4-1998

Resource-constrained project scheduling: A survey of recent developments

Willy HERROELEN

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

Erik DEMEULEMEESTER
Singapore Management University

Pergamon

Computers Ops Res. Vol. 25, No. 4, pp. 279–302, 1998

PII: S0305-0548(97)00055-5
Printed in Great Britain
0305-0548/98 $19.00+0.00

RESOURCE-CONSTRAINED PROJECT SCHEDULING: A SURVEY OF RECENT DEVELOPMENTS

Willy Herroelen†‡, Bert De Reyck‡ and Erik Demeulemeester‡

Operations Management Group, Department of Applied Economics, Catholic University Leuven, Naamsestraat 69, B-3000 Leuven, Belgium

(Received October 1996; in revised form July 1997)

Scope and Purpose—The resource-constrained project scheduling problem involves the scheduling of a project in order to minimize its duration, subject to zero-lag finish–start precedence constraints between the activities and constant availability constraints on the required set of renewable resources. Over the past few years, considerable progress has been made with the use of optimal solution procedures for this basic problem type and its important extensions. The objective of this paper is to provide a survey of what we believe are the important recent developments in the area.

Abstract—We review recent advances in dealing with the resource-constrained project scheduling problem using an efficient depth-first branch-and-bound procedure, elaborating on the branching scheme, bounding calculations and dominance rules, and discuss the potential of using truncated branch-and-bound. We derive conclusions from the research on optimal solution procedures for the basic problem and subsequently illustrate extensions to a rich and realistic variety of related problems involving activity preemption, the use of ready times and deadlines, variable resource requirements and availabilities, generalized precedence relations, time/cost, time/resource and resource/resource trade-offs and non-regular objective functions. © 1998 Published by Elsevier Science Ltd. All rights reserved

1. INTRODUCTION

Scheduling and sequencing is concerned with the optimal allocation of scarce resources over time. *Scheduling* deals with defining which activities are to be performed at a particular time. *Sequencing* concerns the ordering in which the activities have to be performed. The allocation of scarce resources over time has been the subject of extensive research since the early days of operations research in the mid 1950s.

Scheduling and sequencing theory, more than any other field in the area of operations management and operations research, is characterized by a virtually unlimited number of *problem types*. The terminology arose in the processing and manufacturing industries and most research has traditionally been focused on **deterministic machine scheduling** (see the books by Muth and Thompson [1], Conway *et al.* [2], Ashour [3], Baker [4], Rinnooy Kan [5], French [6] Bellmann *et al.* [7], Herroelen [8], Blazewicz *et al.* [9], Morton and Pentico [10], Tanaev *et al.* [11,12], Brucker [13] and Pinedo [14]). In this context the type of resource is traditionally considered to be a *machine* that can perform at most one activity at a time. Over the years, several (unrealistic) assumptions of the basic machine scheduling problems have been relaxed. A natural extension involves the presence of *additional resources*, where each resource has a limited size and each job requires the use of a part of each resource during its execution (Gargeya and Deane [15]). This leads us to the area of *resource-constrained project scheduling* which involves the scheduling of project activities subject to precedence and resource constraints. The field again covers a wide variety of problem types.

This article provides a guided tour through what we believe to be important recent developments in the area of resource-constrained project scheduling. Our main focus will be on the recent progress made with optimal branch-and-bound procedures for the basic resource-constrained project scheduling problem (RCPSP) and its important extensions.

The organization of this paper is as follows. Section 2 focuses on the classical resource-constrained

† To whom all correspondence should be addressed (email: Willy.Herroelen@econ.kuleuven.ac.be).

‡ Willy Herroelen and Erik Demeulemeester are Professors of Operations Management in the Department of Applied Economics at the Katholieke Universiteit Leuven (Belgium). At the time this paper was written Bert De Reyck was completing his Ph.D. dissertation under their guidance. The current research interests of the authors include the construction of exact and heuristic solution procedures for various types of resource-constrained project scheduling problems. Related publications have appeared in *Operations Research, Management Science, European Journal of Operational Research* and *Computers and Operations Research.*

project scheduling problem (RCPSP). Section 3 deals with the preemptive resource-constrained project scheduling problem (PRCPSP). Section 4 concentrates on the recent developments of models dealing with generalized precedence relations. Section 5 addresses the problem of maximizing the net present value of projects, as an illustration of the use of non-regular measures of performance. Section 6 is devoted to project scheduling problems with time/resource and resource/resource trade-offs. Section 7 is reserved for our overall conclusions.

2. THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM (RCPSP)

2.1. Notation and terminology

We assume that a project is represented by an activity-on-the-node network $G=(V,E)$ in which V denotes the set of vertices (nodes) representing the activities and E is the set of edges (arcs) representing the finish–start precedence relationships with zero time-lag. The activities are numbered from 1 to n, where the dummy activities 1 and n mark the beginning and end of the project. The activities are to be performed without preemption. The fixed integer duration of an activity is denoted by d_i ($1 \leq i \leq n$), its integer starting time by s_i ($1 \leq i \leq n$) and its integer finishing time by f_i ($1 \leq i \leq n$). There are K renewable resource types with r_{ik} ($1 \leq i \leq n$, $1 \leq k \leq K$) the constant resource requirement of activity i for resource type k and a_k the constant availability of resource type k. Conceptually, the RCPSP can be formulated as follows:

$$\text{Min } f_n \tag{1}$$

subject to

$$f_1 = 0, \tag{2}$$

$$f_j - d_j \geq f_i, \ \forall (i,j) \in H, \tag{3}$$

$$\sum_{i \in S_t} r_{ik} \leq a_k, \ t = 1,2 \ldots f_n; \ k = 1,2 \ldots, K, \tag{4}$$

where H denotes the set of pairs of activities indicating precedence constraints and S_t denotes the set of activities in progress in time interval $]t-1,t]$: $S_t = \{i \mid f_i - d_i < t \leq f_i\}$. (2) assigns a completion time of 0 to the dummy start activity 1. The precedence constraints given by (3) indicate that activity j can only be started if all predecessor activities i are completed. The resource constraints given in (4) indicate that for each time period $]t-1,t]$ and for each resource type k, the renewable resource amounts required by the activities in progress cannot exceed the resource availability. The objective function is given as (1). The project duration is minimized by minimizing the finishing time of the unique dummy ending activity n.

A problem example is given in Fig. 1. The numbers above each node of the project network denote the fixed activity durations. The numbers below each node denote the per period resource requirement for the single resource type involved in completing this project. The single resource required has a constant availability of 5 units per period. The unique optimal solution for this project is represented in Fig. 2 and has a duration of 7 periods. The minimal resource-constrained makespan for the project happens to equal

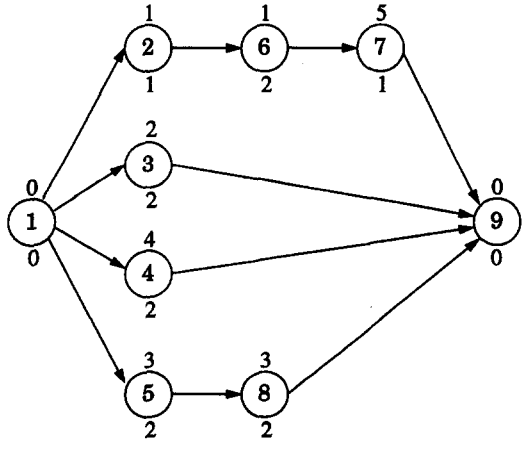

Fig. 1. RCPSP example.

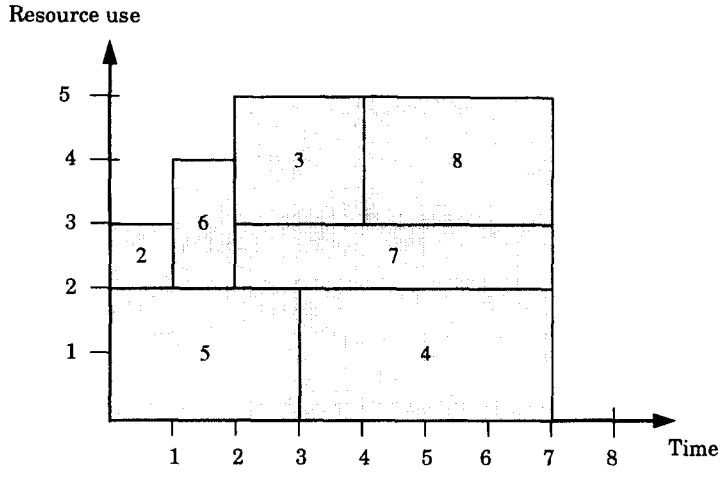

Fig. 2. Resource profile of the optimal solution.

the length of the critical path. This is a mere coincidence. Most often, the addition of resource constraints will lead to an increase of the project duration beyond the critical path length.

2.2. Optimal solution procedures and computational experience

The RCPSP, which as a generalization of the job-shop scheduling problem is NP-hard in the strong sense [16], has been extensively studied in the literature. Previous research on optimal procedures basically involves the use of *mathematical programming* [17–26] and *implicit enumeration*; i.e. dynamic programming [27,28] and branch-and-bound [29–44]. For comprehensive reviews we refer the reader to Davis [45,46], Herroelen [47], Patterson [48], Icmeli *et al.* [49], Elmaghraby [50], Herroelen and Demeulemeester [51], and Özdamar and Ulusoy [52]. Over the past decade, considerable progress in the use of optimal procedures for the RCPSP has been reported on two problem sets: the 110 problems assembled by Patterson [48] and the 480 test instances generated by Kolisch *et al.* [53].

2.2.1. The Patterson problem set. Patterson [48] assembled a set of 110 test problems (with 7 up to 50 activities and 1 up to 3 renewable resource types) which over the years became a de facto standard for validating optimal and suboptimal procedures for the RCPSP. The computational experiments have been conducted by different authors on a wide variety of computers, making direct comparisons rather difficult. Table 1 lists a number of results obtained on this set.

Davis and Heidorn [32] presented an implicit enumeration scheme which exploits the relationship between the RCPSP and the assembly line balancing problem. Representing activity durations as a series of unit duration tasks, the original problem is reduced to one of finding the shortest path between the start and end nodes of a directed graph which is derived from the feasible subsets, i.e. sets, which, if they contain a certain task, also contain all predecessors of this task. Talbot and Patterson [34] developed an implicit enumeration scheme involving a systematic enumeration of all possible finish times for the activities. The algorithm uses the concept of a cut to eliminate from explicit consideration possible

Table 1. Computational results on the Patterson problem set

Authors	Computer language	Computer configuration	Average CPU-time (s)	Time Limit	Problems solved to optimality
Davis and Heidorn [32]	Fortran V	Amdahl 470/V8	14.02	300	96
Talbot and Patterson [34]	Fortran V	Amdahl 470/V8	14.98	300	97
Stinson *et al.* [33]	Fortran V	Amdahl 470/V8	0.82	300	110
CAT revised (Christofides *et al.* [36]; Demeulemeester *et al.* [54])	Fortran V	IBM 3090	56.82	No	110
Bell and Park [38]	Common Lisp	Apple Macintosh Plus	340.57	No	110
DH (Demeulemeester and Herroelen [40])	Borland Turbo C – DOS	80386 25 MHz	0.21	No	110
Carlier and Neron [42]	not reported	Not reported	199.07	No	44
DH1 (Demeulemeester and Herroelen [41])	Microsoft Visual C++ 2.0 – Windows NT	80486 25 MHz	0.02	No	110
DH2	Microsoft Visual C++ 2.0 – Windows NT	Pentium Pro 200 MHz	0.002	No	110

inferior activity completion times. Stinson *et al.* [33] developed a best-first branch-and-bound procedure based on strong bounding criteria and dominance rules. The nodes of the branch-and-bound tree correspond to partial schedules based on the principle of starting activities as early as possible, satisfying both precedence and resource constraints. Patterson [48] found that among the three procedures, only the branch-and-bound of Stinson *et al.* was able to solve all 110 test problems within the time limit of 300 s on the mainframe mentioned in Table 1.

Christofides *et al.* [36] have developed CAT, a depth-first branch-and-bound procedure that generates a branch-and-bound tree, whose nodes correspond to semi-active feasible partial schedules. The procedure only branches to resolve a resource conflict. It has been shown [54] that the CAT algorithm may occasionally miss the optimum. Moreover, computational results obtained by Demeulemeester *et al.* [54] cannot confirm the claim that the revised CAT algorithm is competitive with the Stinson *et al.* [33] procedure.

