by the Extended METRA Potential Method. Report KS-1.1, University of Technology Eindhoven, Department of Industrial Engineering.

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

Lawler, E.L., 1976. Combinatorial Optimization: Networks and Matroids. Holt, Rinehart & Winston, New York.

Malcolm, D.G., Roseboom, J.H., Clark, C.E., Fazar, W., 1959. Applications of a technique for R&D program evaluation (PERT). Oper. Res. 7, 646–669.

Mastor, A.A., 1970. An experimental and comparative evaluation of production line balancing techniques. Management Sci. 16, 728–746.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L., 1997. An exact algorithm for project scheduling with resource constraints based on a new mathematical programming formulation. Management Sci. (to appear).

Moder, J.J., Phillips, C.R., Davis, E.W., 1983. Project Management with CPM, PERT and Precedence Diagramming, 3rd ed. Van Nostrand Reinhold, New York.

Möhring, R.H., 1996 (private communication).

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

Neumann, K., Zhan, J., 1995. Heuristics for the minimum project-duration problem with minimal and maximal time lags under fixed resource constraints. J. Intell. Manuf. 6, 145–154.

Pascoe, T.L., 1966. Allocation of resources – CPM. Revue Française de Recherche Opérationelle 38, 31–38.

Patterson, J.H., 1984. A comparison of exact procedures for solving the multiple resource-constrained project scheduling problem. Management Sci. 30, 854–867.

Roy, B., 1962. Graphes et ordonnancement. Revue Française de Recherche Opérationelle, 323–333.

Russell, A.H., 1970. Cash flows in networks. Management Sci. 16, 357–373.

Schwindt, C., 1996a. Generation of resource-constrained project scheduling problems with minimal and maximal time lags. Technical Report WIOR-489, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe.

Schwindt, C., 1996b (private communication).

Schwindt, C., Neumann, K., 1996. A new branch-and-bound-based heuristic for resource-constrained project scheduling with minimal and maximal time lags. In: Fifth International Workshop on Project Management and Scheduling, 11–13 April, Poznan, pp. 212–215.

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

Wiest, J.D., 1981. Precedence diagramming methods: Some unusual characteristics and their implications for project managers. J. Oper. Mgmt. 1, 121–130.

Wikum, E.D., Donna, C.L., Nemhauser, G.L., 1994. One-machine generalized precedence constrained scheduling problems. Oper. Res. Lett. 16, 87–99.

Zhan, J., 1994. Heuristics for scheduling resource-constrained projects in MPM networks. Eur. J. Oper. Res. 76, 192–205.