DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9710

# LOCAL SEARCH METHODS FOR THE DISCRETE TIME/RESOURCE TRADE-OFF PROBLEM IN PROJECT NETWORKS

by

BERT DE REYCK

ERIK DEMEULEMEESTER

WILLY HERROELEN

Katholieke Universiteit Leuven

Naamsestraat 69, B-3000 Leuven

ONDERZOEKSRAPPORT NR 9710

# LOCAL SEARCH METHODS FOR THE DISCRETE TIME/RESOURCE TRADE-OFF PROBLEM IN PROJECT NETWORKS

by

BERT DE REYCK

ERIK DEMEULEMEESTER

WILLY HERROELEN

D/1997/2376/11

# LOCAL SEARCH METHODS FOR THE DISCRETE TIME / RESOURCE TRADE-OFF PROBLEM IN PROJECT NETWORKS

- March 1997 -

**Bert DE REYCK**
**Erik DEMEULEMEESTER**
**Willy HERROELEN**

Operations Management Group
Department of Applied Economics
Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven, Belgium
Phone: 32-16-32 69 66, 32-16-32 69 72 or 32-16-32 69 70
Fax: 32-16-32 67 32
E-mail: Bert.DeReyck, Erik Demeulemeester or Willy.Herroelen@econ.kuleuven.ac.be
WWW-page: http://econ.kuleuven.ac.be/tew/academic/om

This research was supported by the National Science Foundation under Research Contract FWO N° G.0131.96

# LOCAL SEARCH METHODS FOR THE DISCRETE TIME / RESOURCE TRADE-OFF PROBLEM IN PROJECT NETWORKS

**Bert DE REYCK, Erik DEMEULEMEESTER and Willy HERROELEN**
Katholieke Universiteit Leuven, Leuven, Belgium

## ABSTRACT

In this paper we consider the discrete time/resource trade-off problem in project networks. Given a project network consisting of nodes (activities) and arcs (technological precedence relations specifying that an activity can only start when all of its predecessors have been completed), in which the duration of the activities is a discrete, non-increasing function of the amount of a single renewable resource committed to it, the discrete time/resource trade-off problem minimizes the project makespan subject to precedence constraints and a single renewable resource constraint. For each activity a work content is specified such that all execution modes (duration - resource requirement pairs) for performing the activity are allowed as long as the product of the duration and the resource requirement is at least as large as the specified work content. We present a tabu search procedure which is based on subdividing the problem into a mode assignment phase and a resource-constrained project scheduling phase with fixed mode assignments. Extensive computational experience, including a comparison with other local search methods, is reported.

## KEYWORDS

Project scheduling, Time/resource trade-offs, Local search, Tabu search, Branch-and-bound

## *1. Introduction*

The *resource-constrained project scheduling problem (RCPSP)* involves the nonpreemptive scheduling of project activities subject to finish-start precedence constraints and renewable resource constraints in order to minimize the project duration. Numerous exact and suboptimal procedures have been developed (for recent reviews see Herroelen and Demeulemeester, 1995 and Özdamar and Ulusoy, 1995). In the RCPSP each activity has a *single* execution *mode*: both the activity duration and its requirements for a set of renewable resource types are assumed to be fixed. Herroelen (1972) and Elmaghraby (1977) were the first authors to deal with discrete time-resource trade-offs and, correspondingly, multiple ways for executing the project activities.

In practice it often occurs that only one renewable resource type is available (e.g. labor) in constant amount throughout the project. For each activity a work content (e.g. amount of man-days) is specified. A set of allowable execution modes can then be specified for each activity, 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.

In the *discrete time/resource trade-off problem (DTRTP)* discussed in this paper, the duration of an activity is 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 type, during each period it is in progress, such that $r_{im}d_{im}$ is at least equal to and as close as possible to $W_i$. A mode is called efficient if there is no other mode with equal or smaller duration and smaller resource requirement or with equal or smaller 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 type 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 execution modes, subject to both the finish-start precedence constraint 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:

$$\text{Minimize} \sum_{t=e_n}^{l_n} t\, x_{n1t} \qquad [1]$$

Subject to

$$\sum_{m=1}^{M_i} \sum_{t=e_i}^{l_i} x_{imt} = 1 \qquad 1 \le i \le n \qquad [2]$$

$$\sum_{m=1}^{M_i} \sum_{t=e_i}^{l_i} \left(t + d_{im}\right) x_{imt} \le \sum_{m=1}^{M_j} \sum_{t=e_j}^{l_j} t\, x_{jmt} \qquad (i, j) \in E \qquad [3]$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{im} \sum_{s=\max\{t-d_{im},\, e_i\}}^{\min\{t-1,\, l_i\}} x_{ims} \le a \qquad 1 \le t \le T \qquad [4]$$

$$x_{imt} \in \{0,1\} \qquad 1 \le i \le n \quad 1 \le m \le M_i \quad 0 \le t \le T \qquad [5]$$

with $e_i$ ($l_i$) the earliest (latest) start time of activity $i$ based on the modes with the smallest duration, $T$ an upper bound on the project duration and $E$ the set of precedence relations. The objective function [1] corresponds to minimizing the project makespan. Constraints [2] ensure that each activity is assigned exactly one mode and exactly one start time. Constraints [3] denote the precedence constraints. Constraints [4] secure that the per-period availability of the renewable resource is met. Constraints [5] force the decision variables to assume binary values.

In this paper we present several local search methods for solving the DTRTP. The remainder of the paper is organized as follows. A review of the literature is given in Section 2. Section 3 describes the basic methodology used by the various local search methods. The solution logic of a new tabu search method is presented in Section 4. Computational experience is presented in Section 5, while section 6 is reserved for overall conclusions and suggestions for future research.

## *2. Review of the literature*

To the best of our knowledge, the literature on the DTRTP as defined in this paper is virtually void. Research efforts have been concentrated on two related problems. The *discrete time/cost trade-off problem (DTCTP)* studies time/cost trade-offs for a single *nonrenewable* resource type (De et al., 1995). In the DTCTP the duration of each activity is a discrete, nonincreasing function of the amount of a single nonrenewable resource committed to it. The DTCTP is studied under three different objectives: the minimization of the project duration under fixed resource availability, the minimization of the total resource consumption to achieve a target project completion time, and the construction of the efficient time/resource profile over the feasible project durations. The problem is known to be strongly NP-hard (see De et al., 1992). Optimal procedures and computational experience have been presented by Demeulemeester et al. (1996).

The DTRTP is also a subproblem of the *multi-mode resource-constrained project scheduling problem (MRCPSP),* which includes time/resource and resource/resource trade-offs, multiple renewable, nonrenewable and doubly-constrained resource types and a variety of objective functions. As a generalization of the RCPSP, the MRCPSP is NP-hard (Kolisch, 1995). *Optimal procedures* have been presented by Talbot (1982), Patterson et al. (1989,1990), Speranza and Vercellis (1993), Sprecher (1994), Sprecher et al. (1994), Sprecher and Drexl (1996a,b) and Ahn and Erengüç (1995). All of them use implicit enumeration with branch-and-bound.

Talbot (1982) presents a two-phase solution approach based on his enumeration scheme for the RCPSP. In the first phase activities, modes and renewable resources are sorted in order to speed up the enumeration procedure applied in phase two. In phase two a subset of the feasible schedules is exhaustively searched for the schedule with the smallest makespan. Patterson et al. (1989) refine Talbot's procedure by introducing a precedence tree which allows for the systematic enumeration of mode assignments and start times. Sprecher (1994) subsequently restructured and improved the procedure by adding dominance and bounding rules. He performed a computational experiment on a set of 536 multi-mode test problems (with 10 activities, three modes per activity, two renewable and two nonrenewable resource types) which were generated using the problem generator ProGen developed by Kolisch et al. (1995). The results indicate his algorithm to speed up the algorithm of Patterson et al. (1989,1990) by a factor of approximately one hundred. Nudtasomboon and Randhawa (1997) offer a zero-one integer programming model and slight modifications of the Talbot algorithm to cope with preemption and renewable and nonrenewable resources under various single and multiple time related, cost related and resource levelling objectives. limited computational results are obtained on nine data sets and a small warehouse construction rpoject.

Sprecher et al. (1994) have extended the optimal branch-and-bound procedure of Demeulemeester and Herroelen (1992) to the MRCPSP by fixing the mode of the eligible activities by selecting a mode alternative before putting them in progress and by using the concept of minimal delaying alternatives to resolve resource conflicts. Computational results on the same 536 problems used by Sprecher (1994) indicate that the new algorithm is approximately four times faster than Sprecher's. Sprecher and Drexl (1996a) have revised and restructured the Sprecher (1994) procedure through the incorporation of two preprocessing bounding rules and seven dynamic bounding rules. They report encouraging computational results on an extensive set of problems with up to 20 non-dummy activities, up to 5 execution modes and up to 5 renewable and 3 nonrenewable resource types.

Speranza and Vercellis (1993) proposed a depth-first branch-and-bound procedure which enumerates the set of tight schedules and which uses a precedence-based lower bound. A schedule is called *tight* if it does not contain an activity the finish time of which can be reduced without violating the constraints or changing the completion time or mode of the remaining activities. If there exists an optimal schedule for the MRCPSP, then there exists an optimal tight schedule.

Hence, an algorithm that enumerates all tight schedules will find an optimal solution. It has been shown by Hartmann and Sprecher (1993), however, that the algorithm developed by Speranza and Vercellis (1993) - by excluding from the search space non-tight partial schedules - may miss the optimal solution.

Ahn and Erengüç (1995) study the MRCPSP with the added assumption that within each mode, an activity duration may be crashed. The objective then is to determine the resource requirements, the amount of crashing implemented and the start time of each project activity so that the project cost is minimized. Computational results on 16 data sets generated using ProGen indicate the algorithm to outperform an adaptation of Sprecher's code.

*Heuristic solution procedures* for the MRCPSP have been developed by Talbot (1982), Drexl and Grünewald (1993), Slowinski et al. (1994), Boctor (1993, 1996, 1997), Özdamar and Ulusoy (1994), Kolisch (1995), Yang and Patterson (1995), Ahn and Erengüç (1996) and Sung and Lim (1997). Talbot (1982) recommends the use of a truncated version of his exact enumeration procedure. Drexl and Grünewald (1993) present a regret-based biased random sampling approach based on a joint use of a serial scheduling scheme and the shortest processing time rule. Kolisch (1995) compared his multi-mode heuristic (MMH) to the heuristics of Talbot (1982), Drexl and Grünewald (1993), Boctor (1993) and the truncated method of Sprecher (1994) and reached the conclusion that MMH outperforms every other heuristic except the truncated branch-and-bound procedure. Boctor (1996) presents a new heuristic which schedules the activity-mode combination that has the best value of a chosen evaluation criterion and which outperforms the heuristics presented in Boctor (1993) on 240 randomly generated test problems. Özdamar and Ulusoy (1994) present a constraint-based heuristic with an exponential time complexity.

Simulated annealing has been tried by Slowinski et al. (1994), Yang and Patterson (1995) and Boctor (1997). Boctor (1997) concludes that the simulated annealing algorithm outperforms the heuristics presented in Boctor (1993, 1996) on the 240 problem set. Slowinski et al. (1994) discuss a decision support system which uses three kinds of heuristics (parallel priority rules, simulated annealing and a truncated branch-and-bound) and report computational results on an hypothetical agricultural project. Yang and Patterson (1995) conclude that simulated annealing outperforms the backtracking algorithm of Patterson et al. (1989, 1990), in that it obtains the smallest mean project duration with less computational effort. Sung and Lim (1997) have developed a branch-and-bound procedure using two lower bounds which is incorporated within a two-phase heuristic method.

Despite the fact that excellent results have been reported using tabu search on (generalized) job shop scheduling problems (Vaessens, 1995; Vaessens et al., 1996), efforts to develop tabu search procedures for the RCPSP are rather sparse (Icmeli and Erengüç, 1994; Pinson et al., 1994; Lee and Kim, 1996) and have not yet been reported for the DTRTP. In the next section we describe the global solution logic of several local search heuristics for the DTRTP. A detailed description of a new tabu search procedure is given in Section 4.