Bell and Park [38] have developed an A* algorithm that is a best-first search algorithm, where each state in the search tree consists of a set of precedence constraints. Starting from an initial state network which contains only those precedence constraints expressed in the original problem, the goal state is the minimal makespan network that satisfies resource constraints as well as precedence constraints. State transitions are generated by adding arcs to the network. Computational experience is not convincing (8 problems in the set could not be solved optimally within 1000 s).

Carlier and Latapie [39] have presented a novel branch-and-bound procedure that extends earlier work on the job-shop scheduling problem. An execution interval is associated with each activity based on its computed ready time and due date. Branching consists in choosing a task and splitting its interval of execution into two intervals, whose union gives the original interval again. The authors only report on a selection of the test problems. The procedure failed to generate the optimal solution for some of the test problems (one problem could not be solved after 2 h of CPU-time on a VAX 8530). Carlier and Neron [42] use bounds based on m-machine problems which are generated at the root of the search tree. Their branching scheme is based on so-called left-tight schedules in which activities are either scheduled at the beginning of a schedule or at its end. Results obtained on a subset of the Patterson problems, reveal the rather high computational cost of the procedure, which is still in the development phase.

The depth-first *DH-procedure*, developed by Demeulemeester and Herroelen [40], seems to use the most efficient solution logic. The incorporation of a new bounding argument, a new version of a dominance rule and the use of 32-bit programming led to the development of the more efficient *DH1-procedure* which runs on a Windows NT 3.50 platform [41]. This resulted in a speed boost by a factor of more than 10 on the 110 Patterson problems as compared to the code used for the 1992 paper. Recent efforts to improve and repolish the code (subsequently referred to as the *DH2-procedure*) further reduced the computational effort to a negligible level.

2.2.2. The 480 KSD test problems. Recent research by Kolisch *et al.* [53] questioned the use of the 110 problem set and led to the development of *ProGen*, a network generator which allows for the generation of RCPSP problem instances which satisfy pre-set problem parameters. Table 2 summarizes the computational results obtained on a set of 480 problem instances, generated on the basis of a full factorial design (30 activities, 4 renewable resource types). The DH-procedure could only find (and verify) the optimal solution for 428 problems within a 1 h time limit on a personal computer running at 15 MHz.

Mingozzi *et al.* [43] presented a new 0-1 linear programming formulation that requires an exponential number of variables, corresponding to all feasible subsets of activities that can be simultaneously executed without violating resource or precedence constraints. They present a tree search algorithm

Table 2. Computational results on the ProGen problem set

Authors	Computer language	Computer configuration	Average CPU-time (s)	Time limit	Problems solved to optimality
DH (Demeulemeester and Herroelen [40])	Borland Turbo C – DOS	80386 25 MHz	79.91	3600	428
Mingozzi *et al.* [43]	Fortran 77	80386 15 MHz	not reported	No	480
DH1 (Demeulemeester and Herroelen [41])	Microsoft Visual C++ 2.0 – Windows NT	80486 25 MHz	12.33	3600	479
DH2 Brucker *et al.* [44]	Microsoft Visual C++ 2.0 – Windows NT	Pentium Pro 200 MHz	0.37	No	480
	C-Solaris 2.5 + CPLEX	Sun/Sparc 20/801	17.88	3600	407

BBLB3 based on this formulation which can solve the 52 hard KSD instances that could not be solved by the DH-procedure, while it is reported to be, on the average, 5 times slower on the Patterson test problems. They conclude that BBLB3 is competitive with the DH-procedure on hard instances (an average of 1161.06 s on the 150 so-called hard instances), while it does not dominate DH on easier problems (an average of 88.375 s on 80 so-called easier instances). Brucker *et al.* [44] developed a branch-and-bound algorithm which performs a depth-first search on a binary search tree, the nodes of which correspond to so-called schedule schemes (sets of disjunctions, conjunctions, parallelity and flexibility relations). The authors develop various bounding and dominance rules and concepts of immediate selection.

The DH1-procedure optimally solves the 480 instances, albeit that one of the problems needed 3 h of computation time. Moreover, a truncated version of the procedure yields excellent results. Running the DH1-procedure for small amounts of time yields solutions which are very close to the optimum. The DH2-procedure leads to a further gain in efficiency. These results constitute a new benchmark for the RCPSP. Moreover, the efficient and effective branching scheme, and many of the lower bound and dominance arguments have been extended to a wide and relevant variety of problem settings. They are elaborated on in the subsequent sections.

2.3. The DH- and the new DH-procedures

The *DH1-procedure* [41] is conceptually very similar to the *DH-procedure* described in Demeulemeester and Herroelen [40]. It generates a search tree, the nodes of which correspond to feasible partial schedules in which finish times temporarily have been assigned to a subset of the activities of the project. *Partial schedules* PS_m are only considered at those time instants m which correspond to the completion time of one or more project activities. The partial schedules are constructed by starting activities as soon as possible within the precedence and resource constraints. A partial schedule PS_m at time m thus consists of the set of *temporarily* scheduled activities. Scheduling decisions are temporary in the sense that temporarily scheduled activities may be delayed as a result of decisions made at later stages in the search process. Partial schedules are built up starting at time 0 and proceed systematically throughout the search process by adding at each decision point subsets of activities, including the empty set, until a complete feasible schedule is obtained. In this sense, a complete schedule is a *continuation of a partial schedule.*

At every time instant m we define the *eligible set* E_m as the set of activities which are not in the partial schedule and whose predecessor activities have finished. These eligible activities can start at time m if the resource constraints are not violated. Demeulemeester and Herroelen [40] have proven two theorems which allow the procedure, at decision point m, to decide on which eligible activities must be scheduled by themselves, and which pair of eligible activities must be scheduled concurrently.

Theorem 1. *If at time m the partial schedule* PS_m *has no activity in progress and an eligible activity i cannot be scheduled together with any other unscheduled activity at any time instant* $m' \geq m$ *without violating the precedence and resource constraints, then there exists an optimal continuation of the partial schedule with the eligible activity i put in progress (started) at time m.*

Theorem 2. *If at time m the partial schedule* PS_m *has no activity in progress, and if there is an eligible activity i which can be scheduled concurrently with only one other unscheduled activity j at any time instant* $m' \geq m$ *without violating precedence or resource constraints, and if activity j is both eligible and not longer in duration than activity i, then there exists an optimal continuation of the partial schedule in which both activities i and j are put in progress at time m.*

If it is impossible to schedule all activities at time m, a *resource conflict* occurs which will produce a new branching in the branch-and-bound tree. The branches describe ways to resolve the resource conflict by deciding on which combinations of activities are to be delayed. A *delaying set* $D(p)$ consists of all subsets of activities D_q, either in progress or eligible, the delay of which would resolve the current resource conflict at level p of the search tree. A *delaying alternative* D_q is minimal if it does not contain other delaying alternatives $D_v \in D(p)$ as a subset. Demeulemeester and Herroelen [40] give the proof that in order to resolve a resource conflict, it is sufficient to consider only minimal delaying alternatives.

One of the minimal delaying alternatives (nodes in the search tree) is arbitrarily chosen for branching. The delay of a delaying alternative D_q is accomplished by adding a *temporal constraint* causing the corresponding activities to be delayed up to the *delaying point*, which is defined as the earliest completion of an activity in the set of activities in progress, that does not belong to the delaying alternative.

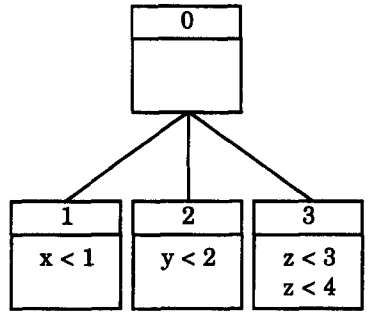

Fig. 3. Minimal delaying alternatives.

The branching scheme can best be illustrated on a small problem example. Assume that the set of activities {1,2,3,4} creates a resource conflict at decision point m and that the minimal delaying set is {{1},{2},{3,4}}. Assume that activity x is the earliest finishing activity among *2*, *3* and *4*, that activity y is the earliest finishing activity among the activities 1, 3 and 4 and that activity z is the earliest finishing activity among the activities 1 and 2. The resulting delaying alternatives are represented in Fig. 3. The operator "<" denotes a *temporal* constraint, i.e. a delay up to the earliest finishing time of an activity in progress that does not belong to the delaying alternative.

The delayed activities are removed from the partial schedule and the set of activities in progress, and the algorithm continues by computing a new decision point. The search process continues until the dummy end activity has been scheduled. Every time such a complete schedule has been found, *backtracking* occurs: a new delaying alternative is arbitrarily chosen from the set of delaying alternatives $D(p)$ at the highest level p of the search tree that still has some unexplored delaying alternatives left, and branching continues from that node. When level zero is reached in the search tree, the search process is completed.

Two *dominance rules* are used to prune the search tree. The first one is a variation of the well-known left-shift dominance rule, and can be stated as follows:

Theorem 3. *If the delay of the delaying alternative at the previous level of the branch-and-bound tree forced an activity i to become eligible at time m, if the current decision is to start activity i at time m and if activity i can be left-shifted without violating the precedence or resource constraints (because activities in progress were delayed), then the corresponding partial schedule is dominated.*

The second dominance rule is based on the concept of a cutset. At every time instant m a *cutset* C_m is defined as the set of unscheduled activities for which all predecessor activities belong to the partial schedule PS_m.

Theorem 4. *Consider a cutset C_m at time m which contains the same activities as a cutset C_k, which was previously saved during the search of another path in the search tree. If time k was not greater than time m and if all activities in progress at time k did not finish later than the maximum of m and the finish time of the corresponding activities in PS_m, then the current partial schedule PS_m is dominated.*

As indicated in Table 3, the original DH-procedure has been tested with three *lower bounding rules*. The well-known remaining critical path length bound *LB0* and critical sequence lower bound *LBS* [33]

Table 3. Basic differences between DH procedures

	DH	DH1	DH2
Critical path-based lower bound *LB0*	✔	✔	✔
Critical sequence lower bound *LBS*	✔		
Extended critical sequence lower bound LBS^{ext}	✔		
Node packing-based lower bound *LB3*		✔	
Resource-based lower bound *LBR*			✔
Theorem 1	✔	✔	
Theorem 2	✔	✔	
Immediate scheduling			✔
Theorem 3	✔	✔	✔
Theorem 4	✔	✔	
Theorem 4 – New			✔
Preprocessing			✔

are supplemented by an extended critical sequence lower bound LBS^{ext} which is computed by repetitively looking at a path of unscheduled, non-critical activities in combination with a critical path. The LBS^{ext} calculation starts by calculating the Stinson critical sequence lower bound. This allows the procedure to determine which activities cannot be scheduled within their slack time. Subsequently, all paths consisting of at least two unscheduled, non-critical activities, which start and finish with an activity that cannot be scheduled within its slack time, are constructed. A simple type of dynamic programming then allows for the calculation of the extended critical sequence bound for every non-critical path. Subsequent research revealed that the use of the cutset dominance rule combined with a bounding argument based solely on *LB0* leads to an improved performance. As a result, both *LBS* and LBS^{ext} have been removed from the procedure.

Recently, Mingozzi *et al.* [43] developed five new lower bounds (*LB1*, *LB2*, *LBP*, *LBX* and *LB3*), derived from different relaxations of a new mathematical programming formulation for the RCPSP. Bounds *LB1*, *LB2*, *LBX* and *LB3* dominate *LB0* and all prove to be tighter than *LBS* on the 110 RCPSP instances assembled by Patterson [48] and the 480 KSD instances. The most promising lower bound, *LB3*, is implemented by using a heuristic for solving a set packing problem. Demeulemeester and Herroelen [41] have incorporated their own version of *LB3* in the new DH-procedure based on the following heuristic. For each activity $i \in A$ they determine its possible *companions*, i.e., the activities with which it can be scheduled in parallel, respecting both the precedence and resource constraints. All unscheduled activities i with a non-zero duration are then entered in a list L in non-decreasing order of the number of companions (non-increasing duration as a tie-breaker). The following procedure then yields a lower bound, *LB3*, for the partial schedule under consideration:

LB3: = the earliest completion time of the activities in progress
while list L not empty **do**

Take activity j on top of list L
$LB3 := LB3 + d_j$
Remove activity j and its companions from list L

enddo

It is clear that other (more computationally intensive) heuristics can be used to calculate the lower bound *LB3*. The procedure described here is very fast and offers an excellent trade-off between tightness of the bound and the required computational effort. It generally improves the critical path lower bound, *LB0*, if there are pairs of activities that can be scheduled in parallel taking into consideration the precedence constraints only, but cannot be scheduled in this manner if resource constraints are taken into consideration.

As indicated in Table 3, the logic of the *DH2-procedure* differs from the logic used by DH1 in the additional use of a new resource-based lower bound *LBR*, and an improved immediate scheduling rule for putting eligible activities in progress, which replaces the rules described in Theorem 1 and 2. Other differences boil down to the use of 64 Mb of addressable memory, a new version of the cutset dominance rule (based on a more effective way of storing only efficient cutsets), some preprocessing and additional code polishing.

2.4. Problem complexity and the prediction of the computational requirements

As mentioned earlier, extensive computational experience with the optimal solution procedures for the RCPSP has been gained on different test sets of problem instances: the 110 Patterson problem set and the 480 KSD problem set. Ideally such a set should span the full range of complexity, from very easy to very hard problem instances. The generation of easy and hard problem instances, however, appears to be a very difficult task which heavily depends on the possibility to isolate the factors that precisely determine the computing effort required by the solution procedure used to solve a problem, and the calibration of the scale that characterizes such effort. The 110 test problems, assembled by Patterson [48], are a collection from different sources and have not been generated by using a controlled design of specified problem parameters. The 480 KSD instances used by Kolisch *et al.* [53] have been generated using the problem generator *ProGen* through the use of a controllable set of specified problem parameters. Recently, *ProGen* has been used to generate thousands of additional test instances [56], which have made it possible to gain additional important insight in the factors that seem to determine the complexity (in terms of the required computation time) of an RCPSP instance.

2.4.1. The relation between problem hardness and topological network structure. Using ProGen, De Reyck and Herroelen [57] generated five sets of 1000 RCPSP instances, each with 25 activities and 3 resource types. In each of the five sets, the *coefficient of network complexity, CNC*, is set at a different value, varying from 1.5 in the first set to 2.5 in the fifth. Each RCPSP instance was then solved using the DH-procedure.

The CNC is undoubtedly one of the most popular "measures of network complexity". Introduced by Pascoe [58] for activity-on-the-arc networks, and simply defined as the ratio of the number of arcs over the number of nodes, the measure has been adopted in a number of studies since then [59,60,48,61,53]. As observed by Kolisch *et al.* [53], in the activity-on-the-node representation, "complexity" has to be understood in the way that for a fixed number of activities (nodes), a higher complexity results in an increasing number of arcs and therefore in a greater connectedness of the network. A number of studies in the literature [62,53] seem to confirm that problems become easier with increasing values of the CNC, which makes the use of the CNC somewhat confounding (Elmaghraby and Herroelen [63] already questioned the use of the CNC). Both Alvarez-Valdes and Tamarit [62] and Kolisch *et al.* [53] observe a negative correlation between the CNC and the required solution time for solving an RCPSP instance. De Reyck and Herroelen [57] reach the conclusion that it is very ambiguous to attach all explanatory power of problem complexity to the CNC. They observed a positive correlation between the CNC and the so-called *complexity index, CI*. The complexity index, CI, is defined as the reduction complexity [64]; i.e. the minimum number of node reductions sufficient (along with series and parallel reductions) to reduce a two-terminal acyclic network to a single edge. Bein *et al.* [64] found that the CI plays an important role in predicting the required computing effort for solving an RCPSP instance (the higher the CI, the easier the RCPSP instance) and that the CI outperforms the CNC as a measure of network complexity (the CNC explains nothing extra above what is already explained by the CNC). The reason for the strong explanatory power attributed to the CNC in previous experiments performed in the literature is probably due to the fact that when the CNC was varied, other parameters (such as the CI) were varied also, which led to problems with significant differences in "complexity".

In a subsequent experiment, De Reyck [65] again found the CI to have a strong impact on the required processing time whereas the CNC had no impact at all. In addition, it was shown that an estimator for the so-called *restrictiveness* of a project network, namely *RT*, is a good network complexity measure for resource-constrained project scheduling problems. De Reyck [65] has shown that RT is actually identical to the *order strength, OS*, one of the best complexity measures for generating and evaluating assembly line balancing problems (see De Reyck and Herroelen [66]). OS is defined as the number of precedence relations, including the transitive ones, divided by $n(n-1)/2$, where n denotes the number of activities [67]. It is sometimes referred to as the *density* [68] and actually equal to 1 minus the *flexibility ratio*, defined by Dar-El [69] as the number of zero entries in the precedence matrix divided by the total number of matrix entries. De Reyck [65] reached the conclusion that OS absorbed the explanatory power of both the CNC and the CI, and that OS outperforms both measures.

2.4.2. The RCPSP and resource availability. De Reyck and Herroelen [57] have also tried to isolate the impact of the resource availability (or resource-constrainedness) on the required solution effort for solving the RCPSP. Elmaghraby and Herroelen [63] conjectured that the relationship between the hardness of a problem (as measured by the CPU-time required for its solution) and resource availability (scarcity) varies according to a bell-shaped curve. If resources are only available in extremely small amounts, there will be relatively little freedom in scheduling the activities. Hence, the corresponding RCPSP instance should be quite easy to solve. If, on the other hand, resources are amply available, the activities can be simply scheduled in parallel and the resulting project duration will be equal to the critical path length, leading again to a small computational effort.

Two of the best known parameters for describing resource availability (scarcity) that have been proposed in the literature are the resource factor and the resource strength. The *resource factor, RF* [58,62,70,53] reflects the average portion of resources requested per activity. If we have RF = 1, then each activity requests all resources. RF = 0 indicates that no activity requests any resource. The *resource strength, RS* [70] is redefined by Kolisch *et al.* [53] as $(a_k - r_k^{\min})/(r_k^{\max} - r_k^{\min})$, where a_k is the total availability of renewable resource type k, $r_k^{\min} = \max_{i=1,\ldots,n} r_{ik}$, and $r_k^{\max}$ is the peak demand of resource type k in the precedence-based earliest start schedule. Hence, with respect to one resource the smallest resource availability is obtained for RS = 0. For RS = 1, the problem is no longer resource-constrained. In their experiments, Kolisch *et al.* [53] conclude (in contradiction with Alvarez-Valdes and Tamarit [62])

that RS has the strongest impact on solution times: the average solution time continuously increases with decreasing RS. De Reyck and Herroelen [57], however, could not confirm the continuous increase of the required solution time with decreasing RS but found a bell-shaped relationship, in accordance with the conjecture of Elmaghraby and Herroelen [63].

Patterson [71] defines the *resource-constrainedness*, *RC*, for each resource k as p_k/a_k, where a_k is the availability of resource type k and p_k is the average quantity of resource k demanded when required by an activity. The arguments for using RC and not RS as a measure of network complexity are that (a) RC is a "pure" measure of resource availability in that it does not yet incorporate information about the precedence structure of a network, and (b) there are occasions where RS can no longer distinguish between easy and hard instances while RC continues to do so (for details, we refer to De Reyck and Herroelen [57]). Again, De Reyck and Herroelen [57] are able to confirm a bell-shaped relationship between the CPU-time and RC.

2.5. Branch-and-bound procedures for solving the RCPSP: conclusions

The fundamental conclusions which can be drawn from the reviewed research on branch-and-bound schemes for the RCPSP can be summarized as follows:

(a) A depth-first branch-and-bound search strategy based on resolving resource conflicts by delaying minimal subsets of activities is a clear favourite for optimally solving RCPSP instances;
(b) The cutset dominance rule ranks amongst the most effective dominance pruning rules, especially if a sufficient amount of memory can be used for storing the cutsets;
(c) The use of easy to compute and effective lower bounds (e.g. *LB3* and its possible variations; the new resource-based bound *LBR* incorporated in DH2) has a strong impact on the computational cost;
(d) It is extremely important to exploit the trade-off between the strength of the bounds or dominance rules used and the time required for their computation;
(e) Truncated depth-first branch-and-bound procedures provide a suitable alternative to priority based heuristics; near-optimal solutions can be obtained even if the truncated procedure is only allowed to run for a small amount of time;
(f) Sufficient attention should be devoted to efficient coding of the solution procedures used. Exploiting the full potential of 32-bit programming provided by recent compilers running on personal computer platforms such as Windows NT and OS/2 may add considerably to the efficiency of the computer code used (e.g. increased available memory allows for the storage of larger amounts of cutset dominance information; the use of 32-bit integers allows for computational speedups by combining for instance four resource types into one 32-bit integer when using 8 bits for every resource type);
(g) Reproducible optimal benchmark results are available on the 110 Patterson problems and the 480 KSD problems. In order to avoid computational bias and to guarantee that procedures are validated on a relevant spectrum of problem complexity (the complexity of a problem instance is entwined to the procedure used to solve it), computational experience should be reported on the complete problem sets and should not be limited to selected problem subsets assumed to be "hard" or "easy".

The above listed conclusions (a)–(f) reveal a number of (desirable) attributes of an efficient optimal solution procedure for the RCPSP. They constitute the spine of the solution logic used by the DH-procedure and its extensions and lie at the very basis of their computational efficiency. Moreover, the DH-solution logic can be extended to important related problem types which are summarized in Table 4. The table indicates how each of the problems mentioned in the table heading differ in the objective function used, the type of precedence relationships, the type, availability and requirements of the resources, the use of activity preemption, and the assumed trade-offs. Table 4 serves as a reference for our subsequent discussion.

3. THE PREEMPTIVE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM (PRCPSP)

In the RCPSP it is assumed that an activity once started, must be continuously processed until completion. In practice, however, it may be the case that the processing of an activity may be interrupted and resumed at a later time. When resource availability is limited, activity preemption may result in a shorter project duration. The introduction of activity preemption increases the number of possible

Table 4. Extensions to the basic RCPSP model

	RCPSP	PRCPSP	GRCPSP	RCPSP-GPR	DTRTP	MRCPSP	MAX-NPV PSP	RCPSP-DC	RCPSPDC-GPR
Objective	makespan	makespan	makespan	regular	makespan	regular	NPV	NPV	NPV
Precedence relations	FS(0)	FS(0)	P	GPR	FS(0)	FS(0)	GPR	FS(0)	GPR
Resources	R	R	R	R	R	R,N,D	none	R	R
Resource availability	constant	variable	variable	variable	constant	variable	–	constant	variable
Resource requirements	constant	constant	constant	variable	constant	constant	–	constant	variable
Preemption	no	yes	no	no	no	no	no	no	no
Time/resource trade-offs	no	no	no	no	yes	yes	no	no	no
Resource/resource trade-offs	no	no	no	no	no	yes	no	no	no

Key:
FS(0)=finish-start precedence relations with zero time lag;
P=precedence diagramming=SS, SF, FS, FF precedence relations with minimal time lag;
GPR=generalized precedence relations=SS, SF, FS, FF precedence relations with minimal and/or maximal time lag;
R=renewable resource types;
N=nonrenewable resource types;
D=doubly-constrained resource types.

solutions and consequently the computational complexity of the resource-constrained project scheduling problem.

The *PRCPSP* allows activities to be preempted at integer points in time; i.e., the fixed integer duration d_i of an activity may be split in $j=1,2,\ldots,d_i$ duration units. Each duration unit j of activity i is then assigned an integer finish time $f_{i,j}$. The variable $f_{i,0}$ denotes the earliest time that an activity i can be started. As only finish–start relations with a time lag of zero are allowed, $f_{i,0}$ equals the latest finish time of all the predecessors of activity i. An activity i belongs to the set S_t of activities in progress in period $]t-1,t]$ if one of its duration units $j=1,2,\ldots,d_i$ finishes at time t (i.e., if $f_{i,j}=t$). The PRCPSP can now be formulated as follows [72]:

$$\text{Min } f_{n,0} \tag{5}$$

subject to

$$f_{1,0}=0, \tag{6}$$

$$f_{j,0}\geq f_{i,d_i},\ \forall(i,j)\in H, \tag{7}$$

$$f_{i,j-1}+1\leq f_{i,j},\ i=1,2,\ldots,n;\ j=1,2\ldots,d_i, \tag{8}$$

$$\sum_{i\in S_t} r_{ik}\leq a_k,\ t=1,2\ldots f_{n,0};\ k=1,2\ldots,K. \tag{9}$$

The objective function (5) minimizes the makespan by minimizing the earliest start time of the dummy end activity which by assumption has a duration of 0. Activity 1 is assigned an earliest start time of 0 through (6). (7) assure that all precedence relations are satisfied: the earliest start time of an activity j cannot be smaller than the finish time of the last unit of duration of its predecessor i. (8) specify that the finish time for every unit of duration of an activity has to be at least one time unit larger than the finish time for the previous unit of duration, while (9) stipulate the resource constraints.