## *3. Local search methods*

### *3.1. Basic methodology*

The local search methodology presented in this paper divides the DTRTP into two distinct phases: a mode assignment phase and a resource-constrained project scheduling phase with fixed mode assignments. The *mode assignment phase* assigns to each activity $i$ a specific execution mode $m_i$ (i.e. a specific duration and corresponding resource requirement). A mode assignment is an $n$-tuple $\mu = (m_1, m_2, \ldots, m_n)$, which yields a resource-constrained project scheduling problem, which is subsequently solved in the *resource-constrained project scheduling phase*.

### *3.2. Truncated complete enumeration*

Enumerating all possible mode assignments and solving each corresponding RCPSP instance to optimality leads to the optimal solution of the DTRTP. However, such an approach proves intractable because of the enormous amount of possible mode assignments ($O(M^n)$, where $M$ denotes the maximum number of modes that can be assigned to each activity). Computational experience with a truncated complete enumeration procedure in which each RCPSP instance is solved by a truncated branch-and-bound procedure will be given in section 5.2.1.

### *3.3. Improvement procedures*

The local search methods we develop start with an initial mode assignment $\mu = (m_1, m_2, \ldots, m_n)$ and compute an upper bound on the project makespan using a fast heuristic for the RCPSP. An improvement procedure is then initiated which evaluates a number of new mode assignments in the neighbourhood of $\mu$ (all mode assignments $\mu_k$ in which exactly one activity is assigned another mode) and selects one of them for further exploration. This process continues until some termination criterion is satisfied.

The evaluation of each move could be accomplished by optimally solving the corresponding RCPSP. However, when the number of activities grows large, no guarantee can be given that the RCPSP can be solved in a reasonable amount of time. Therefore, we use a truncated version of the RCPSP procedure of Demeulemeester and Herroelen (1997a) as a fast heuristic for solving the RCPSP. The procedure, which is an enhanced version of the original code presented in Demeulemeester and Herroelen (1992), is truncated after a very small amount of time has elapsed (namely when 100 backtracking steps have been performed, which requires, on the average, less than 0.01 seconds). Another possibility would be to truncate the procedure after a first feasible solution has been obtained. Upon finding the best mode assignment, it may be beneficial to run a near-optimal RCPSP heuristic, or, if possible, an optimal procedure to further improve on the obtained heuristic solution. We again use the RCPSP procedure of Demeulemeester and Herroelen (1997a), which, if truncated after 1 second of CPU-time, provides high quality, near-optimal solutions.

### 3.3.1. Steepest descent

Given an initial mode assignment μ, a *steepest descent* method (also referred to as *best-fit* or *best improvement* method) evaluates all mode assignments in the neighbourhood of μ and selects the one with the smallest project makespan. Then again the neighbourhood is determined and the best possible mode assignment selected. The steepest descent procedure terminates when no improving mode assignment can be found. Computational results are presented in section 5.2.6.

### 3.3.2. Fastest descent

A *fastest descent* algorithm (also referred to as a *first-fit* or *first improvement* procedure) differs from a steepest descent procedure in that the first improving mode assignment encountered is chosen. This will result in a faster descent at the expense of perhaps missing better mode assignments and steeper paths of descent at each iteration. Computational results are reported in section 5.2.4.

### 3.3.3. Iterated descent

Both steepest descent and fastest descent algorithms can be extended with a *random restart* procedure which randomly generates initial mode assignments upon which the procedure is restarted. Since these types of local search methods are known to be highly sensitive to the initial solution (mode assignment), incorporating random restarts will undoubtedly produce superior results. Results with an iterated fastest descent procedure are reported in section 5.2.5, while section 5.2.7 reports on results obtained with an iterated steepest descent procedure.

### 3.3.4. Tabu search

Improvement procedures such as steepest or fastest descent only accept alterations which result in an improvement of the incumbent solution. As a consequence, a major drawback is their tendency to being trapped in a local optimum. To overcome this disadvantage we will develop a tabu search (*TS*) procedure which behaves like a steepest descent procedure but which also allows for non-improving moves and a temporary deterioration of the objective function if no improving moves can be found. Then it will select the least deteriorating move (steepest descent / mildest ascent). Consequently, additional mechanisms are needed in order to prevent cycling between a number of solutions. The main principles of tabu search and the proposed procedure for the DTRTP will be presented in section 4. Computational results are reported in section 5.2.8.

### 3.3.5. Randomized search

Another way of crossing boundaries of local optimality is provided by biased random sampling or by simulated annealing. For the DTRTP they behave as follows. Given an initial mode assignment μ, both methods randomly generate a new mode assignment (possibly restricted to the neighbourhood of μ) which is enforced or discarded based on the associated value of the objective function. Improving moves are immediately enforced. Non-improving moves are accepted with a probability that decreases with the deterioration with respect to the incumbent solution. In the case of simulated annealing, the probability of accepting a deteriorating move also decreases as a function of a control parameter referred to as the temperature *T*, which guides the procedure by establishing the trade-off between a high probability of being trapped in a local optimum and a high probability of leaving a region of interest or, more specifically, the region of the global optimum. *T* is set relatively high in the initial phase of the procedure and, as the procedure proceeds, *T* is lowered in a systematic manner (often using a geometric cooling schedule of the form $T^{new} = \alpha\, T^{old}$ with $\alpha < 1$) such that the procedure becomes less erratic and further explores smaller regions of the solution space to a greater extent.

Tabu search and the randomized procedures share the tendency to overcome local optimality by also accepting non-improving moves. The main difference between the two, however, is that tabu search actively employs so-called flexible memory (in the short, medium and long term) to guide the search process into promising regions and to avoid cycling whereas randomized procedures are memoryless in which promising regions are probed and cycling is avoided by introducing randomization.

We have chosen not to implement randomized procedures for the DTRTP. This decision was motivated by the excellent results that have been reported using tabu search on various types of scheduling problems such as the job shop scheduling problem (Vaessens et al., 1996), and by various research results which indicate tabu search to outperform randomized procedures such as simulated annealing, both in solution quality and required computational effort (Pinson et al., 1994; Dell'Amico and Trubian, 1993). As a benchmark, however, Section 5.2.2 reports on the computational results obtained with a *random procedure* which randomly generates a number of mode assignments and solves the corresponding RCPSP using a truncated version of the procedure of Demeulemeester and Herroelen (1997a).

## *4. A tabu search procedure*

### *4.1. Neighbourhood*

We define the neighbourhood of a specific mode assignment μ as consisting of all mode assignments $\mu_k$ in which exactly one activity is assigned another mode. The maximum number of possible moves is equal to $\sum_{i=1}^{n}(M_i - 1)$. It is clear that using this neighbourhood structure, the *connectivity property* holds, i.e. that there exists, for each solution, a finite sequence of moves leading to the global optimal solution. Another possibility to define the neighbourhood or, equivalently, the set of available moves given an initial mode assignment, would be to restrict the change of mode $m_i$ of an activity $i$ to an 'adjacent' mode $m_i$-1 or $m_i$+1 (if possible), thereby decreasing, respectively increasing, the duration of activity $i$ by a small amount and increasing, respectively decreasing, the resource requirement of activity $i$ correspondingly. The maximum number of moves to be examined would then be restricted to 2($n$-2). Experimentation has revealed that local search procedures based on such a (more restricted) neighbourhood, although they are able to perform iterations much more quickly because less moves have to be evaluated, do not perform as well as do procedures based on the more extended neighbourhood. The main reason for this is that the procedures lose their aggressiveness and their ability to perform highly influential moves (moves that have a high impact on the structure of the solution, cf. infra) and to realize large improvements.

## *4.2. Short term recency-based memory*

### 4.2.1. Tabu principle

In order to avoid cycling tabu search employs short term memory, which excludes from consideration a specific number of moves which may lead to cycling. For the DTRTP several possibilities may be explored to prevent cycling: (a) classify tabu those moves that *reverse* one of the recently made moves, (b) allow reversals but prohibit *repetitions* of earlier made moves, such that revisiting an earlier solution is allowed but another search path has to be chosen for leaving the revisited solution, and (c) prohibit moves that lead to a mode assignment which was already encountered in the recent past.

Experimentation has revealed that approach (b), although effective in cycle avoidance, is outperformed by approach (a). Approach (c), although perfectly able to prevent revisits of earlier solutions, has two main disadvantages. First, comparing a set of mode assignments is of time complexity $O(nL)$, where $L$ denotes the length of the tabu list, compared to the constant time required by approach (a). Second, preventing earlier encountered solutions from being revisited is not synonymous to preventing cycling between a number of solutions. Sometimes it is advantageous that earlier visited solutions are revisited, in the hope that another search path out of that solution may be taken. Constructs developed to diversify the search (see section 4.6.2) will ensure that no indefinite cycling will occur, even if previously visited solutions are revisited.

An approach which partially eliminates the computational disadvantage of approach (c) is to use some kind of hashing function to either speed-up finding the relevant information in the tabu list using pointer structures (computational complexity $O(nL')$ where $L'$ is, in general, much smaller than $L$) or to compress the information in the tabu list itself. If we transform each mode assignment into a single number $h$ using some kind of hashing function, the computational complexity is reduced to $O(L)$. However, the cost of using such a hashing function is that due to the loss of information, some moves may be classified tabu although they do not lead to a revisit of an earlier encountered solution. Experiments revealed that this approach for solving the DTRTP is outperformed by approach (a). Since the number of moves that have to be evaluated at each iteration can be relatively large, it is important that the evaluation can be done very quickly. Hence, we opt for approach (a).

### 4.2.2. Using move attributes

Although we refer to moves being classified as tabu, we have to distinguish between moves and *attributes* of a move. A move can consist of several attributes (so-called from- and to-attributes) such that either the from-attributes or the to-attributes (or both) can be classified tabu. It is more effective to focus on from-attributes and to-attributes (and declaring attributes of moves tabu), than focusing on order pairs of attributes (complete moves).

If a move contains an attribute that is tabu, we declare it tabu. A move is defined as a change from $m_i = x$ to $m_i = y$ $(x \neq y)$, where $m_i$ denotes the mode assigned to activity $i$. Therefore, a reversal of such a move can be defined as a change from $m_i = y$ to $m_i = x$ which consequently can be classified tabu. However, prohibiting such reversals will not prevent cycling because a move from $m_i = y$ to $m_i = z$ $(z \neq x)$ followed by a move from $m_i = z$ to $m_i = x$ will not be prevented. Therefore, it is advisable to view the move from $m_i = x$ to $m_i = y$ $(x \neq y)$ as being comprised of the *from-attribute* $m_i = x$ and the *to-attribute* $m_i = y$ $(x \neq y)$. Then, cycling is excluded by prohibiting the to-attribute of a future move to be equal to the from-attribute of a previous move. Thus, after a move from $m_i = x$ to $m_i = y$ $(x \neq y)$ we prevent a move from $m_i = z$ to $m_i = x$ for arbitrary values of $z$ $(z \neq x)$, instead of restricting $z$ to being equal to $y$.

Actually, a change from $m_i = x$ to $m_i = y$ $(x \neq y)$ is comprised of two different attribute changes: changing $m_i = x$ to $m_i \neq x$, and changing $m_i \neq y$ to $m_i = y$. Consequently, it is possible to (a) prohibit a move from $m_i \neq x$ to $m_i = x$, or (b) to prohibit a move from $m_i = y$ to $m_i \neq y$. We prefer option (a) because it is less restrictive. Indeed, option (b) is more restrictive in that it corresponds to fixing the mode assigned to activity $i$ ($m_i = y$) for a number of iterations. The fact that a less restrictive way of classifying moves tabu is more effective than a more restrictive way has already been observed by Glover (1990b), who states that "when tabu restrictions are based on a single type move attribute, it is generally preferable to select an attribute whose tabu status less rigidly restricts the choice of available moves".

## *4.3. Tabu list management*

The length of the tabu list defines the time span during which moves retain their tabu status. A long tabu list decreases the probability of revisiting a previously examined mode assignment. However, it may also forbid a number of moves which would not have led to cycling at all, eventually causing the procedure to get stuck in a local optimum of poor quality.