Slowinski [73] and Weglarz [74] presented optimal solution procedures for the case of continuous processing times for the different activities. Davis and Heidorn [32] developed an implicit enumeration scheme based on the splitting of activities in unit duration tasks. Kaplan [75,76] presented a dynamic programming formulation and suggests a solution by a reaching procedure.

The DH-procedure has been extended to the PRCPSP [77]. In order to do so, a distinction is made between *activities* and *subactivities*. At the start the procedure creates a new project network in which all activities are replaced by one or more subactivities. All activities are split into subactivities, their number being equal to the duration of the original activity, each having a duration of 1 and resource requirements that are equal to those of the original activity. Demeulemeester and Herroelen [77] prove that in order to solve the PRCPSP, it is sufficient to construct partial schedules by semi-active timetabling at the level of the subactivities. Theorems 1, 2, 3 and 4 stated above for the RCPSP can be extended for the PRCPSP. The authors also show that it is sufficient to consider only minimal delaying alternatives in order to resolve resource conflicts. In addition, they have shown that all three lower bounds discussed earlier (*LB0*, *LBS* and LBS^{ext}) remain applicable, at the trade-off of increased computational requirements. Therefore, they only included *LB0* in the code. *LB3*, which is extendible to the PRCPSP but was only developed very recently, could not be included at the time the code was written.

The literature on the PRCPSP is almost void and very little computational experience is available. Demeulemeester and Herroelen [77] have programmed their procedure in Turbo C Version 2.0 for a personal computer IBM PS/2 Model 70. On the same 41 Patterson test problems used by Kaplan [75,76] and using a similar PC running at 16 MHz, it finds the optimal solution in an average CPU-time of 4.99 s, while the Kaplan code requires an average of 425 s. Using a personal computer IBM PS/2 running at 25 MHz, they have tested their algorithm on all 110 Patterson test problems. All problems could be solved within 5 min of CPU-time, requiring an average of 6.90 s.

Demeulemeester [72] has extended the code for the PRCPSP with *variable resource availabilities*. In that case, the adapted versions of Theorems 1 and 2 no longer apply. A total of 107 out of the 110 Patterson test problems, modified by Simpson and Patterson [78] to incorporate variable resource availabilities, could be solved with an average computation time of 12.63 s.

4. PROJECT SCHEDULING UNDER GENERALIZED PRECEDENCE RELATIONS

A lot of research efforts have been directed towards relaxing the strict precedence assumption of CPM/PERT. The resulting types of precedence relations are often referred to as MPM (Metra Potential Method) precedence constraints [79,80], precedence diagramming [81], time windows [37], minimal and maximal time lags [82–87], and generalized precedence constraints [88]. In accordance with Elmaghraby and Kamburowski [89], we denote them as generalized precedence relations (GPRs) and distinguish between start–start (*SS*), start–finish (*SF*), finish–start (*FS*) and finish–finish (*FF*).

GPRs can specify a minimal or maximal time lag between any 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. Many practical situations can be modelled using GPRs, such as activity ready times and deadlines, permissible and mandatory activity overlaps, time-varying resource requirements and availabilities, overlapping production activities and set-up times.

4.1. The generalized resource-constrained project scheduling problem (GRCPSP)

Demeulemeester and Herroelen [90] have extended the DH-procedure to the case of *minimal* time lags, activity release dates and deadlines and variable resource availabilities. The resulting problem, denoted as the GRCPSP in Table 4, can be conceptually formulated as follows:

$$\text{Min } f_n \tag{10}$$

subject to

$$f_1 = 0, \tag{11}$$

$$f_j - d_j \geq f_i - d_i + SS_{ij}, \ \forall (i,j) \in H_1, \tag{12}$$

$$f_j \geq f_i - d_i + SF_{ij}, \ \forall (i,j) \in H_2, \tag{13}$$

$$f_j - d_j \geq f_i + FS_{ij}, \ \forall (i,j) \in H_3, \tag{14}$$

$$f_j \geq f_i + FF_{ij}, \ \forall (i,j) \in H_4, \tag{15}$$

$$f_i - d_i \geq g_i, \ i = 1,2\ldots,n, \tag{16}$$

$$f_i \leq h_i, \ i = 1,2\ldots,n, \tag{17}$$

$$\sum_{i \in S_t} r_{ik} \leq a_{kt}, \ t = 1,2,\ldots f_n; \ k = 1,2\ldots,K, \tag{18}$$

where H_1, H_2, H_3, H_4 are sets of pairs of activities indicating precedence relations of the type SS_{ij}, SF_{ij}, FS_{ij}, and FF_{ij}, respectively; g_i is the ready time of activity i, h_i is the deadline of activity i, and a_{kt} is the availability of resource type k during period $]t-1,t]$.

The objective function (10) is to minimize the project duration by minimizing the finish time of the unique dummy end activity n. (11)(12)(13)(14) ensure that the various types of precedence constraints are satisfied. (15) assigns the dummy start activity 1 a completion time of 0. (16) guarantee that the ready times are respected, while (17) guarantee that no deadlines are violated. (18) specify that the resource utilization during any time interval $]t-1,t]$ does not exceed the resource availability levels during that

time interval for any of the resource types.

In order to extend the DH-procedure to the GRCPSP, all precedence constraints are converted to finish–start precedence relations using the following conversion formula:

$$FS_{ij}' = \max\{SS_{ij} - d_i,\ SF_{ij} - d_i - d_j,\ FS_{ij},\ FF_{ij} - d_j\}. \tag{19}$$

The *ready time* g_i of an activity i can easily be transformed into a finish–start relation between the dummy start activity 1, which starts and finishes at time 0, and activity i itself:

$$FS_{1i}'' = \max\{g_i,\ FS_{1i}'\}. \tag{20}$$

Coping with *deadlines* h_i is somewhat more involved. For every activity j a latest allowable start time ls_j has to be computed such that whenever this activity j is delayed to start later than ls_j, the deadline of this activity or of one of its direct or indirect successors is exceeded even if all subsequent activities were scheduled as soon as possible without considering the resource constraints. Consequently, if during the branch-and-bound procedure an activity j is assigned an early start time s_j that exceeds its latest allowable start time ls_j, backtracking can occur as no feasible solution can be found by continuing the search from this partial schedule.

Demeulemeester and Herroelen [90] prove that the partial schedules may be constructed by semi-active timetabling. In addition, they show that it is sufficient to consider only minimal delaying alternatives in order to resolve a resource conflict. Last but not least, they extend the left-shift and cutset dominance rules (Theorems 3 and 4). They also show that the critical sequence bound *LBS* and the extended critical sequence bound LBS^{ext} cannot be extended, leaving the remaining critical path length *LB0* as a possible lower bound (again *LB3* can be extended to the GRCPSP but was not yet known when writing the code).

The literature on the GRCPSP is very limited and a standard set of test problems has not yet been established. Demeulemeester and Herroelen [90] have coded the *GDH-procedure* in Turbo C Version 2.0 for IBM PS/2 computers with 80486 processor operating at 25 MHz (or compatibles). The procedure was then tested on the 110 Patterson test problems as modified by Simpson and Patterson [78] to incorporate variable resource availabilities. The GDH-procedure could find the optimal solution for all 110 problems with constant resource availabilities in an average of 0.14 s. For the problems with variable resource availabilities, the GDH-procedure, when given a time limit of 10 min, could optimally solve 109 out of the 110 problems to optimality in an average CPU-time of 8.11 s (vis-à-vis 100.85 s required on average by Simpson's serial procedure to solve 97 problems and 96.63 s required on average by Simpson's parallel procedure to solve 98 problems). When the code is allowed to run to completion, all problems are solved in an average of 60.16 s. As such, the GDH-procedure seems to be a very efficient and effective exact solution procedure for the GRCPSP. In addition, the computational experience obtained indicates that moderate changes in the ready times or in the resource availabilities do not have a significant impact on the computation times. The introduction of deadlines significantly reduces the solution time required. Allowing activity overlaps (negative FS_{ij} values) causes a strong increase in the required computation time.

In the next section it is shown that a modification of the delaying scheme allows the DH-procedure to be extended to the case of resource-constrained project scheduling with minimal *and* maximal time lags.

4.2. The resource-constrained project scheduling problem with generalized precedence relations (RCPSP-GPR)

The resource-constrained project scheduling problem with generalized precedence relations (problem RCPSP-GPR in Table 4) allows for start–start, finish–start, start–finish and finish–finish constraints with minimal *and* maximal time lags. The minimal and maximal time lags between two activities i and j have the form:

$$s_i + SS_{ij}^{\min} \le s_j \le s_i + SS_{ij}^{\max};\ s_i + SF_{ij}^{\min} \le f_j \le s_i + SF_{ij}^{\max};$$
$$f_i + FS_{ij}^{\min} \le s_j \le f_i + FS_{ij}^{\max};\ f_i + FF_{ij}^{\min} \le f_j \le f_i + FF_{ij}^{\max}.$$

The different types of GPRs can be represented in a *standardized form* by reducing them to just one type, e.g. the minimal start–start precedence relations, using the following transformation rules [37]:

$$s_i+SS_{ij}^{\min}\leq s_j \Rightarrow s_i+l_{ij}\leq s_j \text{ with } l_{ij}=SS_{ij}^{\min};$$

$$s_i+SS_{ij}^{\max}\geq s_j \Rightarrow s_j+l_{ji}\leq s_i \text{ with } l_{ji}=-SS_{ij}^{\max};$$

$$s_i+SF_{ij}^{\min}\leq f_j \Rightarrow s_i+l_{ij}\leq s_j \text{ with } l_{ij}=SF_{ij}^{\min}-d_j;$$

$$s_i+SF_{ij}^{\max}\geq f_j \Rightarrow s_j+l_{ji}\leq s_i \text{ with } l_{ji}=d_j-SF_{ij}^{\max};$$

$$f_i+FS_{ij}^{\min}\leq s_j \Rightarrow s_i+l_{ij}\leq s_j \text{ with } l_{ij}=d_i+FS_{ij}^{\min};$$

$$f_i+FS_{ij}^{\max}\geq s_j \Rightarrow s_j+l_{ji}\leq s_i \text{ with } l_{ji}=-d_i+FS_{ij}^{\max};$$

$$f_i+FF_{ij}^{\min}\leq f_j \Rightarrow s_i+l_{ij}\leq s_j \text{ with } l_{ij}=d_i-d_j+FF_{ij}^{\min};$$

$$f_i+FF_{ij}^{\max}\geq f_j \Rightarrow s_j+l_{ji}\leq s_i \text{ with } l_{ji}=d_j-d_i+FF_{ij}^{\max}.$$

Conceptually, the RCPSP-GPR can then be formulated as follows:

$$\text{Minimize } s_n \tag{21}$$

Subject to

$$s_i+l_{ij}\leq s_j,\ \forall(i,j)\in E, \tag{22}$$

$$\sum_{i\in S(t)} r_{ik}\leq a_{kt},\ k=1,2,\ldots,m;\ t=1,2,\ldots,T, \tag{23}$$

$$s_1=0, \tag{24}$$

$$s_i\in N,\ i=1,2,\ldots,n. \tag{25}$$

The objective function given in (21) minimizes the project duration, given by the starting time (or finishing time: $d_n=0$) of the dummy activity n. The precedence constraints are denoted in standardized form by (22). (23) represent the resource constraints. The resource requirements are assumed to be constant over time, although this assumption can be relaxed using GPRs without having to change the solution procedures. (24) forces the dummy start activity to begin at time zero and (25) ensure that the activity starting times assume nonnegative integer values. Once started, activities run to completion.

The RCPSP-GPR is known to be strongly NP-hard, and even the feasibility problem, i.e. the problem of testing whether an RCPSP-GPR instance has a feasible solution, is NP-complete [37]. To the best of our knowledge, the only optimal solution procedures for the RCPSP-GPR are the branch-and-bound algorithm of Bartusch *et al.* [37] and De Reyck and Herroelen [92]. The computational experience obtained with the procedure of Bartusch *et al.* [37] is limited to a single bridge construction project and the computer code is no longer available [91]. The procedure is a depth-first type branch-and-bound procedure which is based on the concept of a *forbidden set*, i.e. a set of activities which may never be scheduled in parallel because a violation of the resource constraints would result. Such a set is called *minimal* if no subset of that set constitutes a forbidden set in itself. Moreover, a minimal forbidden set is labelled a *reduced* forbidden set if the activities belonging to that set can be scheduled in parallel without violating the (generalized) precedence constraints between them. The procedure starts with computing the earliest start schedule and consequently adds new precedence relations between activities in order to eliminate the reduced forbidden sets until no such set is scheduled in parallel in the earliest start schedule.