Both *static* and *dynamic* tabu list management procedures have been described in the literature. A static tabu list has a specific constant length, such as 7 (a 'magical' number which has proven to be very effective in many TS applications) or $\sqrt{n}$ (advisable in situations when the tabu list length should vary with the problem dimension). Dynamic aspects include randomly varying the tabu list length in a specific interval, decreasing the tabu list length systematically, introducing moving gaps in the tabu list (parts of the tabu list are inactivated periodically) or making the tabu status of a move dependent on the state of the solution process (as in the cancellation sequence method or the reverse elimination method; see Glover, 1990a; Dammeyer and Voss, 1993). We use a tabu list which varies randomly in the interval $\left[\sqrt{n}, 3\sqrt{n}\right]$ and which initially contains $2\sqrt{n}$ moves. Each time a specific number of iterations is performed without improving the best known solution, the tabu list length is either decreased or increased by one unit (if possible), or remains the same (each with the same probability).

Note that, although we speak in terms of tabu *lists*, we are actually enforcing tabu restrictions based on time thresholds, i.e. when the current iteration is higher than the iteration in which the move was classified tabu plus a specific - possibly variable - threshold value.

## *4.4. Aspiration criteria*

Aspiration criteria are introduced to determine which tabu restrictions should be overridden, thereby removing the tabu status of certain moves. The purpose is to identify and again make available those moves that can lead to an overall improvement of the objective function or that can never lead to cycling and can therefore be released of their tabu status. We have chosen not to completely revoke the tabu status of such improving moves, but to transform them into a so-called *pending* tabu status (Glover and Laguna, 1993), which means that the move is eligible for selection if no other non-tabu *improving* move exists. Several aspiration criteria have been suggested in the literature (Glover and Laguna, 1993).

### 4.4.1. Aspiration by default

A *conflict* occurs when at some point in the search process no admissible moves are available for selection, i.e. when all possible moves are classified tabu. If such a conflict occurs the procedure will terminate unless some action is undertaken to revoke the tabu status of some moves. We remove the oldest entries from the tabu list until the list is empty or an admissible move exists. In some cases where the length of the tabu list is too long (such that every possible move is classified tabu), aspiration by default will automatically decrease the tabu list length.

### 4.4.2. Aspiration by objective

If a move that is classified tabu would lead to the best solution obtained so far, the tabu status is overridden and the corresponding move is selected. Obviously, when a new upper bound on the project makespan is found, we are certain that we are not revisiting an earlier examined solution. This aspiration criterion is known as *global aspiration by objective*. *Regional* (or *local*) *aspiration by objective* extends this reasoning to the best solution obtained so far in specific regions of the solution space. If a move that is classified tabu would lead to the best possible solution obtained so far with a specific mode assigned to a specific activity, we override the tabu status of that move. Therefore, we store for each activity-mode combination the best possible solution that has been obtained so far in which that activity has been given that specific mode.

### 4.4.3. Aspiration by influence

Moves can be classified according to their *influence*, i.e. their impact (induced degree of change) on the structure of the incumbent solution. For example, a move that changes the mode $(d_{im},r_{im})$ of an activity $i$ with work content $W_i$=40 from (4,10) to (5,8) will probably have a smaller impact on the corresponding schedule than a move to mode (10,4). Accordingly, we define the influence of a move as the absolute value of the difference between the current duration of an activity and its duration in the new mode assignment. Although such influential moves are often not very attractive because they lead to a substantial increase in the project makespan, they should be favoured from time to time in order to overcome local optimality and explore diverse regions of the solution space. If many moves of small influence have already been made and none of them is able to improve the best known solution so far, it is advisable to select a highly influential move after which a series of low-influence moves may again lead to a new local (hopefully global) optimum. Therefore, we will favour highly influential moves when a series of low-influence moves did not lead to a better solution. After such a high-influence move is chosen, the low-influence moves may again be tolerated until they show negligible gain opportunities. We revoke the tabu status of moves of rather low influence, provided that between the time (iteration) the move has been classified tabu and the current time (iteration), a move of higher influence has been chosen. Theoretically, aspiration by influence may lead to cycling. If this happens, other strategies such as diversification (see section 4.6.2) will be required to prevent indefinite cycling.

We also favour influential moves by making them more attractive in the move selection process. Moves are selected based on the associated upper bound on the project makespan. Influential moves are given a bonus that further increases their attractiveness. In order to obtain the right balance between moves that improve the makespan of the incumbent solution and highly influential moves, we use a move influence weight of 1/10. This means that (a) when two moves with the same project makespan are under consideration, the move with highest influence is preferred, and that (b) a difference in influence of magnitude 10 has the same weight as one unit difference in project makespan. We will only take into account the influence of moves when there are no moves that lead to a reduction in the makespan of the incumbent solution.

#### 4.4.4. Aspiration by search direction

In some cases, a revisit of an earlier examined solution is not necessarily bad, because another search path leading out of that solution might be chosen (thereby preventing cycling). Aspiration by search direction provides a mechanism for preventing that, upon a revisit, the same path out of that solution will be chosen. Therefore, we store for each move whether it was improving or not. If the current (tabu) move is an improving move and if the most recent move out of the new (tabu) mode assignment was also an improving move, the tabu status of the current move is revoked, thereby making it possible that the earlier examined solution is revisited. In this way, we revisit the local optimum examined before but we now choose another (non-improving) path leading out of that local optimum.

#### 4.4.5. Aspiration by strong admissibility

A move is labelled *strongly admissible* if it is eligible to be selected and does not rely on any aspiration criterion to qualify for admissibility, or if it would lead to the best solution obtained so far (Glover, 1990b). If such a strongly admissible move was made prior to the most recent iteration during which a non-improving move has been made, we revoke the tabu status of every improving move. In doing so, we make sure that the search proceeds to a local optimum, even if reaching a local optimum would require moves that would normally be classified tabu.

### *4.5. Termination criteria*

The procedure is terminated when (a) 10,000 iterations are performed, or (b) 1,000 iterations are performed without improving the best known solution (this number is deemed to be sufficient for the procedure to have either reached the global optimum or to have converged to and being stuck in a local optimum), or (c) the time limit of 100 seconds is exceeded, or (d) a solution is encountered with a makespan equal to a known lower bound. The lower bound $lb$ used for these calculations is the maximum of a critical path-based lower bound $lb_0$ and a resource-based lower bound $lb_r$. The critical path-based lower bound $lb_0$ is obtained by calculating the critical path in the activity network where each activity is assigned its shortest feasible mode, taking into account the resource availability $a$. The resource-based lower bound $lb_r$ is computed as $lb_r = \left\lceil \left( \sum_{i=1}^{n} W_i \right) / a \right\rceil$, where $\lceil x \rceil$ denotes the smallest integer equal to or larger than $x$. If for a specific activity $i$, $W_i' = \min_{m=1}^{M_i} \{d_{im} r_{im}\}$ exceeds $W_i$, we can use $W_i'$ for computing $lb_r$ rather than $W_i$ itself ( $W_i' - W_i$ is called the redundant work content).

### *4.6. Medium and long term frequency-based memory*

#### 4.6.1. Frequency information

The core of all TS procedures is a steepest descent / mildest ascent procedure supplemented with recency-based memory in the form of a tabu list to prevent cycling. Although this basic scheme, supplemented with appropriate aspiration criteria, may already outperform pure descent procedures, another component is necessary that typically operates on a somewhat longer time horizon to ensure that the search process examines solutions throughout the entire solution space (*diversification*) and that promising regions of the solution space (good local optima) are examined more thoroughly (*intensification*). This component can be supplied by using *frequency-based memory*. Essentially, frequency-based memory stores information about the frequency that a specific solution characteristic (attribute) occurs over all generated solutions or about the frequency that a move with a specific attribute has occurred. For instance, we can store (a) the number of generated solutions in which a specific activity was executed in a specific mode, (b) the number of times a move occurred in which an *arbitrary* new mode was reassigned to a specific activity, or (c) the number of times a *specific* mode was assigned to a specific activity. Option (a) represents a *residence frequency*, because it reports on the frequency of specific generated solutions, whereas options (b) and (c) represent *transition frequencies*, since they report on the frequency of specific moves. Although in many cases, residence frequencies are more suited to act as a frequency-based memory, both types of frequency-based memories can be used in unison to achieve a better performance.

#### 4.6.2. Diversification

Although the tabu list prevents the procedure from cycling between a number of solutions, it cannot prevent the search process from being restricted to a small region of the entire solution space. Furthermore, it may be advantageous to examine large parts of the solution space rather than intensively searching in a restricted part. Therefore, we will use frequency-based memory to detect whether the search space has been confined to a small region of the entire solution space, and use that information to guide the procedure into new unexplored regions. Diversification can be accomplished in two ways, using transition or residence frequency information.

#### 4.6.2.1. Transition frequencies

A first possibility is to adapt the attractiveness of the moves under consideration by including a frequency-based component which makes moves containing frequently encountered attributes less attractive than moves which contain rarely encountered attributes. In the proposed TS procedure we use a transition frequency-based memory that stores $f_i$, the frequency that a new mode was assigned to activity $i$. Moves concerning activities with small $f_i$ are favoured against moves pertaining to activities with large $f_i$. This is accomplished by adding a penalty term to the move selection criterion.

The diversifying influence on the move selection process is restricted to those occasions when no admissible moves exist that lead to a reduction in the makespan of the incumbent solution. In that case we penalize non-improving moves by assigning a larger penalty to moves with greater frequency counts. The reason for applying this restricted form of diversification is to preserve the aggressiveness (greediness) of the search, which is an essential characteristic of well-designed TS algorithms. A weight will have to be determined to trade-off between a smaller project makespan and a smaller frequency count. We use a weight of 1/3, which means that a unit difference in project makespan has the same weight in the move selection process as a frequency count of 3. Note that this penalty function ensures that, even when the tabu list cannot prevent a previously encountered solution from being visited again, no indefinite cycling will occur, because the penalty values of the activities participating in this cycle will grow until another activity will be chosen for mode reassignment and another search path that leads away from the current solution is selected and the cycle is broken.

#### 4.6.2.2. Residence frequencies

A second possibility of applying frequency-based memory for the purpose of diversification, is to divide the search process in different (possibly reoccurring) phases, which will diversify or intensify the search. After an initial data collection phase in which the required data for computing the frequencies is stored, a diversification phase can be started in which the procedure will be guided into unexplored regions of the search space. This can be accomplished through the use of residence frequencies which store information about the frequency that an activity was assigned a specific mode. If the frequencies indicate that for a specific activity only a small subset of all possible modes have been assigned to that activity, the search space is restricted by excluding those moves that assign one of these modes to that activity.

Consequently, a threshold value will have to be determined which defines which frequency counts should be considered as being significantly different from a uniform distribution. A straightforward threshold value would be the relative frequency of a specific activity-mode combination that have resulted when all modes for that activity were selected uniformly multiplied by a certain factor. Naturally, such threshold values should depend on the number of modes allowed for each activity. We have designed the following threshold value for diversification: 1.2 + M/5. This means that, for instance, when 4 modes are allowed, the relative frequency should be higher than 2 times its 'normal' value before the diversification procedure penalizes the activity-mode combination by classifying it as tabu for the time of the diversification phase.

The use of such residence frequencies is facilitated if we express them as a percentage by dividing the respective frequencies by the total number of iterations performed during the data collection phase. After such a diversification phase, all frequency-based memory is erased and a new data collection is initiated.

### 4.6.3. Intensification

Diversification phases should be alternated with intensification phases, in which the search is concentrated on a specific region of the solution space and promising regions are explored more intensively. This can also be accomplished through the use of frequency-based memory which stores the number of times a specific mode was assigned to each activity. When a high frequency count for a specific activity-mode combination is combined with a small associated project makespan, it may be advantageous to 'fix' the mode assignment of that activity to one mode or a small subset of all possible modes.

The intensification procedure examines all residence frequencies of the previously saved high quality local optima (defined as having an upper bound on the project makespan equal to the current best solution) and detects which activities have been assigned a specific mode or a small subset of all possible modes in each or a large number of these solutions. Then, the search space is restricted by limiting the possible modes for each activity to that small subset. This type of intensification strategy is often referred to as *reinforcement by restriction* (Glover and Laguna, 1993), because intensification is achieved by narrowing the realm of possible moves to those ones that promise high quality local optima instead of guiding the search process by using penalty and incentive functions. An advantage of reinforcement by restriction over penalty/incentive approaches is that the restriction of the search space leads to a significant speedup of each iteration, since the number of admissible moves will be substantially reduced. Note that reinforcement by restriction is not limited to intensification strategies only. The diversification strategy based on residence frequencies described in section 4.6.2.2. was also based on reinforcement by restriction, albeit to diversify the search (often referred to as *selective diversification*) rather than intensify it.