The procedure of De Reyck and Herroelen [92] is a branch-and-bound procedure based on the concepts of minimal delaying alternatives as developed by Demeulemeester and Herroelen [40] for the RCPSP. The nodes in the search tree represent the initial project network, described by a matrix $D=[d_{ij}]$, where d_{ij} represents the longest path length between activities i and j, extended with extra (zero-lag finish–start) precedence relations to resolve a resource conflict present in the parent node, which results in an *extended* matrix D'. Nodes which represent time-feasible but resource-infeasible project networks and which are not fathomed by any 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* [40]. For the RCPSP-GPR, the concept of a minimal delaying alternative is extended to the concept of a *minimal delaying mode* as follows. Each of the minimal delaying alternatives is delayed (enforced by extra zero-lag finish-start precedence relations $i\prec j$, implying $s_i+d_i\leq s_j$) 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). In this way, each delaying alternative can give rise to several delaying modes. In general, the

delaying set D, i.e. the set of all minimal delaying alternatives, is equal to:

$$D=\Big\{ D_d \mid D_d \subset S(t^*) \text{ and } \forall \text{ resource type } k: \sum_{i\in S(t^*)} r_{ik} - \sum_{i\in D_d} r_{ik} \le a_k \text{ and } \forall D_{d'} \in D\backslash\{D_d\} : D_{d'} \not\subset D_d \Big\}.$$

The set of minimal delaying modes equals: $M=\{M_m \mid M_m=\{k<D_d\}, k\in S(t^*)\backslash D_d, D_d\in D\}$. Activity k is called the *delaying activity*: $k<D_d$ implies that $k<l$ for all $l\in D_d$. For the example described in Section 2.2 (see Fig. 3), the 8 resulting delaying modes are depicted in Fig. 4. Now, the operator "<" does not represent a temporal constraint as in the DH-procedure, but denotes a zero-lag finish–start precedence constraint (which is an important distinction because in the RCPSP-GPR, contrary to the RCPSP, the delaying activity can again be delayed later on).

Each minimal delaying mode is examined for time-feasibility and, if feasible, evaluated by computing the critical path based lower bound *LB0*. Each time-feasible minimal delaying mode with a lower bound $LB0<T$ is then considered for further branching, which occurs from the node with smallest *LB0*. If the node represents a project network in which a resource conflict occurs, a new branching occurs. The procedure is of the depth-first type, i.e. 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. Nodes are fathomed when they represent a time-infeasible network or when *LB0* exceeds or equals *T*. The following two dominance rules rely on the identification of redundant delaying alternatives and redundant delaying modes:

Theorem 5. *If there exists a minimal delaying alternative D_d with activity $i\in D_d$ but its successor $j\notin D_d$ ($d_{ij}\ge 0$), D_d can be extended with activity j. If the resulting delaying alternative becomes non-minimal as a result of this operation it may be eliminated from further consideration.*

Theorem 6. *When a minimal delaying alternative D_d gives rise to two delaying modes with delaying activities i and j, the latter mode is dominated by the former iff $d_{ij}+d_j\ge d_i$.*

In addition, the following precedence subset dominance rule is incorporated in the procedure:

Theorem 7. *If the set of added precedence constraints which leads to the project network (in the form of an extended distance matrix) in node x contains as a subset another set of precedence constraints leading to the project network (extended distance matrix) in a previously examined node y in another*

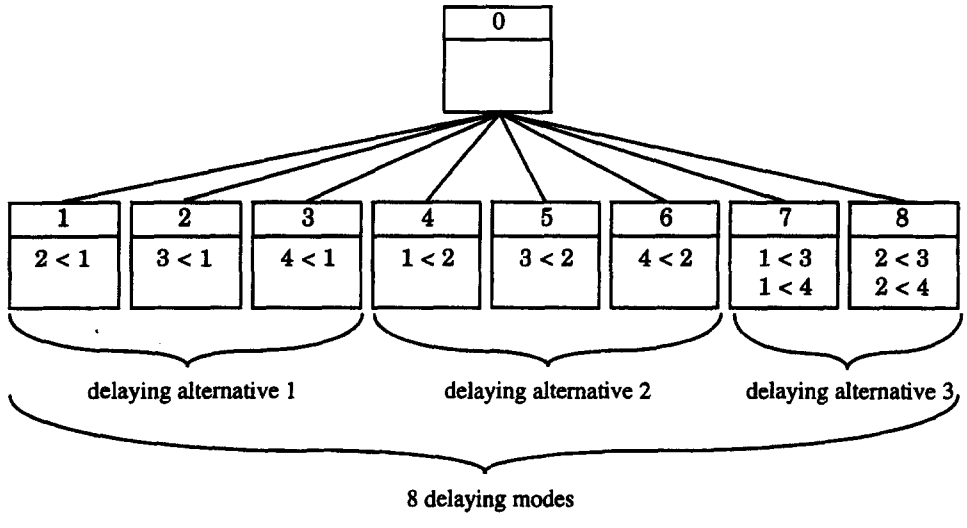

Fig. 4. The concept of minimal delaying modes.

branch of the search tree, node x can be fathomed.

Before initiating the branch-and-bound procedure, the solution space can be reduced by the following preprocessing rule:

Theorem 8. *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.*

The bounding argument involves the use of a lower bound $LB3^g$ which is computed using an extended version of the procedure of Demeulemeester and Herroelen [41] for computing the lower bound *LB3*.

The procedure has been programmed in Microsoft Visual C++ 2.0 under Windows NT for use on a Pentium 60 MHz personal computer with 16 Mb of internal memory. It has been validated on 550 RCPSP-GPR instances, generated from the problem set for the RCPSP assembled by Patterson [48]. A detailed analysis reveals 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 and the higher the number of minimal time lags that allow for activity overlaps, the more efficient the procedure. In order to test the performance of a truncated version of the procedure, an experiment was performed in which the procedure was run until (a) the first feasible solution was found, (b) for 1 s, and (c) for 10 s. The average deviation from the optimum for these three cases was 4.6%, 0.8% and 0.1%, respectively. When many (tight) maximal time lags are present and the minimal time lags allow for activity overlaps, the average deviation decreases to 3.75%, 0% and 0%, respectively.

De Reyck and Herroelen [92] also report computational experience on three different problem sets generated using the problem generator *ProGen/max* developed by Schwindt [85]. The first set consists of 1080 instances involving 100 activities and 5 resource types, satisfying a variety of pre-set parameters. The second set of 100 activity problems consists of 1440 instances with up to 8 resource types generated by Franck and Neumann [86]. De Reyck and Herroelen [92] use a third set of 7200 instances ranging in problem size from 10 up to 100 activities with a requirement for 5 resource types. They show that a truncated version of the procedure outperforms a combination of the best heuristic procedures available [82,83,86,87].

5. MAXIMIZING THE NET PRESENT VALUE IN PROJECT NETWORKS

Originally, the focus of most project scheduling problems was on minimizing the project makespan. Minimizing the project duration is an example of a *regular* objective function. A regular objective function is a nondecreasing function of the activity completion times, i.e. when the activity completion times increase, the objective function value increases (or remains the same). Other regular objective functions include the minimization of total project costs including tardiness penalties with respect to activity and project due dates. This objective function is often used to model the multi-project scheduling problem, in which multiple projects are to be scheduled simultaneously. In recent years, researchers have concentrated on *non-regular* objective functions for which the condition above does not hold. This implies that delaying activities may improve the performance of the schedule, even if such a delay is not imposed by resource or other (temporal or precedence) constraints. Non-regular objective functions include minimizing the weighted earliness-tardiness of the activities (project) relative to activity (project) due dates, and maximizing the net present value (NPV) of the project. Especially the latter objective has become increasingly popular for coping with financial considerations in project scheduling.

The majority of the NPV-contributions assume a completely *deterministic* project setting, in which all relevant problem data, including the various cash flows, are assumed known from the outset. Research efforts have led to optimal procedures for the *unconstrained* project scheduling problem, where activities are only subject to precedence constraints (problem MAX-NPV in Table 4). In addition, numerous efforts aim at providing optimal or suboptimal solutions to the project scheduling problem under various types of resource *constraints* (among them problems RCPSP-DC and RCPSPDC-GPR mentioned in Table 4), using a rich variety of often confusing assumptions with respect to network representation (activity-on-the-node versus activity-on-the-arc), cash flow patterns (positive and/or negative, event-oriented or activity-based), and resource constraints (capital, constrained, different resource types, materials considerations, time/cost trade-offs). A number of efforts focus on the simultaneous determination of both the amount and timing of payments. Last, a modest start has been taken in tackling the *stochastic*

aspects of the scheduling problem involved. For a recent review of the vast literature and a categorization of the solution procedures, we refer the reader to Herroelen *et al.* [93].

5.1. The MAX-NPV problem

The deterministic unconstrained MAX-NPV problem, which is sometimes denoted as the payment scheduling problem (hence the notation MAX-NPV, PSP in Table 4), can be described as follows. The project is represented in activity-on-the-node format by a network $G=(V,E)$ where the set of nodes, V, represents activities and the set of arcs, E, represents finish–start precedence constraints with a time lag of zero. We assume, without loss of generality, that there is a single dummy start node 1 and a single dummy end node $n=|V|$. The problem is unconstrained in that no constraints are imposed on the use of resources. The activities have a fixed duration, d_i $(i=1,2,\ldots,n)$, and the performance of each activity involves a series of cash flow payments and receipts throughout the activity duration. A terminal value of each activity upon completion can be calculated by compounding the associated cash flows to the end of the activity as follows:

$$C_i = \sum_{t=1}^{d_i} F_{it} e^{\alpha(d_i - t)}, \tag{26}$$

where C_i denotes the terminal value of cash flows in activity i at its completion, F_{it} represents the cash flows for activity i in period t, $t=1,2,\ldots,d_i$ and α is the discount rate.

A conceptual formulation of the deterministic *unconstrained MAX-NPV problem* may now look as follows:

$$\text{Max} \sum_{i=1}^{n} q_{f_i} C_i \tag{27}$$

subject to

$$f_i \leq f_j - d_i, \forall (i,j) \in H, \tag{28}$$

$$f_n \leq T, \tag{29}$$

where q_t denotes the factor for discounting over t periods to period zero; i.e., $q_t=\exp(-\alpha t)$ and T is the project deadline.

The objective (27) is to maximize the net present value of the project. The constraint set given in (28) maintains the finish–start precedence relations among the activities. The final constraint (29) limits the project duration to a negotiated project deadline.

Several procedures have been presented for solving the MAX-NPV problem. Russell [94] proposed a successive approximation approach using the first term of a Taylor series expansion of the objective function and a dual formulation that he showed to be a transshipment problem. Grinold [95] transformed this problem into a linear problem and developed two solution procedures which exploit the solution logic of a weighted distribution problem. Elmaghraby and Herroelen [96] and Herroelen and Gallens [97] describe a solution procedure based on the intuitive argument that positive cash flows should be scheduled as early as possible and negative cash flows as late as possible.

The most efficient algorithm is due to Demeulemeester *et al.* [98]. The algorithm starts by computing the earliest completion time for the activities based on traditional forward pass critical path calculations and determines the corresponding *early tree* in the network which spans all activities (nodes) scheduled at their earliest completion times and which corresponds to a feasible solution with a project duration equal to the critical path length (the arcs of the early tree denote the binding precedence relations). The algorithm then builds the *current tree* by delaying, in reverse order, all nodes with a negative cash flow as much as possible within the early tree; i.e., by linking them to their earliest starting successor. Using the dummy node 1 as the search base, the algorithm will enter a *recursive search* of the current tree to identify partial trees that might be shifted forwards (away from time zero) in order to increase the *NPV* of the project. Due to the structure of the recursive search it can never happen that a backward shift (towards time zero) of a partial tree can lead to an increase in the *NPV* of the project: any partial tree that is not scheduled at its earliest starting point has a negative *NPV* and should be scheduled as late as possible. When a partial tree is the subject of a forward shift, the algorithm computes its minimal displacement interval and updates the current tree. Upon a shift, the algorithm repeats the recursive search on the current tree associated with the new feasible solution. During the search, it is possible that

the current tree disconnects into two parts, one part being shifted forward till it hits the deadline. If this happens, repetitively performing the recursion will only further optimize the tree connected to node 1, since the tree connected to node n is already optimal. The algorithm stops when no partial trees can be shifted that increase the *NPV* of the project.