As was the case for diversification, a threshold value will have to be determined to decide when a relative frequency can be regarded as being significantly different from a uniform distribution. Since reinforcement by restriction ought to restrict the realm of available moves as much as possible, the threshold value will have to be higher than for diversification. We have chosen the following threshold value: 0.8 + M/2. This means that, for instance, when 4 modes are allowed, the relative frequency should be higher than 2.8 times its 'normal' value before the intensification phase limits the available moves to such moves exceeding the threshold value.

### 4.6.4. Combined diversification and intensification

Some constructs, although based on frequency-based memory, cannot be classified as performing the function of diversification and intensification, because they perform both functions simultaneously. One such construct is the concept of *solutions evaluated but not visited* (Glover and Laguna, 1993). Often, the choice between a number of moves is arbitrary because they have the same upper bound on the project makespan. We store the number of times an improving move was not selected although its associated project makespan was equal to the makespan of the selected mode assignment. If after a number of iterations (data collection phase), for a specific activity, a move (mode assignment) receives a high such frequency count although it has a low frequency count in the solutions that actually have been visited, the search space is restricted to those modes for that activity (reinforcement by restriction), thereby serving the goals of both intensification and diversification.

### 4.6.5. Phases

The total search time will be divided into several phases. We have chosen the following structure, which is truncated if one of the termination criteria applies:

PHASE 1. Proceed without intensification or diversification until 100 iterations have been made without improving the best known solution. Set $x$ equal to the number of the current iteration.

PHASE 2. Until iteration $x$+50: data collection for intensification

PHASE 3. Until iteration $x$+100: intensification

PHASE 4. Until iteration $x$+150: data collection for diversification

PHASE 5. Until iteration $x$+200: diversification

PHASE 6. Until iteration $x$+250: data collection for combined intensification and diversification

PHASE 7. Until iteration $x$+300: combined intensification and diversification

PHASE 8. Until iteration $x$+350: data collection for diversification

PHASE 9. Until iteration $x$+400: diversification

PHASE 10. Go to PHASE 2.

Each time the best known solution is improved upon, all frequency information is erased, the iteration counter $x$ is reset to the current iteration and the procedure continues with PHASE 1.

## *5. Computational experience*

The procedures have been programmed in Microsoft Visual C++ 2.0 under Windows NT for use on a Digital Venturis Pentium-60 personal computer with 16 Mb of internal memory. The codes itself require at most 120 kb and the data structures use at most 1.3 Mb of internal memory, which allows the procedures to be used on computer platforms with very little available memory.

### *5.1. Benchmark problem set*

Schwindt (1995) extended ProGen, the problem generator for the RCPSP developed by Kolisch et al. (1995), to ProGen/max which can randomly generate instances of various types of generalized resource-constrained project scheduling problems. We used ProGen/max to generate 150 networks using the control parameters given in Table 1. For each combination of control parameter values, 10 problem instances have been generated. The indication [*x*,*y*] means that the value is randomly generated in the interval [*x*,*y*], whereas *x*; *y*; *z* means that three settings for that parameter were used in a full factorial experiment. Every activity is then randomly assigned a work content between 10 and 100. Several versions of each problem instance are solved using a different restriction of the number of modes allowed (denoted *M*, varying from 1 to 6 and one in which the number of modes is unlimited) and a different setting for the resource availability *a* (equal to 10, 20, 30, 40 and 50). The parameters used in the full factorial experiment are given in Table 2. A total of 5,250 problem instances results.

**Table 1.** The parameter settings of the benchmark problem set

| Control parameter | Value |
|---|---|
| number of activities | 10; 15; 20; 25; 30 |
| activity work content | [10,100] |
| number of initial and terminal activities | [2,5] |
| maximal number of predecessors and successors | 3 |
| *OS*[1] | 0.25; 0.50; 0.75 |

**Table 2.** The parameter settings of the full factorial experiment

| Control parameter | Value |
|---|---|
| number of activities | 10; 15; 20; 25; 30 |
| *OS* | 0.25; 0.50; 0.75 |
| *a* | 10; 20; 30; 40; 50 |
| *M* | 1; 2; 3; 4; 5; 6; unlimited |

[1] Schwindt (1996) uses an estimator for the restrictiveness (Thesen, 1977) as a network complexity measure. However, De Reyck (1995) has shown that this measure is identical to the order strength (Mastor, 1970), the flexibility ratio (Dar-El, 1973) and the density (Kao and Queyranne, 1982). We will use *order strength* when referring to this measure.

When a restriction is imposed on the number of modes, it is enforced as follows. The procedure first generates the mode with duration $d_{im} = \max\left\{\left\lfloor\sqrt{W_i}\right\rfloor, \left\lceil\frac{W_i}{a}\right\rceil\right\}$ and resource requirement $r_{im} = \left\lceil\frac{W_i}{d_i}\right\rceil$, where $\lfloor x \rfloor$ denotes the highest integer number equal to or smaller than $x$ and $\lceil x \rceil$ denotes the smallest integer number equal to or higher than $x$. This mode is typically situated somewhere in the 'middle' of the realm of available modes. Then, the procedure generates a mode with a duration equal to $d_{im} - 1$ and a corresponding resource requirement. Consequently, the mode with duration $d_{im} + 1$ is generated. This mode generation process continues (alternatively decreasing resp. increasing the activity duration) until the desired number of modes is reached or no modes are left. Naturally, other criteria to restrict the number of modes can be used, such as eliminating modes that are not allowed due to technological or other constraints.

When the number of modes is unlimited, the actual number of modes per activity $i$ depends on the work content $W_i$ and the resource availability $a$. Table 3 indicates for each value for the resource availability $a$, the global average number of modes for all corresponding problem instances and their theoretical minimum and maximum. For the complete problem set, the global average number of modes when there is no restriction on their number equals 11.82. When the number of modes for each activity is restricted to 1, 2, 3, 4, 5 or 6, the actual number of modes for each activity is exactly equal to that number, except when $a$=10 and $r_i$=11, which only allows for 5 feasible *efficient* modes $(d_{im}, r_{im})$, namely (2,6), (3,4), (4,3), (6,2) and (11,1).

**Table 3.** Average number of modes

| | $a$ = 10 | $a$ = 20 | $a$ = 30 | $a$ = 40 | $a$ = 50 |
|---|---|---|---|---|---|
| global average number of modes | 8.84 | 11.60 | 12.49 | 12.92 | 13.26 |
| theoretical minimum number of modes | 5 | 6 | 6 | 6 | 6 |
| theoretical maximum number of modes | 10 | 15 | 16 | 17 | 18 |

## *5.2. Computational results*

### 5.2.1. A truncated complete enumeration procedure

The complete enumeration procedure enumerates all possible activity-mode combinations and evaluates each resulting RCPSP instance using a fast heuristic procedure. A global time limit of 100 seconds is imposed. The best results (shown in Tables 4 through 10) are obtained by running the RCPSP procedure of Demeulemeester and Herroelen (1997a) for a very short time (until 100 backtracking steps have been performed, which requires, on the average, less than 0.01

seconds) and by enumerating the activity-mode combinations starting from the modes with the smallest duration. Finally, the RCPSP instance which corresponds to the best mode assignment encountered so far is further investigated using the procedure of Demeulemeester and Herroelen (1997a) with a time limit of 1 second. Table 4 denotes the average and maximal deviation with respect to the best known solution for each problem instance, which is obtained using all procedures presented in this paper (4,929 (±94%) of those solutions are known to be optimal). In Table 5 the number of times the best known solution is obtained is shown. In Table 6 the average deviation of the heuristic solutions with respect to the lower bound *lb* are given. Table 7 indicates the number of problems solved to optimality. Solutions are known to be optimal when they have a makespan equal to the lower bound *lb* or when all possible solutions have been enumerated. Table 8 reports the average number of RCPSP instances solved, whereas Table 9 indicates the average required CPU-time.

**Table 4.** Truncated complete enumeration: average deviation from best solution

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n$ = 10 | 0.00% | 0.00% | 0.00% | 0.05% | 0.35% | 1.44% | 3.35% |
| $n$ = 15 | 0.00% | 0.00% | 0.79% | 3.41% | 3.95% | 4.62% | 6.79% |
| $n$ = 20 | 0.00% | 0.31% | 1.28% | 4.01% | 4.51% | 4.92% | 6.82% |
| $n$ = 25 | 0.00% | 1.28% | 2.50% | 6.52% | 6.91% | 6.88% | 8.10% |
| $n$ = 30 | 0.00% | 1.75% | 2.69% | 6.34% | 7.02% | 7.05% | 8.47% |

**Table 5.** Truncated complete enumeration: number of times best solution is obtained

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n$ = 10 | 150 | 150 | 150 | 149 | 134 | 98 | 52 |
| $n$ = 15 | 150 | 150 | 129 | 99 | 89 | 63 | 5 |
| $n$ = 20 | 150 | 140 | 106 | 93 | 84 | 61 | 4 |
| $n$ = 25 | 150 | 117 | 92 | 77 | 77 | 47 | 0 |
| $n$ = 30 | 149 | 97 | 88 | 85 | 79 | 51 | 0 |

**Table 6.** Truncated complete enumeration: average deviation from *lb*

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n$ = 10 | 5.42% | 3.27% | 2.25% | 2.44% | 2.25% | 3.60% | 6.49% |
| $n$ = 15 | 3.96% | 2.55% | 2.67% | 5.42% | 5.53% | 6.29% | 10.48% |
| $n$ = 20 | 3.39% | 2.48% | 2.80% | 5.96% | 5.99% | 6.68% | 10.81% |
| $n$ = 25 | 4.04% | 3.79% | 4.41% | 8.39% | 8.28% | 8.37% | 12.00% |
| $n$ = 30 | 3.82% | 4.32% | 4.67% | 8.58% | 8.76% | 9.18% | 13.41% |

**Table 7.** Truncated complete enumeration: number of problems solved to optimality

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 83 | 90 | 80 | 84 | 62 | 20 |
| $n = 15$ | 150 | 90 | 92 | 80 | 80 | 58 | 0 |
| $n = 20$ | 150 | 94 | 93 | 81 | 80 | 59 | 0 |
| $n = 25$ | 150 | 84 | 83 | 75 | 76 | 46 | 0 |
| $n = 30$ | 149 | 84 | 83 | 76 | 77 | 50 | 0 |

**Table 9.** Truncated complete enumeration: average number of RCPSP instances solved

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 1 | 466 | 24,107 | 503,411 | 1,865,430 | 2,698,000 | 5,055,088 |
| $n = 15$ | 1 | 13,400 | 869,831 | 756,340 | 820,282 | 1,473,462 | 3,323,424 |
| $n = 20$ | 1 | 244,974 | 543,339 | 523,337 | 580,677 | 1,013,794 | 2,271,647 |
| $n = 25$ | 1 | 341,138 | 365,540 | 281,593 | 274,180 | 613,974 | 754,581 |
| $n = 30$ | 1 | 261,113 | 302,929 | 289,860 | 267,145 | 459,602 | 760,187 |

**Table 8.** Truncated complete enumeration: average CPU-time (seconds)

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0,00 | 0,06 | 1,42 | 20,57 | 47,02 | 60,91 | 88,98 |
| $n = 15$ | 0,00 | 2,66 | 40,01 | 46,67 | 46,68 | 61,36 | 100,01 |
| $n = 20$ | 0,00 | 32,24 | 38,24 | 46,24 | 46,70 | 60,93 | 100,02 |
| $n = 25$ | 0,02 | 44,16 | 45,02 | 50,28 | 49,83 | 70,21 | 100,05 |
| $n = 30$ | 0,02 | 44,06 | 44,73 | 49,40 | 48,74 | 66,80 | 100,09 |

**Table 10.** Truncated complete enumeration: summary

| | Global summary | Unlimited number of modes |
|---|---|---|
| Average deviation from the best known solution | 3.20% | 6.70% |
| Maximum deviation from the best known solution | 43.18% | 24.00% |
| Number of times the best known solution is obtained | 3,315 (±63%) | 61 (±8%) |
| Average deviation from *lb* | 5.79% | 10.64% |
| Maximum deviation from *lb* | 48.84% | 33.33% |
| Problems solved to optimality | 2709 (±52%) | 20 (±3%) |
| Average number of RCPSP instances solved | 787,224 | 2,432,985 |
| Average CPU-time (in seconds) | 44.40 | 97.83 |

### 5.2.2. A random procedure

For use as a benchmark we have also tested a fully randomized procedure, in which a number of random mode assignments are generated and the corresponding RCPSP instances solved for a very short time (100 backtracking steps). The procedure continues until 100 seconds have elapsed or until a solution equal to *lb* is encountered. Then, a final intensification phase is initiated in which the RCPSP based on the best mode assignment is solved with a time limit of 1 second. The results are given in Tables 11 through 17.