The procedure has been programmed in Microsoft® Visual C++ 2.0 under Windows NT for use on a Digital Venturis Pentium 60 MHz personal computer. Computational tests on two data sets (98 test problems adapted from the 110 Patterson problem set [48] and 1980 networks adapted from the De Reyck and Herroelen set of ALB test problems [66]) reveal that the recursive search algorithm is very efficient. It finds the optimal solution in an average of 0.43 ms for the Patterson set and 0.42 ms for the ALB problem set. It outperforms Grinold's procedure in that it is on the average 2.5 times faster on the Patterson set and 2.6 times faster on the ALB set at a much smaller CPU-time variance.

De Reyck and Herroelen [99] recently extended the Demeulemeester *et al.* [98] procedure to cope with the above discussed generalized precedence relations, which introduce arbitrary minimal and maximal time lags between the start and completion of activities. For the set of 7200 problem instances generated for the RCPSP-GPR [92] by ignoring the resource requirements and using uniformly generated cash flows in the interval $[-500, +500]$, the required CPU-times are very small (average computation times are smaller than 1 s, even for the 100-activity projects, with a maximum of 1.6 s). The number of activities has a strong impact on the required computation time. Moreover, there is a positive correlation between the order strength (OS) and the required CPU-time: when OS increases, the problem becomes harder.

5.2. The RCPSP with discounted cash flows (RCPSP-DC)

Adding renewable resource constraints to the model of (27)(28)(29) yields the NP-hard [100] *resource-constrained project scheduling problem with discounted cash flows*. Icmeli and Erengüç [101] present the only branch-and-bound procedure for the resource-constrained MAX-NPV problem currently available. The project due date T is obtained as $T=s*D$, where D is the project duration obtained from a heuristic solution procedure, and s is a constant greater than 1. The branch-and-bound procedure is to be considered an extension of the *DH-procedure* [40] for solving the RCPSP. At each node of the search tree a complete schedule (which may be resource infeasible) is obtained. At the initial node of the tree, an optimal solution to the corresponding unconstrained MAX-NPV problem is obtained using the fixed deadline algorithm of Grinold [95], yielding an upper bound. If this solution is resource feasible the procedure terminates. If not, branching is done using the modified version of the delaying scheme used by Demeulemeester and Herroelen [40] to resolve resource conflicts, as described in the previous section. This modification is necessary because semi-active timetabling which starts activities as early as possible within the given constraints is inappropriate under the non-regular NPV-objective.

The subproblems generated by the branching process are solved using Grinold [95]. A node is fathomed either if the optimal unconstrained solution has a project duration exceeding the due date, or if it is less than or equal to that of the incumbent solution. The node with the greatest objective function value is selected for further branching.

The algorithm is written in Fortran and run on an IBM3090 computer with vector processing. The code was validated on a problem set derived from some of the Patterson [48] problems and some of the problems generated by Kolisch *et al.* [53]. The algorithm was shown to outperform the integer programming procedure developed by Yang *et al.* [102]. It is to be expected that computational gains can be obtained from solving the subproblems using the optimal recursive search algorithm of Demeulemeester *et al.* [98] for solving the MAX-NPV problem instead of Grinold's procedure.

Recently, De Reyck and Herroelen [103] presented an optimal solution procedure for the resource-constrained project scheduling problem with discounted cash flows *and* generalized precedence relations (problem RCPSPDC-GPR in Table 4). This depth-first branch-and-bound algorithm, the only optimal procedure currently available, uses the MAX-NPV procedure of De Reyck and Herroelen [92] for the computation of upper and lower bounds. Extensive computational experience indicates that a truncated version of the procedure yields very promising results.

6. DISCRETE TRADE-OFFS IN PROJECT SCHEDULING

Various types of trade-offs occur in project scheduling practice and have been studied in the context of project scheduling. In this section we focus on discrete time/resource and resource/resource trade-offs.

6.1. The discrete time/resource trade-off problem (DTRTP)

In the RCPSP, each activity has a *single* execution *mode*: both the activity duration and its requirements for a set of renewable resources are assumed to be fixed. Herroelen [104] and Elmaghraby [105] were the first authors to deal with discrete time/resource trade-offs and, correspondingly, multiple ways for executing the project activities. In many real-life construction and software development projects, it often occurs that only one renewable bottleneck resource is available (e.g. labor) in constant amount throughout the project. During the project planning phase, project management traditionally relies on the work breakdown structure to specify work packages and to estimate the work content (e.g. amount of man-days) for each individual activity. In practice, several scenarios are available for the execution of the individual activities. Given the estimated work content for an activity, a set of allowable execution modes can be specified for its execution, each characterized by a fixed duration (e.g. days) and a constant resource requirement (e.g. units/day), the product of which is at least equal to the activity's specified work content.

The problem that arises in such project scheduling environments is referred to as the *discrete time/ resource trade-off problem* (problem *DTRTP* in Table 4). In the DTRTP, the duration of an activity is assumed to be a discrete, non-increasing function of the amount of a single *renewable* resource committed to it. Given the specified work content W_i for activity i ($1 \leq i \leq n$), all M_i efficient execution modes for its execution are determined based on time/resource trade-offs. Activity i when performed in mode m ($1 \leq m \leq M_i$) has a duration d_{im} and requires a constant amount r_{im} of the renewable resource during each period it is in progress, such that $r_{im}d_{im}$ is at least equal to W_i. A mode is called efficient if there is no other mode with equal or smaller duration and smaller resource requirement or equal resource requirement and smaller duration. Without loss of generality, we assume that the modes of each activity are sorted in the order of non-decreasing duration. The single renewable resource has a constant per period availability a. We assume that the dummy start node 1 and the dummy end node n have a single execution mode with zero duration and zero resource requirement. The objective is to schedule each activity in one of its modes, subject to the finish–start precedence and the renewable resource constraint, under the objective of minimizing the project makespan. Introducing the decision variables

$$x_{imt} = \begin{cases} 1, & \text{if activity } i \text{ is performed in mode } m \text{ and started at time } t, \\ 0, & \text{otherwise,} \end{cases}$$

the DTRTP can be formulated as follows:

$$\text{Min} \sum_{t=e_n}^{l_n} t x_{n1t} \tag{30}$$

subject to

$$\sum_{m=1}^{M_i} \sum_{t=e_i}^{l_i} x_{imt} = 1, \; i = 1,2,\ldots,n, \tag{31}$$

$$\sum_{m=1}^{M_i} \sum_{t=e_i}^{l_i} (t + d_{im}) x_{imt} \leq \sum_{m=1}^{M_j} \sum_{t=e_j}^{l_j} t x_{jmt}, \; (i,j) \in H, \tag{32}$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{im} \sum_{s=\max\{t-d_{im},e_i\}}^{\min\{t-1,l_i\}} x_{ims} \leq a, \; t = 1,2,\ldots,T, \tag{33}$$

$$x_{imt} \in \{0,1\}, \; i = 1,2,\ldots,n; \; m = 1,2\ldots,M_i; \; t = 0,1\ldots,T, \tag{34}$$

where e_i (l_i) is the critical path based earliest (latest) start time of activity i based on the modes with the smallest duration, T is the upper bound on the project duration, and H is the set of precedence related activities.

The objective function (30) minimizes the makespan of the project. Constraint set (31) ensures that each activity is assigned exactly one mode and exactly one start time. Constraints (32) denote the precedence constraints. Constraints (33) secure that the per period availability of the renewable resource is met. Finally, constraints (34) force the decision variables to assume 0-1 values.

The DTRTP is a subproblem of the *multi-mode resource-constrained project scheduling problem* (*MRCPSP*), which includes, next to time/resource trade-offs, time/cost and resource/resource trade-offs. The MRCPSP also allows for multiple renewable, nonrenewable and doubly-constrained resources (limited on a per period basis and a total project basis) and a variety of objective functions [55]. As a

generalization of the RCPSP, the DTRTP is NP-hard.

Demeulemeester *et al.* [106] present a branch-and-bound procedure for the DTRTP based on the concept of maximal activity-mode combinations, i.e. subsets of activities executed in a specific mode. At each decision point t (corresponding to the completion time of one or more activities), the branch-and-bound procedure evaluates the feasible partial schedules PS_t (which correspond to nodes in the search tree) obtained by enumerating all *feasible maximal* (i.e. subset-maximal) *activity-mode combinations*. Activity-mode combinations are *feasible* if the activities can be executed in parallel in the specified mode without resulting in a resource constraint violation. They are *maximal* when no other activity can be added in one of its modes without causing a resource conflict.

Each partial schedule is evaluated using a precedence-based and a resource-based lower bound. The node with the smallest lower bound is selected for further branching at the next decision point t'. At time t' again all feasible maximal activity-mode combinations are enumerated under the assumption that earlier scheduled activities can be removed from the partial schedule. If they are not removed from the partial schedule, another mode can be selected provided that they are restarted at time t' and terminate earlier than in their previous mode. Backtracking occurs when a schedule is completed or when a branch is fathomed by one of the proposed dominance rules. The procedure stops with the optimal solution upon backtracking to level 0 in the search tree. The procedure uses several dominance rules, including a single-mode left-shift rule, a multi-mode left-shift rule and a cutset dominance rule which are extensions of similar rules originally developed for the RCPSP.

The procedure has been programmed in Microsoft® Visual C++ 4.0 under Windows NT for use on a Pentium Pro 200 MHz personal computer. Computational experimentation using a full factorial experiment on a randomly generated problem set consisting of 5250 instances reveals that the procedure is capable of solving relatively large problem instances to optimality. A truncated version of the procedure is also validated against several local search procedures developed by De Reyck *et al.* [107], including a full-fledged tabu search procedure. The results show the truncated branch-and-bound procedure to outperform the set of local search methods, making it a viable alternative for solving relatively large instances of the DTRTP.

6.2. The multi-mode case

As mentioned above, the multi-mode resource-constrained project scheduling problem (problem MRCPSP in Table 4) includes time/cost, time/resource and resource/resource trade-offs, multiple renewable (limited on a per period basis), nonrenewable (limited for the entire project) and doubly-constrained resources (limited on both a per period and a total project basis) and a variety of objective functions. In the basic problem setting, activities have to be scheduled in one of their possible execution modes subject to renewable and nonrenewable resource constraints in order to minimize the project duration. Doubly-constrained resources can easily be taken into account by enlarging the sets of renewable and nonrenewable resources.

Sprecher *et al.* [108] have extended the DH-procedure to the multi-mode case under the minimum makespan objective. They borrowed the notion of tight schedules from Speranza and Vercellis [109] and introduce the notion of mode-minimal schedules. A schedule is tight if there does not exist an activity the finish time of which can be reduced without violating the constraints or changing the finish time or mode of any of the remaining activities in progress. They show that if there is an optimal schedule for a given instance, then there is an optimal schedule which is both tight and mode-minimal. Sprecher *et al.* [108] use a branching scheme which fixes the mode of eligible activities before putting them in progress. Resource conflicts are then resolved through the logic of the DH-procedure; i.e., by delaying minimal delaying alternatives. The algorithm has been coded in Borland C for an IBM-compatible 386DX personal computer with 40 MHz clockpulse, and has been tested on 536 instances generated using ProGen. Each instance consists of 10 activities, three possible execution modes for each activity with a duration varying between 1 and 10 periods, two renewable and two nonrenewable resources. The problems are solved in an average CPU-time of 0.53 s. As such it outperforms previously developed procedures by Sprecher [110] and Speranza and Vercellis [109]. Moreover, it has been shown by Hartmann and Sprecher [111] that the procedure of Speranza and Vercellis, by excluding from the search non-tight partial schedules, may miss the optimal solution.