**Table 11.** Random procedure: average deviation from best solution

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00% | 0.00% | 0.00% | 0.62% | 0.40% | 0.90% | 7.43% |
| $n = 15$ | 0.00% | 0.01% | 0.32% | 2.04% | 2.37% | 4.95% | 19.95% |
| $n = 20$ | 0.00% | 0.49% | 1.17% | 5.07% | 5.34% | 8.28% | 31.49% |
| $n = 25$ | 0.00% | 0.62% | 2.21% | 5.87% | 8.14% | 11.62% | 32.69% |
| $n = 30$ | 0.00% | 1.31% | 3.26% | 7.54% | 10.07% | 14.64% | 45.36% |

**Table 12.** Random procedure: number of times best solution is obtained

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 150 | 150 | 119 | 128 | 111 | 13 |
| $n = 15$ | 150 | 149 | 118 | 53 | 51 | 7 | 0 |
| $n = 20$ | 150 | 95 | 70 | 4 | 6 | 3 | 0 |
| $n = 25$ | 150 | 85 | 39 | 4 | 1 | 0 | 0 |
| $n = 30$ | 149 | 47 | 7 | 0 | 0 | 0 | 0 |

**Table 13.** Random procedure: average deviation from *lb*

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 5.42% | 3.27% | 2.25% | 3.03% | 2.30% | 3.04% | 10.80% |
| $n = 15$ | 3.96% | 2.56% | 2.19% | 3.98% | 3.88% | 6.58% | 24.35% |
| $n = 20$ | 3.39% | 2.67% | 2.69% | 6.97% | 6.76% | 10.01% | 36.81% |
| $n = 25$ | 4.04% | 3.10% | 4.10% | 7.65% | 9.43% | 13.09% | 37.86% |
| $n = 30$ | 3.82% | 3.84% | 5.23% | 9.66% | 11.72% | 16.79% | 52.27% |

**Table 14.** Random procedure: number of problems solved to optimality

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 83 | 90 | 74 | 82 | 64 | 7 |
| $n = 15$ | 150 | 90 | 92 | 52 | 49 | 6 | 0 |
| $n = 20$ | 150 | 78 | 69 | 4 | 6 | 3 | 0 |
| $n = 25$ | 150 | 73 | 39 | 4 | 1 | 0 | 0 |
| $n = 30$ | 149 | 45 | 7 | 0 | 0 | 0 | 0 |

**Table 15.** Random procedure: average number of RCPSP instances solved

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 1 | 352,913 | 389,634 | 559,937 | 566,193 | 796,851 | 1,452,940 |
| $n = 15$ | 1 | 265,278 | 352,813 | 671,535 | 809,707 | 1,110,983 | 1,280,826 |
| $n = 20$ | 1 | 321,401 | 523,093 | 918,648 | 933,751 | 952,433 | 1,090,505 |
| $n = 25$ | 1 | 269,490 | 599,221 | 724,857 | 768,135 | 729,823 | 920,483 |
| $n = 30$ | 1 | 406,657 | 690,497 | 668,518 | 685,720 | 659,328 | 838,681 |

**Table 16.** Random procedure: average CPU-time (seconds)

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00 | 44.67 | 40.10 | 50.95 | 48.22 | 62.23 | 96.90 |
| $n = 15$ | 0.00 | 40.04 | 41.22 | 66.94 | 75.81 | 97.85 | 100.00 |
| $n = 20$ | 0.00 | 48.16 | 61.46 | 97.35 | 96.45 | 98.59 | 100.00 |
| $n = 25$ | 0.02 | 52.00 | 80.79 | 97.48 | 99.76 | 100.02 | 100.01 |
| $n = 30$ | 0.02 | 70.65 | 97.54 | 100.02 | 100.03 | 100.03 | 100.02 |

**Table 17.** Random procedure: summary

| | Global summary | Unlimited number of modes |
|---|---|---|
| Average deviation from the best known solution | 6.69% | 27.38% |
| Maximum deviation from the best known solution | 186.67% | 186.67% |
| Number of times the best known solution is obtained | 2,159 (±41%) | 13 (±2%) |
| Average deviation from *lb* | 9.41% | 32.42% |
| Maximum deviation from *lb* | 186.67% | 186.67% |
| Problems solved to optimality | 1767 (±34%) | 7 (±1%) |
| Average number of RCPSP instances solved | 608,882 | 1,116,687 |
| Average CPU-time (in seconds) | 67.58 | 99.39 |

### 5.2.3. A branch-and-bound procedure

In Demeulemeester et al. (1997) we present a branch-and-bound procedure for the DTRTP based on the concept of 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 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 activity-mode combination is evaluated by computing a critical path-based and a resource-based lower bound. The one with the smallest lower bound is selected for further branching.

Backtracking occurs when a schedule is completed or when a branch is to be fathomed by the lower bound calculations or 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, some of which are extensions of rules originally developed for the RCPSP. At a certain decision point certain maximal activity-mode combinations may be discarded, namely those in which an activity can be performed in a mode with shorter duration (and its completion time reduced) without causing a resource conflict at that decision point or affecting the start times or modes of the other activities, such that the activity then terminates the earliest among all activities in progress. These so-called non-tight activity-mode combinations can be eliminated since they will never lead to a superior solution than when branching from the partial schedule in which a faster mode is chosen for that activity. However, not all non-tight partial schedules (in which *at least one arbitrary* activity can be put in a faster mode without causing a resource conflict or affecting the completion times of the other activities) can be eliminated from further consideration, because this may lead to missing optimal solution, as was shown to be the case in the procedure of Speranza and Vercellis (1993) by Hartmann and Sprecher (1993).

The *single-mode left-shift rule* dominates partial schedules (nodes) for which an activity can be scheduled in the same mode with an earlier completion time. The *multi-mode left-shift rule* dominates partial schedules for which an activity can be scheduled in another mode with an earlier completion time. The *mode reduction rule* dominates partial schedules for which an activity can be scheduled in a shorter mode with the same completion time. The *multi-mode cutset dominance rule* dominates partial schedules which contain the same set of activities as previously generated (and saved) partial schedules but in which the activities in progress do not finish earlier and, if they were also in progress in the earlier encountered partial schedule, use more (or at least as many) resource units.

At each decision point $t$ the extended precedence-based lower bound $lb_0^{ext}$ is computed as follows: $lb_0^{ext} = \max\left\{\max_{i \notin S}\{x_i\}, \max_{i \in S}\{\min\{x_{i1}, s_{i2}\}\}\right\}$, where $S$ denotes the set of scheduled activities, $x_i$ is computed as the next decision point $t'$ plus the remaining critical path length of the unscheduled eligible activity $i$ in its shortest mode, $x_{i1}$ is computed as the start time of the scheduled activity $i$ plus the remaining critical path length based on the chosen duration of scheduled activity $i$ and $x_{i2}$ is calculated as the next decision point $t'$ plus the remaining critical path length of the scheduled activity $i$ based on minimal durations. The resource-based lower bound $lb_r$ is computed as the ratio of the sum of the activity work contents and the resource availability, rounded to the next higher integer and updated for resource-period availability losses: $lb_r = \left\lceil \left( \sum_{i=1}^{n} W_i \right) / a \right\rceil$, where $\lceil x \rceil$ denotes the smallest integer equal to or larger than $x$. These losses may be due to unused (lost) resource-period availabilities on one hand and the redundant work content resulting from the fact that the product of the activity duration and the corresponding resource requirement exceeds the specified work content $W_i$. The lower bound $lb$ used in the branch-and-bound procedure is then obtained as $\max\left\{lb_0^{ext}, lb_r\right\}$. The depth-first branching rule branches from the *tight* activity-mode combinations with smallest lower bound first. The smallest resource-period availability losses and the latest next decision point are used as tie-breaking rules.

In order to compare the performance of the branch-and-bound procedure with the local search methods, we have recompiled it for use on a Digital Venturis Pentium-60 personal computer with 16 Mb of internal memory. As for all procedures researched in this paper, a time limit of 100 seconds is imposed. However, sometimes the branch-and-bound procedure is unable to find a feasible solution within the given time limit. Then it is allowed to continue until a first feasible solution has been found. The fact that a first feasible solution is sometimes only encountered after considerable computational effort is due to the inherent complexity of the DTRTP and the nature of the search process. Although the procedure is of the depth-first type, it often examines thousands of maximal activity-mode combinations (nodes) at each level of the search tree. For one specific problem instance 167,524 nodes were examined before encountering a first feasible solution. The CPU-time required for evaluating those nodes equals 1,567 seconds, which is the maximum CPU-time required by the branch-and-bound procedure.

**Table 18.** Branch-and-bound: average deviation from best solution

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0,00% | 0,00% | 0,00% | 0,00% | 0,00% | 0,00% | 0,33% |
| $n = 15$ | 0,00% | 0,00% | 0,00% | 0,00% | 0,00% | 0,00% | 0,70% |
| $n = 20$ | 0,00% | 0,00% | 0,00% | 0,00% | 0,00% | 0,00% | 0,29% |
| $n = 25$ | 0,00% | 0,00% | 0,00% | 0,00% | 0,00% | 0,00% | 1,22% |
| $n = 30$ | 0,00% | 0,00% | 0,00% | 0,00% | 0,01% | 0,04% | 1,50% |

**Table 19.** Branch-and-bound: number of times best solution is obtained

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 150 | 150 | 150 | 150 | 150 | 145 |
| $n = 15$ | 150 | 150 | 150 | 150 | 150 | 150 | 139 |
| $n = 20$ | 150 | 150 | 150 | 150 | 150 | 150 | 140 |
| $n = 25$ | 150 | 150 | 150 | 150 | 150 | 150 | 123 |
| $n = 30$ | 150 | 150 | 150 | 150 | 149 | 147 | 98 |

**Table 20.** Branch-and-bound: average deviation from *lb*

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 5.42% | 3.27% | 2.25% | 2.39% | 1.89% | 2.13% | 3.38% |
| $n = 15$ | 3.96% | 2.55% | 1.85% | 1.88% | 1.45% | 1.55% | 4.18% |
| $n = 20$ | 3.39% | 2.16% | 1.48% | 1.80% | 1.34% | 1.59% | 4.02% |
| $n = 25$ | 4.04% | 2.44% | 1.83% | 1.66% | 1.20% | 1.33% | 4.88% |
| $n = 30$ | 3.82% | 2.47% | 1.88% | 1.97% | 1.52% | 1.94% | 6.10% |

**Table 21.** Branch-and-bound: number of problems solved to optimality

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 150 | 150 | 150 | 150 | 150 | 144 |
| $n = 15$ | 150 | 150 | 150 | 150 | 150 | 150 | 128 |
| $n = 20$ | 150 | 150 | 150 | 150 | 150 | 150 | 100 |
| $n = 25$ | 150 | 150 | 150 | 150 | 149 | 146 | 48 |
| $n = 30$ | 150 | 150 | 150 | 148 | 146 | 133 | 30 |

**Table 22.** Branch-and-bound: average CPU-time (seconds)

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00 | 0.00 | 0.01 | 0.01 | 0.02 | 0.10 | 11.49 |
| $n = 15$ | 0.00 | 0.01 | 0.01 | 0.02 | 0.04 | 0.08 | 28.61 |
| $n = 20$ | 0.01 | 0.04 | 0.05 | 0.09 | 0.21 | 1.38 | 50.37 |
| $n = 25$ | 0.06 | 0.13 | 0.72 | 0.77 | 2.75 | 6.64 | 102.01 |
| $n = 30$ | 0.06 | 0.17 | 0.77 | 2.36 | 5.18 | 15.51 | 111.21 |

**Table 23.** Branch-and-bound: summary

| | Global summary | Unlimited number of modes |
|---|---|---|
| Average deviation from the best known solution | 0.12% | 0.81% |
| Maximum deviation from the best known solution | 21.05% | 21.05% |
| Number of times the best known solution is obtained | 5,141 (±98%) | 645 (86%) |
| Average deviation from *lb* | 2.60% | 4.51% |
| Maximum deviation from *lb* | 42.86% | 27.78% |
| Problems solved to optimality | 4922 (±94%) | 450 (60%) |
| Average CPU-time (in seconds) | 9.74 | 60.74 |

### 5.2.4. Fastest descent

Tables 24 through 30 show the results for a fastest descent algorithm in which the initial mode assignment is determined by assigning to each activity its mode with smallest duration (which resulted in the best performance). Again, each RCPSP instance is solved for a very short time (100 backtracking steps). Allowing more time per RCPSP instance does not improve the performance of the fastest descent algorithm significantly. In the final intensification phase the RCPSP instance resulting from the best mode assignment is solved with a time limit of 1 second.

**Table 24.** Fastest descent: average deviation from best solution

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00% | 1.22% | 1.42% | 2.97% | 2.88% | 4.25% | 6.81% |
| $n = 15$ | 0.00% | 1.15% | 1.27% | 2.66% | 2.70% | 4.30% | 6.54% |
| $n = 20$ | 0.00% | 0.92% | 0.98% | 1.98% | 2.24% | 3.30% | 6.07% |
| $n = 25$ | 0.00% | 0.83% | 1.07% | 2.30% | 2.48% | 4.11% | 6.00% |
| $n = 30$ | 0.00% | 1.15% | 1.21% | 2.48% | 2.27% | 4.03% | 5.45% |

**Table 25.** Fastest descent: number of times best solution is obtained

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 117 | 108 | 86 | 82 | 68 | 15 |
| $n = 15$ | 150 | 112 | 103 | 90 | 82 | 61 | 6 |
| $n = 20$ | 150 | 109 | 96 | 87 | 81 | 68 | 6 |
| $n = 25$ | 150 | 97 | 89 | 77 | 74 | 51 | 0 |
| $n = 30$ | 149 | 98 | 90 | 83 | 77 | 57 | 1 |

**Table 26.** Fastest descent: average deviation from *lb*

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 5.42% | 4.58% | 3.73% | 5.48% | 4.85% | 6.51% | 10.02% |
| $n = 15$ | 3.96% | 3.77% | 3.17% | 4.66% | 4.25% | 5.97% | 10.19% |
| $n = 20$ | 3.39% | 3.13% | 2.50% | 3.86% | 3.64% | 5.00% | 10.01% |
| $n = 25$ | 4.04% | 3.33% | 2.94% | 4.04% | 3.74% | 5.54% | 9.81% |
| $n = 30$ | 3.82% | 3.69% | 3.13% | 4.56% | 4.07% | 6.05% | 10.23% |

**Table 27.** Fastest descent: number of problems solved to optimality

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 79 | 81 | 68 | 69 | 52 | 1 |
| $n = 15$ | 150 | 87 | 88 | 80 | 80 | 54 | 0 |
| $n = 20$ | 150 | 91 | 91 | 81 | 81 | 65 | 2 |
| $n = 25$ | 150 | 84 | 85 | 74 | 74 | 49 | 0 |
| $n = 30$ | 149 | 84 | 84 | 77 | 76 | 55 | 1 |

**Table 28.** Fastest descent: average number of RCPSP instances solved

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 1 | 13 | 28 | 44 | 61 | 81 | 158 |
| $n = 15$ | 1 | 21 | 42 | 66 | 88 | 123 | 276 |
| $n = 20$ | 1 | 27 | 57 | 89 | 124 | 169 | 406 |
| $n = 25$ | 1 | 36 | 75 | 118 | 165 | 225 | 521 |
| $n = 30$ | 1 | 44 | 94 | 139 | 205 | 285 | 658 |

**Table 29.** Fastest descent: average CPU-time (seconds)

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00 | 0.00 | 0.01 | 0.01 | 0.02 | 0.01 | 0.02 |
| $n = 15$ | 0.00 | 0.01 | 0.03 | 0.04 | 0.07 | 0.06 | 0.09 |
| $n = 20$ | 0.00 | 0.17 | 0.20 | 0.61 | 0.49 | 0.34 | 0.64 |
| $n = 25$ | 0.03 | 0.40 | 0.51 | 2.20 | 2.37 | 2.19 | 2.37 |
| $n = 30$ | 0.03 | 0.63 | 1.29 | 3.97 | 3.73 | 4.28 | 8.37 |

**Table 30.** Fastest descent: summary

| | Global summary | Unlimited number of modes |
|---|---|---|
| Average deviation from the best known solution | 2.49% | 6.17% |
| Maximum deviation from the best known solution | 27.78% | 16.67% |
| Number of times the best known solution is obtained | 2,920 (±56%) | 28 (±4%) |
| Average deviation from *lb* | 5.05% | 10.05% |
| Maximum deviation from *lb* | 50.00% | 28.57% |
| Problems solved to optimality | 2642 (±50%) | 4 (±1%) |
| Average number of RCPSP instances solved | 127 | 404 |
| Average CPU-time (in seconds) | 1.01 | 2.30 |

### 5.2.5. Fastest iterated descent

Tables 31 through 37 indicate the results for a fastest descent algorithm with random restarts. Unless restricted by the time limit of 100 seconds, 1,000 mode assignments are determined randomly, after which the fastest descent procedure is initiated. Each RCPSP instance is solved until 100 backtracking steps have been performed. A final intensification step with a time limit of 1 second is used for further examining the best mode assignment. As was to be expected, equipping a fastest descent procedure with random restarts significantly increases the quality of the obtained solutions, at the expense of increased computational requirements.

**Table 31.** Fastest iterated descent: average deviation from best solution

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00% | 0.00% | 0.09% | 0.41% | 0.33% | 0.30% | 1.07% |
| $n = 15$ | 0.00% | 0.02% | 0.21% | 0.61% | 0.69% | 0.59% | 1.65% |
| $n = 20$ | 0.00% | 0.09% | 0.40% | 0.80% | 0.93% | 1.03% | 2.21% |
| $n = 25$ | 0.00% | 0.22% | 0.68% | 1.41% | 1.55% | 1.58% | 3.19% |
| $n = 30$ | 0.00% | 0.37% | 0.77% | 1.44% | 1.70% | 1.90% | 3.22% |

**Table 32.** Fastest iterated descent: number of times best solution is obtained

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 150 | 141 | 125 | 126 | 136 | 109 |
| $n = 15$ | 150 | 147 | 128 | 116 | 111 | 109 | 67 |
| $n = 20$ | 150 | 138 | 119 | 104 | 106 | 87 | 43 |
| $n = 25$ | 150 | 119 | 101 | 84 | 83 | 69 | 16 |
| $n = 30$ | 149 | 105 | 101 | 87 | 89 | 70 | 11 |

**Table 33.** Fastest iterated descent: average deviation from *lb*

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 5.42% | 3.27% | 2.35% | 2.81% | 2.22% | 2.43% | 4.13% |
| $n = 15$ | 3.96% | 2.57% | 2.08% | 2.51% | 2.16% | 2.15% | 5.13% |
| $n = 20$ | 3.39% | 2.25% | 1.90% | 2.63% | 2.29% | 2.65% | 5.99% |
| $n = 25$ | 4.04% | 2.67% | 2.54% | 3.13% | 2.79% | 2.94% | 6.89% |
| $n = 30$ | 3.82% | 2.87% | 2.69% | 3.46% | 3.25% | 3.84% | 7.89% |

**Table 34.** Fastest iterated descent: number of problems solved to optimality

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 83 | 90 | 80 | 86 | 75 | 42 |
| $n = 15$ | 150 | 90 | 92 | 85 | 85 | 72 | 5 |
| $n = 20$ | 150 | 93 | 101 | 88 | 89 | 76 | 2 |
| $n = 25$ | 150 | 85 | 88 | 79 | 79 | 63 | 0 |
| $n = 30$ | 149 | 89 | 90 | 80 | 83 | 60 | 0 |

**Table 35.** Fastest iterated descent: average number of RCPSP instances solved

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 1 | 8,373 | 16,166 | 29,680 | 38,579 | 58,340 | 207,638 |
| $n = 15$ | 1 | 11,773 | 23,674 | 40,817 | 55,676 | 81,508 | 388,840 |
| $n = 20$ | 1 | 13,697 | 23,670 | 44,849 | 53,826 | 71,155 | 357,870 |
| $n = 25$ | 1 | 17,642 | 33,097 | 45,566 | 48,932 | 66,676 | 174,467 |
| $n = 30$ | 1 | 19,085 | 36,938 | 48,235 | 46,211 | 69,336 | 163,781 |

**Table 36.** Fastest iterated descent: average CPU-time (seconds)

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00 | 3.06 | 4.49 | 5.84 | 7.10 | 9.95 | 19.82 |
| $n = 15$ | 0.00 | 7.69 | 10.03 | 15.63 | 20.62 | 31.65 | 70.64 |
| $n = 20$ | 0.00 | 11.91 | 13.67 | 25.23 | 32.00 | 41.42 | 89.81 |
| $n = 25$ | 0.02 | 21.48 | 28.83 | 41.18 | 43.71 | 54.89 | 100.00 |
| $n = 30$ | 0.02 | 23.31 | 31.25 | 44.27 | 43.83 | 56.54 | 99.16 |

**Table 37.** Fastest iterated descent: summary

| | Global summary | Unlimited number of modes |
|---|---|---|
| Average deviation from the best known solution | 0.84% | 2.27% |
| Maximum deviation from the best known solution | 10.00% | 10.00% |
| Number of times the best known solution is obtained | 3,746 (±71%) | 246 (±33%) |
| Average deviation from *lb* | 3.35% | 6.01% |
| Maximum deviation from *lb* | 42.86% | 16.67% |
| Problems solved to optimality | 2879 (±55%) | 49 (±7%) |
| Average number of RCPSP instances solved | 65,603 | 258,519 |
| Average CPU-time (in seconds) | 28.83 | 75.89 |

### 5.2.6. Steepest descent

Tables 38 through 44 denote the results for a steepest descent algorithm in which the initial mode assignment is determined by assigning to each activity its mode with smallest duration. Each RCPSP instance is solved until 100 backtracking steps have been performed. Contrary to expectations, the overall quality of the solutions obtained with steepest descent do not significantly differ from the those obtained using a fastest descent approach. On the average, steepest descent is only slightly superior, both in the number of problems solved to optimality and in the deviations with respect to *lb* and the best known solution. Only for an unlimited amount of modes steepest descent is distinctly superior. We have also extended the steepest descent algorithm with several tie-breaking rules, which decide which move has to be selected when several moves have equal associated upper bounds on the project makespan (instead of choosing one arbitrarily). The tie-breakers we examined included selecting mode assignments in which the new activity-mode combination has the smallest redundant work content ($d_i r_i - W_i$) and selecting modes which have a large impact on the structure of the solution (measured by the *influence* of a move as defined in section 4.4.3). Such tie-breaking rules, however, do not seem to lead to a superior performance.