Sprecher and Drexl [55] have subsequently developed a branch-and-bound procedure which relies on an enumeration scheme based on the precedence tree concept introduced by Patterson *et al.* [112,113] and already used by Sprecher [110]. In the precedence tree, an activity is considered to become eligible (and

to become a descendant of a parent node in the search tree) if all its predecessors are scheduled but not necessarily finished. The basic scheme is enhanced by different static and dynamic search tree reduction schemes, preprocessing and bounding rules. The procedure has been coded in GNU C and runs under OS/2 on a personal computer (80486DX processor, 66MHz clockpulse, 16 Mb memory). More than ten thousand problem instances have been generated using ProGen to evaluate the algorithm's performance. The number of activities in the test instances ranges from 10 to 20, from 1 up to 5 execution modes, 1 up to 5 renewable and 1 up to 3 nonrenewable resources. For the basic problem subset used to evaluate ProGen [53]; i.e. 536 ten-activity problems, 3 modes, 2 renewable and 2 nonrenewable resources, the authors report an average CPU-time of 0.14 s (standard deviation of 0.21 s, with a maximum of 2.31 s). Over the complete set of instances, CPU-times seem to increase exponentially with the number of jobs and modes and seem to decrease with an increasing complexity (measured by the above mentioned CNC). The number of renewable resources seems to influence the CPU-times linearly. The number of nonrenewable resources has a strong (positive correlation) impact on CPU-times. The higher the resource factor RF, and the lower the resource strength RS, the higher the CPU-time required. Encouraging results are reported using a truncated version of the algorithm. The Sprecher and Drexl [55] procedure clearly constitutes a benchmark for the MRCPSP.

Recently, the MRCPSP logic was extended through the incorporation of partially (non)renewable resources, in which resources are introduced which are nonrenewable for a certain time period rather than for the entire project duration, i.e. they can be renewed in certain time periods [114–116]. These partially renewable resources can be viewed as a generalization of both renewable and nonrenewable resources. In addition, recent research efforts have been directed towards the incorporation of mode identity constraints, a special case of the MRCPSP in which activities are grouped into sets for which one single execution mode has to be chosen [117,118].

7. CONCLUSIONS

Over the past decade, and especially over the past five years, considerable progress has been made in designing optimal solution procedures for the resource-constrained project scheduling problem (RCPSP). While at the time of the first extensive performance evaluation of optimal enumeration procedures [48], only one procedure [33] was capable of solving all the 110 Patterson test problems on a mainframe, we now witness the situation that all problems can be solved optimally by the DH2 procedure in an average CPU-time of 0.002 s on a Pentium Pro processor with 200 MHz clock pulse. These remarkable results indicate that the Patterson 110 problem set, an assembled set of test problems which do not satisfy pre-set values of problem parameters and which for many years has served as the de facto standard test set, can no longer uniquely serve as the benchmark test set for the RCPSP.

New optimal (and suboptimal) procedures should be validated on a wider set of test instances, generated to satisfy precept values of relevant problem parameters. ProGen, the problem generator developed by Kolisch *et al.* [53], has been used to generate a set of 480 RCPSP test instances which currently serves as the benchmark test set. The DH2 procedure has recently optimally solved all problems in this set in an average CPU-time of 0.37 s (with a maximum of 50.97 s) on a 200 MHz Pentium Pro personal computer.

Clearly, the state of the art is such that properly designed depth-first branch-and-bound procedures offer the best potential for solving the RCPSP. The solution logic which relies on a branching strategy based on resolving resource conflicts by delaying minimal subsets of activities in combination with an effective cutset dominance rule and easy to compute and effective lower bounds, lies at the very heart of the efficient DH-procedure [40] and its extensions DH1 [41] and DH2. The logic has a wide field of application as witnessed by the possible extensions into a number of important derived resource-constrained scheduling problems (see Table 5).

Computational experience gained on a wide variety of test instances confirms the rich potential of truncated branch-and-bound procedures. Often, optimal solutions can be found within short amounts of computation time. Often also, the quality of the solutions obtained by truncated branch-and-bound procedures outperforms the solution quality of many heuristics. Truncated exact procedures are promising tools for solving real problems (of sufficiently large size) within an acceptable computational burden and with acceptable solution quality.

Most computational experience has been gained on the 110 Patterson problem set and the test problems generated using ProGen (a project scheduling problem library has been made available on an ftp-site by Kolisch and Sprecher [56]). Results confirm that the 110 test problems no longer serve as the de facto

Table 5. Extensions of DH solution concepts

	RCPSP	PRCPSP	GRCPSP	RCPSP-GPR	DTRTP	MRCPSP	RCPSP-DC	RCPSPDC-GPR
Semi-active timetabling	yes	yes	yes	no	yes	yes	no	no
Min delaying alternatives(max scheduling alternatives)	yes	yes	yes	yes	yes	yes	yes	yes
Min delaying modes	no	no	no	yes	no	no	yes	yes
LB0	yes	yes	yes	yes	yes	yes	no	no
LBS	yes	yes	no	no	yes	yes	no	no
LBS^{ext}	yes	yes	no	no	yes	yes	no	no
LB3	yes	yes	yes	yes	yes	yes	no	no
Theorem 1	yes	yes	yes[a]	no	yes	yes	no	no
Theorem 2	yes	yes	yes[a]	no	yes	yes	no	no
Theorem 3	yes	yes	yes	no	yes	yes	no	no
Theorem 4	yes	yes	yes	no	yes	yes	no	no
Theorem 7 (subset dominance)	yes	yes	yes	yes	yes	yes	yes	yes

[a]Not if resource availabilities are allowed to vary over time.

standard. Gaining computational experience on a set of test problems which satisfy pre-set values of relevant problem parameters and which span the full range of complexity is crucial. This warrants further research on the issue of establishing additional relevant factors which explain the real complexity of a problem instance. Efforts to incorporate such parameters in the existing problem generators, or removing others with no explanatory power, should make it possible to establish workable problem test sets as a base for full factorial experiments which can be used for validating optimal and suboptimal procedures for solving the RCPSP and its important, realistic problem derivatives.

Acknowledgements—We would like to thank two anonymous referees for their comments and helpful suggestions for improving the structure and readability of this paper.

REFERENCES

1. Muth, J. F. and Thompson, G. L., *Industrial Scheduling*. Prentice Hall, Englewood Cliffs, N.J., 1963.
2. Conway, R. W., Maxwell, W. L. and Miller L. W., *Theory of Scheduling*. Addison-Wesley Publishing Company, 1967.
3. Ashour, S., *Sequencing Theory*. Springer-Verlag, Berlin, 1972.
4. Baker, K. R., *Introduction to Sequencing and Scheduling*. John Wiley, New York, 1974.
5. Rinnooy Kan, A. H. G., *Machine Scheduling Problems—Classification, Complexity and Computations*. M. Nijhoff, The Netherlands, 1976.
6. French, S., *Sequencing and Scheduling—An Introduction to the Mathematics of the Job Shop*. John Wiley, New York, 1982.
7. Bellmann, R., Esogbue, A. O. and Nabeshima, I., *Mathematical Aspects of Scheduling and Applications*. Pergamon Press, Oxford, U.K., 1982.
8. Herroelen, W., *Operationele Produktieplanning*. Acco, Leuven, 1991.
9. Blazewicz, J., Ecker, K., Schmidt, G. and Weglarz, J., *Scheduling in Computer and Manufacturing Systems*. Springer-Verlag, Berlin, 1993.
10. Morton, Th. E. and Pentico, D. W., *Heuristic Scheduling Systems—With Applications to Production Systems and Project Management*. Wiley Interscience, New York, 1993.
11. Tanaev, V. S., Gordon, V. S. and Shafransky, Y. M., *Scheduling Theory: Single-Stage Systems*. Kluwer Academic Publishers, Dordrecht, 1994.
12. Tanaev, V. S., Sotskov, Y. N. and Strusevich, V. A., *Scheduling Theory: Multi-Stage Systems*. Kluwer Academic Publishers, Dordrecht, 1994.
13. Brucker, P., *Scheduling Algorithms*. Springer-Verlag, Berlin, 1995.
14. Pinedo, M., *Scheduling—Theory, Algorithms and Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1995.
15. Gargeya, V. B. and Deane, R. H., Scheduling research in multiple resource constrained job shops: A review and critique. *International Journal of Production Research*, 1996, **34**, 2077–2097.
16. Blazewicz, J., Lenstra, J. K. and Rinnooy Kan, A. H. G., Scheduling projects to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 1983, **5**, 11–24.
17. Bowman, E. H., The schedule-sequencing problem. *Operations Research*, 1959, **7**, 621–624.
18. Brand, J. D., Meyer, W. L. and Shaffer, L. R., The resource scheduling problem in construction, Civil Engineering Studies, Report 5, Department of Civil Engineering, University of Illinois, Urbana, 1964.
19. Wiest, J. D., Some properties of schedules for large projects with limited resources. *Operations Research*, 1964, **12**, 395–418.
20. Moodie, C. L. and Mandeville, D. E., Project resource balancing by assembly line balancing techniques. *Journal of Industrial Engineering*, 1966, **17**, 377–383.
21. Elmaghraby, S. E., The sequencing of *n* jobs on *m* parallel processors. Unpublished paper, North Carolina State University at Raleigh, Raleigh, U.S.A., 1967.
22. Pritsker, A. B., Watters, L. J. and Wolfe, P. M., Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 1969, **16**, 93–108.
23. Patterson, J. H. and Huber, W. D., A horizon-varying zero-one approach to project scheduling. *Management Science*, 1974, **20**, 990–998.
24. Patterson, J. H. and Roth, G., Scheduling a project under multiple resource constraints: A zero-one programming approach. *AIIE Transactions*, 1976, **8**, 449–456.
25. Deckro, R. F., Winkofsky, E. P., Hebert, J. E. and Gagnon, R., A decomposition approach to multi-project scheduling. *European Journal of Operational Research*, 1991, **51**, 110–118.

26. Icmeli, O. and Rom, W. O., Solving the resource-constrained project scheduling problem with optimization subroutine library. *Computers and Operations Research*, 1996, **23**, 801–817.
27. Carruthers, J. A. and Battersby, A., Advances in critical path methods. *Operational Research Quarterly*, 1966, **17**, 359–380.
28. Petroviç, R., Optimisation of resource allocation in project planning. *Operations Research*, 1968, **16**, 559–586.
29. Johnson, T. J. R., An algorithm for the resource constrained project scheduling problem. Ph.D. Dissertation, MIT, 1967
30. Balas, E., Project Scheduling with Resource Constraints. In *Applications of Mathematical Programming*, ed E. M. L. Beale. The English University Press, London, 1971, pp. 187–200.
31. Schrage, L., Solving resource-constrained network problems by implicit enumeration—Nonpreemptive case. *Operations Research*, 1970, **10**, 263–278.
32. Davis, E. W. and Heidorn, G. E., An algorithm for optimal project scheduling under multiple resource constraints. *Management Science*, 1971, **27**, B803–B816.
33. Stinson, J. P., Davis, E. W. and Khumawala, B. M., Multiple resource-constrained scheduling using branch-and-bound. *AIIE Transactions*, 1978, **10**, 252–259.
34. Talbot, B. and Patterson, J. H., An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. *Management Science*, 1978, **24**, 1163–1174.
35. Radermacher, F. J., Scheduling of project networks. *Annals of Operations Research*, 1985, **4**, 227–252.
36. Christofides, N., Alvarez-Valdes, R. and Tamarit, J. M., Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 1987, **29**, 262–273.
37. Bartusch, M., Möhring, R. H. and Radermacher, F. J., Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 1988, **16**, 201–240.
38. Bell, C. A. and Park, K., Solving resource-constrained project scheduling problems by A* search. *Naval Research Logistics*, 1990, **37**, 61–84.
39. Carlier, J. and Latapie, B., Une méthode arborescente pour les problèmes cumulatifs. *R.A.I.R.O.*, 1991, **25**, 311–340.
40. Demeulemeester, E. and Herroelen, W., A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 1992, **38**, 1803–1818.
41. Demeulemeester, E. and Herroelen, W., New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 1997, **43**, 1485--1492.
42. Carlier, J. and Neron, E., A new branch and bound method for solving the resource constrained project scheduling problem. *Proceedings of the Fifth International Workshop on Project Management and Scheduling*, Poznan, April 11–13, 1996, pp. 61–65.
43. Mingozzi, A., Maniezzo, V., Ricciardelli, S. and Bianco, L., An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. *Management Science*, 1998, to appear.
44. Brucker, P., Schoo, A. and Thiele, O., A branch-and-bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 1998, to appear.
45. Davis, E. W., Resource allocation in project network models—A survey. *Journal of Industrial Engineering*, 1966, **17**, 177–188.
46. Davis, E. W., Project scheduling under resource constraints: Historical review and categorization of procedures. *AIIE Transactions*, 1973, **5**, 297–313.
47. Herroelen, W., Resource-constrained project scheduling—The state of the art. *Operational Research Quarterly*, 1972, **23**, 261–275.
48. Patterson, J. H., A comparison of exact procedures for solving the multiple constrained resource project scheduling problem. *Management Science*, 1984, **30**, 854–867.
49. Icmeli, O., Erengüç, S. S. and Zappe, J. C., Project scheduling problems: A survey. *International Journal of Production and Operations Management*, 1993, **13**, 80–91.
50. Elmaghraby, S. E., Activity nets: A guided tour through some recent developments. *European Journal of Operational Research*, 1995, **82**, 383–408.
51. Herroelen, W. and Demeulemeester, E., Recent advances in branch-and-bound procedures for resource-constrained project scheduling problems. In *Scheduling Theory and Its Applications*, eds Ph. Chrétienne, E. G. Coffmann Jr., J. K. Lenstra and Z. Liu. John Wiley and Sons, 1995, pp. 259–276.
52. Özdamar, L. and Ulusoy, G., A Survey on the resource-constrained project scheduling problem. *IIE Transactions*, 1995, **27**, 574–586.
53. Kolisch, R., Sprecher, A. and Drexl, A., Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 1995, **41**, 1693–1703.
54. Demeulemeester, E., Herroelen, W., Simpson, W. P., Baroum, S., Patterson, J. H. and Yang, K.-K., On a paper by Christofides *et al.* for solving the multiple-resource constrained single project scheduling problem. *European Journal of Operational Research*, 1994, **76**, 218–228.
55. Sprecher, A. and Drexl, A., Solving multi-mode resource-constrained project scheduling problems by a simple, general and powerful sequencing algorithm. *European Journal of Operational Research*, 1998, to appear.
56. Kolisch, R. and Sprecher, A., PSPLIB—A project scheduling problem library. *European Journal of Operational Research*, 1997, **96**, 205–216.
57. De Reyck, B. and Herroelen, W., On the use of the complexity index as a measure of complexity in activity networks. *European Journal of Operational Research*, 1996, **91**, 347–366.
58. Pascoe, T. L., Allocation of resources—CPM. *Revue Française de Recherche Opérationelle*, 1966, **38**, 31–38.
59. Davis, E. W., Project network summary measures and constrained resource scheduling. *IIE Transactions*, 1975, **7**, 132–142.
60. Talbot, F. B., Resource-constrained project scheduling with time-resource trade-offs: The nonpreemptive case. *Management Science*, 1982, **28**, 1197–1210.
61. Kurtulus, I. S. and Narula, S. C., Multi-project scheduling: Analysis of project performance. *IIE Transactions*, 1985, **17**, 58–66.
62. Alvarez-Valdes, R. and Tamarit, J. M. Heuristic algorithms for resource-constrained project scheduling: A review and empirical analysis. In *Advances in Project Scheduling*, ed R. Slowinski and J. Weglarz. Elsevier, Amsterdam, 1989.
63. Elmaghraby, S. E. and Herroelen, W., On the measurement of complexity in activity networks. *European Journal of Operational Research*, 1980, **5**, 223–234.
64. Bein, W. W., Kamburowski, J. and Stallmann, M. F. M., Optimal reduction of two-terminal directed acyclic graphs. *SIAM Journal on Computing*, 1992, **21**, 1112–1129.