**Table 38.** Steepest descent: average deviation from best solution

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00% | 1.45% | 1.38% | 2.95% | 2.70% | 4.18% | 6.25% |
| $n = 15$ | 0.00% | 1.56% | 1.20% | 3.05% | 2.79% | 4.14% | 6.10% |
| $n = 20$ | 0.00% | 1.06% | 0.87% | 2.27% | 2.21% | 3.37% | 5.70% |
| $n = 25$ | 0.00% | 0.89% | 1.02% | 2.62% | 2.50% | 3.79% | 5.60% |
| $n = 30$ | 0.00% | 1.47% | 1.16% | 2.61% | 2.53% | 3.80% | 5.11% |

**Table 39.** Steepest descent: number of times best solution is obtained

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 117 | 112 | 86 | 83 | 72 | 17 |
| $n = 15$ | 150 | 111 | 106 | 92 | 82 | 63 | 6 |
| $n = 20$ | 150 | 111 | 100 | 87 | 82 | 73 | 6 |
| $n = 25$ | 150 | 103 | 91 | 78 | 77 | 57 | 0 |
| $n = 30$ | 149 | 100 | 94 | 86 | 78 | 61 | 3 |

**Table 40.** Steepest descent: average deviation from *lb*

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 5.42% | 4.84% | 3.69% | 5.46% | 4.66% | 6.43% | 9.46% |
| $n = 15$ | 3.96% | 4.21% | 3.09% | 5.06% | 4.34% | 5.81% | 9.73% |
| $n = 20$ | 3.39% | 3.28% | 2.38% | 4.16% | 3.61% | 5.08% | 9.62% |
| $n = 25$ | 4.04% | 3.39% | 2.89% | 4.37% | 3.76% | 5.21% | 9.40% |
| $n = 30$ | 3.82% | 4.03% | 3.09% | 4.70% | 4.13% | 5.82% | 9.88% |

**Table 41.** Steepest descent: number of problems solved to optimality

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 82 | 82 | 68 | 69 | 55 | 2 |
| $n = 15$ | 150 | 88 | 90 | 80 | 80 | 57 | 0 |
| $n = 20$ | 150 | 91 | 97 | 81 | 82 | 70 | 2 |
| $n = 25$ | 150 | 84 | 85 | 75 | 76 | 54 | 0 |
| $n = 30$ | 149 | 87 | 88 | 77 | 77 | 57 | 2 |

**Table 42.** Steepest descent: average number of RCPSP instances solved

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 1 | 14 | 31 | 59 | 79 | 98 | 182 |
| $n = 15$ | 1 | 23 | 51 | 98 | 127 | 169 | 374 |
| $n = 20$ | 1 | 32 | 72 | 149 | 195 | 253 | 598 |
| $n = 25$ | 1 | 48 | 98 | 232 | 293 | 396 | 896 |
| $n = 30$ | 1 | 57 | 133 | 295 | 383 | 505 | 1,193 |

**Table 43.** Steepest descent: average CPU-time (seconds)

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00 | 0.00 | 0.01 | 0.02 | 0.02 | 0.02 | 0.03 |
| $n = 15$ | 0.00 | 0.02 | 0.03 | 0.08 | 0.08 | 0.07 | 0.12 |
| $n = 20$ | 0.00 | 0.20 | 0.31 | 1.47 | 1.43 | 0.62 | 0.94 |
| $n = 25$ | 0.03 | 0.51 | 0.78 | 6.34 | 5.40 | 4.02 | 3.47 |
| $n = 30$ | 0.03 | 1.28 | 2.66 | 7.51 | 8.55 | 5.45 | 8.47 |

**Table 44.** Steepest descent: summary

| | Global summary | Unlimited number of modes |
|---|---|---|
| Average deviation from the best known solution | 2.47% | 5.75% |
| Maximum deviation from the best known solution | 27.78% | 19.05% |
| Number of times the best known solution is obtained | 2,983 (±57%) | 32 (±4%) |
| Average deviation from *lb* | 5.03% | 9.62% |
| Maximum deviation from *lb* | 50.00% | 28.57% |
| Problems solved to optimality | 2687 (±51%) | 6 (±1%) |
| Average number of RCPSP instances solved | 204 | 649 |
| Average CPU-time (in seconds) | 1.71 | 2.60 |

### 5.2.7. Steepest iterated descent

Tables 45 through 51 indicate the results for a steepest descent algorithm with random restarts. Unless restricted by the time limit of 100 seconds, 1,000 random restarts are performed. Each RCPSP instance is solved until 100 backtracking steps have been performed. A final intensification step with a time limit of 1 second is used for further examining the best mode assignment. Again, equipping a steepest descent procedure with random restarts significantly increases the quality of the obtained solutions. Surprisingly however, the slight advantage of steepest descent versus fastest descent disappears when random restarts are introduced. In fact, iterated fastest descent performs significantly better than iterated steepest descent. The main reason for this is the fact that a fastest descent approach requires much less time to perform each iteration, thereby making it possible to perform more iterations and random restarts. Notice, however, that the total amount of RCPSP instances evaluated is substantially higher using a steepest descent approach. The number of iterations, however, is much less because using a fastest descent approach an iteration requires solving much less RCPSP instances.

**Table 45.** Steepest iterated descent: average deviation from best solution

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00% | 0.00% | 0.05% | 0.29% | 0.30% | 0.15% | 0.84% |
| $n = 15$ | 0.00% | 0.01% | 0.23% | 0.62% | 0.80% | 0.58% | 1.70% |
| $n = 20$ | 0.00% | 0.08% | 0.43% | 0.80% | 1.03% | 1.14% | 2.49% |
| $n = 25$ | 0.00% | 0.20% | 0.85% | 1.79% | 1.93% | 2.09% | 3.83% |
| $n = 30$ | 0.00% | 0.37% | 0.97% | 1.76% | 1.98% | 2.36% | 4.23% |

**Table 46.** Steepest iterated descent: number of times best solution is obtained

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 150 | 145 | 131 | 129 | 143 | 116 |
| $n = 15$ | 150 | 148 | 126 | 112 | 106 | 108 | 64 |
| $n = 20$ | 150 | 139 | 115 | 105 | 101 | 85 | 35 |
| $n = 25$ | 150 | 121 | 95 | 77 | 76 | 60 | 8 |
| $n = 30$ | 149 | 106 | 98 | 83 | 83 | 66 | 4 |

**Table 47.** Steepest iterated descent: average deviation from *lb*

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 5.42% | 3.27% | 2.30% | 2.69% | 2.19% | 2.28% | 3.90% |
| $n = 15$ | 3.96% | 2.57% | 2.09% | 2.52% | 2.27% | 2.15% | 5.17% |
| $n = 20$ | 3.39% | 2.24% | 1.93% | 2.63% | 2.39% | 2.76% | 6.28% |
| $n = 25$ | 4.04% | 2.65% | 2.72% | 3.51% | 3.17% | 3.46% | 7.55% |
| $n = 30$ | 3.82% | 2.86% | 2.89% | 3.80% | 3.55% | 4.32% | 8.94% |

**Table 48.** Steepest iterated descent: number of problems solved to optimality

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 83 | 90 | 81 | 85 | 78 | 44 |
| $n = 15$ | 150 | 90 | 92 | 83 | 84 | 72 | 5 |
| $n = 20$ | 150 | 93 | 101 | 88 | 89 | 76 | 2 |
| $n = 25$ | 150 | 85 | 85 | 74 | 73 | 58 | 0 |
| $n = 30$ | 149 | 89 | 88 | 79 | 81 | 58 | 0 |

**Table 49.** Steepest iterated descent: average number of RCPSP instances solved

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 1 | 11,349 | 29,731 | 58,003 | 76,420 | 109,567 | 433,934 |
| $n = 15$ | 1 | 21,151 | 60,613 | 109,901 | 136,040 | 186,227 | 1,016,542 |
| $n = 20$ | 1 | 30,925 | 74,990 | 112,324 | 121,544 | 166,949 | 1,054,419 |
| $n = 25$ | 1 | 43,685 | 76,719 | 95,687 | 122,442 | 168,947 | 484,645 |
| $n = 30$ | 1 | 48,114 | 78,001 | 95,037 | 123,286 | 222,499 | 522,723 |

**Table 50.** Steepest iterated descent: average CPU-time (seconds)

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00 | 3.59 | 6.61 | 8.96 | 11.59 | 14.63 | 25.79 |
| $n = 15$ | 0.00 | 10.12 | 16.80 | 28.20 | 33.40 | 41.78 | 83.68 |
| $n = 20$ | 0.00 | 16.49 | 27.41 | 38.20 | 37.89 | 46.23 | 97.24 |
| $n = 25$ | 0.02 | 30.48 | 37.86 | 46.71 | 46.40 | 56.22 | 100.20 |
| $n = 30$ | 0.02 | 35.97 | 39.57 | 46.72 | 45.95 | 59.47 | 100.30 |

**Table 51.** Steepest iterated descent: summary

| | Global summary | Unlimited number of modes |
|---|---|---|
| Average deviation from the best known solution | 0.97% | 2.62% |
| Maximum deviation from the best known solution | 10.00% | 10.00% |
| Number of times the best known solution is obtained | 3,684 (±70%) | 227 (30%) |
| Average deviation from *lb* | 3.48% | 6.37% |
| Maximum deviation from *lb* | 42.86% | 16.67% |
| Problems solved to optimality | 2855 (±54%) | 51 (±7%) |
| Average number of RCPSP instances solved | 168,355 | 702,453 |
| Average CPU-time (in seconds) | 34.13 | 81.44 |

5.2.8. Tabu search

Tables 52 through 58 indicate the results obtained with the TS procedure. The initial mode assignment is determined by assigning to each activity the mode with duration $\max\left\{\left\lfloor\sqrt{W_i}\right\rfloor,\left\lceil\frac{W_i}{a}\right\rceil\right\}$, where $\lfloor x \rfloor$ denotes the highest integer number equal to or smaller than $x$ and $\lceil x \rceil$ denotes the smallest integer number equal to or higher than $x$. The reason for choosing this mode instead of the mode with the smallest duration is the fact that it is situated somewhere in the middle of all available modes, thereby opening up possibilities for both increasing and decreasing the duration of the activities.

In general, using the modes with the smallest associated duration will cause local search procedures to get stuck in local optima rather quickly. Often, the inevitable increase in the duration of an activity will cause an equivalent increase in the project duration. However, these local optima are often already of high quality. That is why for the descent algorithms described above, using these modes led to the best performance, and why, in other types of multiple mode resource-constrained project scheduling problems, the heuristics in which for each activity the shortest duration is selected led to a superior performance than when any other mode is chosen (see, for instance, Boctor, 1996). For the TS procedure, the initial solution does not have a significant impact on the quality of the obtained solutions. When the shortest modes are selected for the initial mode assignment, the overall average deviations from the best known solution and lower bound are 0.29% (vs. 0.27%) and 2.78% (vs. 2.76%), respectively. The number of problems solved to (verified) optimality equals 2932 (vs. 2933). Each RCPSP instance is solved until 100 backtracking steps have been performed. A final intensification step with a time limit of 1 second is used for further examining the best mode assignment.