65. De Reyck, B., On the use of the restrictiveness as a measure of complexity for resource-constrained project scheduling, Research Report 9535, Department of Applied Economics, K.U. Leuven., 1995.
66. De Reyck, B. and Herroelen, W., Assembly line balancing by resource-constrained project scheduling techniques—A critical appraisal. *Foundations of Computing and Decision Sciences*, 1997, **99**, 143--167.
67. Mastor, A. A., An experimental and comparative evaluation of production line balancing techniques. *Management Science*, 1970, **16**, 728–746.
68. Kao, E. P. C. and Queyranne, M., On dynamic programming methods for assembly line balancing. *Operations Research*, 1992, **30**, 375–390.
69. Dar-El, E. M., MALB—A heuristic technique for balancing large single-model assembly lines. *AIIE Transactions*, 1973, **5**, 343–356.
70. Cooper, D. F., Heuristics for scheduling resource-constrained projects: An experimental comparison. *Management Science*, 1976, **22**, 1186–1194.
71. Patterson, J. H., Project scheduling: The effects of problem structure on heuristic performance. *Naval Research Logistics*, 1976, **23**, 95–123.
72. Demeulemeester, E., Optimal algorithms for various classes of multiple resource-constrained project scheduling problems. Ph.D. Thesis, Department of Applied Economic Sciences, Katholieke Universiteit Leuven, 1992.
73. Slowinski, R., Two approaches to problems of resource allocation among project activities—A comparative study. *Journal of the Operational Research Society*, 1980, **31**, 711–723.
74. Weglarz, J., Project scheduling with continuously-divisible doubly-constrained resources. *Management Science*, 1981, **27**, 1040–1053.
75. Kaplan, L., Resource-constrained project scheduling with preemption of jobs. Ph.D. Dissertation, University of Michigan, 1988.
76. Kaplan, L., Resource-constrained project scheduling with setup times, Unpublished paper, Department of Management, University of Tenessee, Knoxville, 1991.
77. Demeulemeester, E. and Herroelen, W., An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. *European Journal of Operational Research*, 1996, **90**, 334–348.
78. Simpson III, W. P. and Patterson, J. H., A multiple-tree search procedure for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 1996, **89**, 525–542.
79. Kerbosch, J. A. G. M. and Schell, H. J., Network planning by the extended METRA potential method, Report KS-1.1, University of Technology Eindhoven, Department of Industrial Engineering, 1975.
80. Zahn, J., Heuristics for scheduling resource-constrained projects in MPM networks. *European Journal of Operational Research*, 1994, **76**, 192–205.
81. Moder, J. J., Phillips, C. R. and Davis, E. W., *Project Management with CPM, PERT and Precedence Diagramming*. Van Nostrand Reinhold, 1983.
82. Brinkmann, K. and Neumann, K., Heuristic procedures for resource-constrained project scheduling with minimal and maximal time lags: The minimum project duration and resource levelling problem, Technical Report WIOR-443, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe, 1994.
83. Neumann, K. and Zahn, J., Heuristics for the minimum project-duration problem with minimal and maximal time lags under fixed resource constraints. *Journal of Intelligent Manufacturing*, 1995, **6**, 145–154.
84. Neumann, K. and Schwindt, C., Projects with minimal and maximal time lags: Construction of activity-on-node networks and applications, Technical Report WIOR-447, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe, 1995.
85. Schwindt, C., 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, 1996.
86. Franck, B. and Neumann, K., Priority-rule methods for the resource-constrained project scheduling problem with minimal and maximal time lags—An empirical analysis. *Proceedings of the Fifth International Workshop on Project Management and Scheduling*, April 11–13, Poznan, 1996, pp. 88–91.
87. Schwindt, C. and Neumann, K., A new branch-and-bound-based heuristic for resource-constrained project scheduling with minimal and maximal time lags. *Proceedings of the Fifth International Workshop on Project Management and Scheduling*, April 11–13, Poznan, pp. 212–215, 1996.
88. Wikum, E. D., Donna, C. L. and Nemhauser, G. L., One-machine generalized precedence constrained scheduling problems. *Operations Research Letters*, 1994, **16**, 87–99.
89. Elmaghraby, S. E. and Kamburowski, J., The analysis of activity networks under generalized precedence relations. *Management Science*, 1992, **38**, 1245–1263.
90. Demeulemeester, E. and Herroelen, W., A branch-and-bound procedure for the generalized resource-constrained project scheduling problem. *Operations Research*, 1997, **45**, 201–212.
91. Möhring, R. H., Private communication, 1996.
92. De Reyck, B. and Herroelen, W., A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence constraints. *European Journal of Operational Research*, 1997, to appear.
93. Herroelen, W., Van Dommelen, P. and Demeulemeester, E., Project network models with discounted cash flows—A guided tour through recent developments. *European Journal of Operational Research*, 1997, **100**, 97–121.
94. Russell, A. H., Cash flows in networks. *Management Science*, 1970, **16**, 357–373.
95. Grinold, R. C., The payment scheduling problem. *Naval Research Logistics Quarterly*, 1972, **19**, 123–136.
96. Elmaghraby, S. E. and Herroelen, W., The scheduling of activities to maximize the net present value of projects. *European Journal of Operational Research*, 1990, **49**, 35–49.
97. Herroelen, W. and Gallens, E., Computational experience with an optimal procedure for the scheduling of activities to maximize the net present value of projects. *European Journal of Operational Research*, 1993, **65**, 274–277.
98. Demeulemeester, E., Herroelen, W. and Van Dommelen, P., An optimal recursive search procedure for the deterministic unconstrained MAX-NPV scheduling problem, Research Report 9603, Department of Applied Economics, K.U. Leuven, 1996.
99. De Reyck, B. and Herroelen, W., An optimal procedure for the unconstrained MAX-NPV project scheduling problem with generalized precedence relations, Research Report 9642, Department of Applied Economics, K.U. Leuven, 1996.
100. Baroum, S. M., An exact solution procedure for maximizing the net present value of resource-constrained projects. Unpublished Ph.D. Dissertation, Indiana University, 1992.

101. Icmeli, O. and Erenguç, S., A branch-and-bound procedure for the resource-constrained project scheduling problem with discounted cash flows. *Management Science*, 1996, **42**, 1395–1408.
102. Yang, K. K., Talbot, F. B. and Patterson, J. H., Scheduling a project to maximize its net present value: An integer programming approach. *European Journal of Operational Research*, 1992, **64**, 188–198.
103. De Reyck, B. and Herroelen, W., An optimal procedure for the resource-constrained project scheduling problem with discounted cash flows and generalized precedence relations. *Computers and Operations Research*, 1998, **25**, 1–17.
104. Herroelen, W., Een probleem van "resource allocation": Het toewijzen van netwerkactiviteiten aan ingenieurs. Comm. Eng. Thesis, Department of Applied Economics, K.U. Leuven, 1968.
105. Elmaghraby, S. E., *Activity Networks: Project Planning and Control by Network Models*. John Wiley and Sons, New York, 1977.
106. Demeulemeester, E., De Reyck, B. and Herroelen, W., The discrete time/resource trade-off problem in project networks—A branch-and-bound approach, Research Report 9717, Department of Applied Economics, K.U. Leuven, 1997.
107. De Reyck, B., Demeulemeester, E. and Herroelen, W., Local search methods for the discrete time/resource trade-off problem in project networks, Research Report 9710, Department of Applied Economics, K.U. Leuven, 1997.
108. Sprecher, A., Hartmann, S. and Drexl, A., An exact algorithm for project scheduling with multiple modes. *OR Spektrum*, 1997, **19**, 195–203.
109. Speranza, M. G. and Vercellis, C., Hierarchical models for multi-project planning and scheduling. *European Journal of Operational Research*, 1993, **64**, 312–325.
110. Sprecher, A., Resource-constrained project scheduling: Exact methods for the multi-mode case. In *Lecture Notes in Economics and Mathematical Systems*, Vol. 409, Springer-Verlag, Berlin, 1994.
111. Hartmann, S. and Sprecher, A., A note on "Hierarchical models for multi-project planning and scheduling". *European Journal of Operational Research*, 1996, **94**, 377–383.
112. Patterson, J. H., Slowinski, R., Talbot, F. B. and Weglarz, J., An algorithm for a general class of precedence and resource constrained scheduling problems. In *Advances in Project Scheduling*, ed R. Slowinski and J. Weglarz. Elsevier, Amsterdam, 1989.
113. Patterson, J. H., Slowinski, R., Talbot, F. B. and Weglarz, J., Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. *European Journal of Operational Research*, 1990, **49**, 68–79.
114. Böttcher, J., Drexl, A. and Kolisch, R., A branch-and-bound procedure for project scheduling with partially renewable resource constraints. *Proceedings of the Fifth International Workshop on Project Management and Scheduling*, Poznan, April 11–13, 1996, pp. 48–51.
115. Drexl, A., Local search methods for project scheduling under partially renewable resource constraints. *INFORMS San Diego Meeting*, May 4–7, 1997.
116. Schirmer, A. and Drexl, A., Partially renewable resources—A generalization of resource-constrained project scheduling. *IFORS 14th Triennal Conference*, Vancouver, B.C., July 8–12, 1996.
117. Salewski, F. and Lieberam-Schmidt, S., Greedy look ahead methods for project scheduling under resource and mode identity constraints. *Proceedings of the Fifth International Workshop on Project Management and Scheduling*, Poznan, April 11–13, 1996, pp. 207–211.
118. Salewski, F., Tabu search algorithms for project scheduling under resource and mode identity constraints. *IFORS 14th Triennal Conference*, Vancouver, B.C., July 8–12, 1996.