**Table 52.** Tabu search: deviation from best solution

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0.00% | 0.00% | 0.00% | 0.03% | 0.01% | 0.09% | 0.19% |
| $n = 15$ | 0.00% | 0.00% | 0.00% | 0.05% | 0.05% | 0.13% | 0.73% |
| $n = 20$ | 0.00% | 0.02% | 0.01% | 0.07% | 0.20% | 0.34% | 1.21% |
| $n = 25$ | 0.00% | 0.05% | 0.06% | 0.22% | 0.40% | 0.73% | 1.70% |
| $n = 30$ | 0.00% | 0.08% | 0.09% | 0.18% | 0.35% | 0.78% | 1.83% |

**Table 53.** Tabu search: number of times best solution is obtained

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 150 | 150 | 149 | 149 | 145 | 141 |
| $n = 15$ | 150 | 150 | 150 | 147 | 146 | 137 | 108 |
| $n = 20$ | 150 | 148 | 149 | 144 | 133 | 117 | 73 |
| $n = 25$ | 150 | 145 | 145 | 133 | 107 | 87 | 46 |
| $n = 30$ | 150 | 141 | 139 | 127 | 107 | 83 | 36 |

**Table 54.** Tabu search: average deviation from *lb*

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 5.42% | 3.27% | 2.25% | 2.42% | 1.90% | 2.22% | 3.25% |
| $n = 15$ | 3.96% | 2.55% | 1.85% | 1.93% | 1.50% | 1.69% | 4.18% |
| $n = 20$ | 3.39% | 2.18% | 1.49% | 1.87% | 1.54% | 1.94% | 4.96% |
| $n = 25$ | 4.04% | 2.50% | 1.89% | 1.89% | 1.61% | 2.07% | 5.35% |
| $n = 30$ | 3.82% | 2.56% | 1.98% | 2.16% | 1.87% | 2.69% | 6.43% |

**Table 55.** Tabu search: number of problems solved to optimality

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 150 | 83 | 90 | 80 | 86 | 79 | 62 |
| $n = 15$ | 150 | 90 | 92 | 87 | 90 | 81 | 13 |
| $n = 20$ | 150 | 93 | 101 | 88 | 89 | 76 | 2 |
| $n = 25$ | 150 | 85 | 88 | 79 | 80 | 64 | 0 |
| $n = 30$ | 149 | 89 | 91 | 81 | 83 | 60 | 2 |

**Table 56.** Tabu search: average number of RCPSP instances solved

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 1 | 7,153 | 11,935 | 21,682 | 27,411 | 39,795 | 84,513 |
| $n = 15$ | 1 | 9,124 | 17,005 | 28,563 | 36,989 | 48,811 | 156,065 |
| $n = 20$ | 1 | 10,672 | 17,275 | 30,575 | 35,951 | 42,696 | 121,475 |
| $n = 25$ | 1 | 13,305 | 21,257 | 29,903 | 29,185 | 32,594 | 63,726 |
| $n = 30$ | 1 | 14,445 | 24,505 | 33,596 | 26,998 | 33,394 | 53,376 |

**Table 57.** Tabu search: average CPU-time (seconds)

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 0,00 | 3,97 | 6,00 | 8,77 | 10,99 | 16,66 | 28,89 |
| $n = 15$ | 0,00 | 8,39 | 11,80 | 16,31 | 21,49 | 34,46 | 82,81 |
| $n = 20$ | 0,00 | 13,50 | 14,91 | 26,30 | 34,49 | 46,25 | 98,77 |
| $n = 25$ | 0,02 | 23,66 | 32,05 | 43,04 | 46,89 | 59,00 | 100,46 |
| $n = 30$ | 0,02 | 23,86 | 32,40 | 45,14 | 46,47 | 61,57 | 99,40 |

**Table 58.** Tabu search: summary

| | Global summary | Unlimited number of modes |
|---|---|---|
| Average deviation from the best known solution | 0.27% | 1.13% |
| Maximum deviation from the best known solution | 8.33% | 8.33% |
| Number of times the best known solution is obtained | 4,532 (±86%) | 404 (±54%) |
| Average deviation from *lb* | 2.76% | 4.83% |
| Maximum deviation from *lb* | 42.86% | 16.67% |
| Problems solved to optimality | 2933 (±56%) | 79 (±11%) |
| Average number of RCPSP instances solved | 32,114 | 95,831 |
| Average CPU-time (in seconds) | 31.39 | 82.07 |

### 5.2.9. Effect of tabu list management, aspiration criteria, intensification and diversification

Tables 59 and 60 indicate the performance of the TS procedure starting from a steepest descent / mildest ascent procedure based on recency-based memory (tabu list) only, up to the full-fledged TS procedure including a randomly varying tabu list length, all proposed aspiration criteria, intensification and diversification strategies and an extended final intensification step. The extended final intensification phase consists of solving all RCPSP instances corresponding to the mode assignments that led to the minimal value for the project makespan (instead of using only the first one encountered in executing the procedure) to near-optimality using the procedure of Demeulemeester and Herroelen (1997a) for 1 second. A maximum of 1,000 such mode assignments is imposed. In order to prevent that one single mode assignment is stored (and investigated) several times, we use a hashing function of the form $h = rem(\sum_{i=1}^{n} p_i m_i , p_{i+1})$, where *rem(y,x)* denotes the remainder of the division of $y$ by $x$ and $p_i$ denotes the $i^{\text{th}}$ prime number (starting from 3), to determine which mode assignments have to be saved. If upon finding a mode assignment with a minimal value for the project makespan the associated hashing value has already been encountered before, the mode assignment is not withheld because of the risk that it has already been saved earlier.

**Table 59.** Impact of various components of the TS procedure

| | Basic TS procedure | Variable tabu list length | Aspiration criteria | Intensifi-cation | Diversifi-cation | Solutions evaluated but not visited | Extended final intensifi-cation |
|---|---|---|---|---|---|---|---|
| Avg. dev. from best sol. | 1.35% | 1.02% | 0.61% | 0.52% | 0.28% | 0.27% | 0.28% |
| Max. dev. from best sol. | 100.00% | 100.00% | 129.63% | 129.63% | 7.14% | 8.33% | 8.33% |
| Best solution | 3,522 (±67%) | 3,776 (±72%) | 4,112 (±78%) | 4,375 (±83%) | 4,514 (±86%) | 4,532 (±86%) | 4,525 (±86%) |
| Avg. dev. from *lb* | 3.86% | 3.52% | 3.11% | 3.01% | 2.77% | 2.76% | 2.77% |
| Max. dev. from *lb* | 107.69% | 107.69% | 138.46% | 138.46% | 42.86% | 42.86% | 42.86% |
| Optimal | 2406 (±46%) | 2521 (±48%) | 2613 (±50%) | 2858 (±54%) | 2932 (±56%) | 2933 (±56%) | 2936 (±56%) |
| Avg. RCPSPs solved | 39,489 | 38,790 | 38,303 | 30,562 | 31,956 | 32,114 | 31,911 |
| Avg. CPU-time | 34.35 | 34.34 | 34.64 | 31.67 | 30.81 | 31.39 | 31.14 |

**Table 60.** Impact of various components of the TS procedure for unlimited number of modes

| | Basic TS procedure | Variable tabu list length | Aspiration criteria | Intensifi-cation | Diversifi-cation | Solutions evaluated but not visited | Extended final intensification |
|---|---|---|---|---|---|---|---|
| Avg. dev. from best sol. | 3.53% | 2.71% | 2.11% | 2.00% | 1.17% | 1.13% | 1.20% |
| Max. dev. from best sol. | 100.00% | 100.00% | 129.63% | 129.63% | 7.14% | 8.33% | 8.33% |
| Best solution | 284 (±38%) | 344 (±46%) | 439 (±59%) | 444 (±59%) | 395 (±53%) | 404 (±54%) | 389 (±52%) |
| Avg. dev. from *lb* | 7.32% | 6.47% | 5.85% | 5.73% | 4.87% | 4.83% | 4.90% |
| Max. dev. from *lb* | 107.69% | 107.69% | 138.46% | 138.46% | 16.67% | 16.67% | 16.67% |
| Optimal | 53 (±7%) | 71 (±9%) | 80 (±11%) | 75 (10%) | 76 (±10%) | 79 (±11%) | 75 (10%) |
| Avg. RCPSPs solved | 105,556 | 105,233 | 102,820 | 86,862 | 97,397 | 95,831 | 95,928 |
| Avg. CPU-time | 83.44 | 83.79 | 84.17 | 80.12 | 81.40 | 82.07 | 81.84 |

Clearly, all proposed extensions to the basic TS procedure (except the extended final intensification) enhance the global performance of the algorithm. The basic TS procedure based on recency-based memory (tabu list) only, although significantly superior to single-pass pure descent methods, is not able to outperform iterated descent methods. Only the introduction of aspiration criteria allows the TS procedure to outperform all other local search procedures. This result is conform with many similar results from the literature, in which rather simple TS procedures based on tabu lists and aspiration criteria only already outperform pure descent procedures.

However, the maximum deviations with respect to the lower bound and the best known solution are much worse. Equipping a TS procedure with aspiration criteria may sometimes cause the algorithm to quickly get stuck in local optima of poor quality. Especially aspiration by influence or aspiration by strong admissibility, which do not guarantee that cycling will not occur, may lead to a premature termination of the procedure. That is why only after including a diversification strategy, the maximum deviations (which will prevent indefinite cycling through the use of penalties in the move evaluation) are reduced to an acceptable level. The maximum deviations of pure descent methods (without random restarts) are considerably better than when using a steepest descent / mildest descent approach without diversification, mainly because the initial solution is different (based on the modes with the smallest associated duration), which leads to relatively good initial local optima out of which, however, the pure descent procedure seldom escapes. Starting pure descent methods with the initial solution used for the tabu search leads to substantially worse results (even worse than the basic TS procedure without any enhancements).

The required computational effort does not increase when either aspiration criteria or intensification and diversification strategies are introduced. Extending the TS procedure with an extended final intensification phase does not lead to a significantly superior performance (mainly because of the increased computational effort per iteration which leads to less iterations within the given time limit). When there is no limit on the number of modes extended final intensification even leads to slightly worse results because then, the extra required computational effort for storing the local optima is highest. Therefore, we chose to only store the first best solution obtained so far and to run a near-optimal RCPSP procedure only for that problem instance.

### 5.2.10. Effect of problem characteristics

Fig. 1 shows the effect of *OS* on the computational complexity of the DTRTP represented by the average deviation of the solutions obtained by the TS procedure with respect to the best known solution. Although the deviation with respect to a lower bound can be used for comparing the performance of different algorithms on a problem set or on specific subsets of that problem set, it should never be used for investigating the impact of certain problem characteristics on the problem complexity, since these problem characteristics may have an impact on the lower bound itself rather than on the computational complexity of the problem. Therefore, we have opted for the deviation with respect to the best known solution. Similar observations can be made for the number of problems solved to optimality. As is clear from Fig. 1, *OS* has a negative impact on the DTRTP complexity: the higher *OS*, the easier the corresponding DTRTP instance, regardless of the number of modes.

**Fig. 1.** The effect of *OS* on the DTRTP complexity

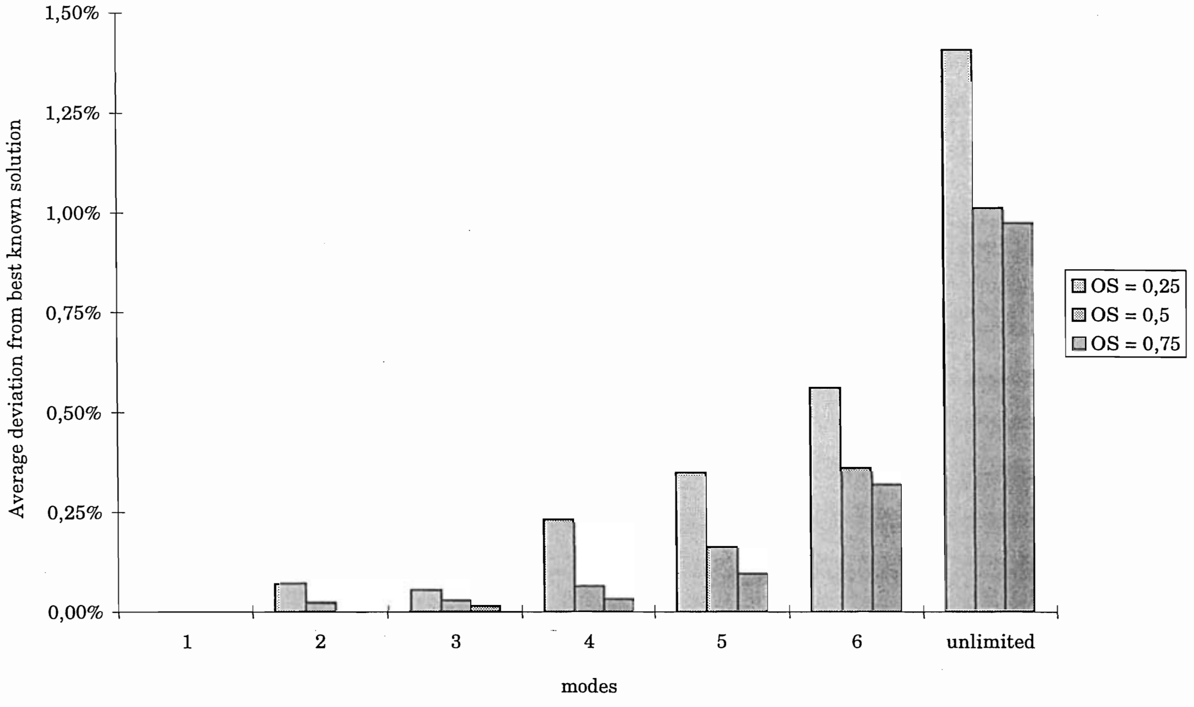

**Fig. 2.** The effect of resource availability *a* on the DTRTP complexity

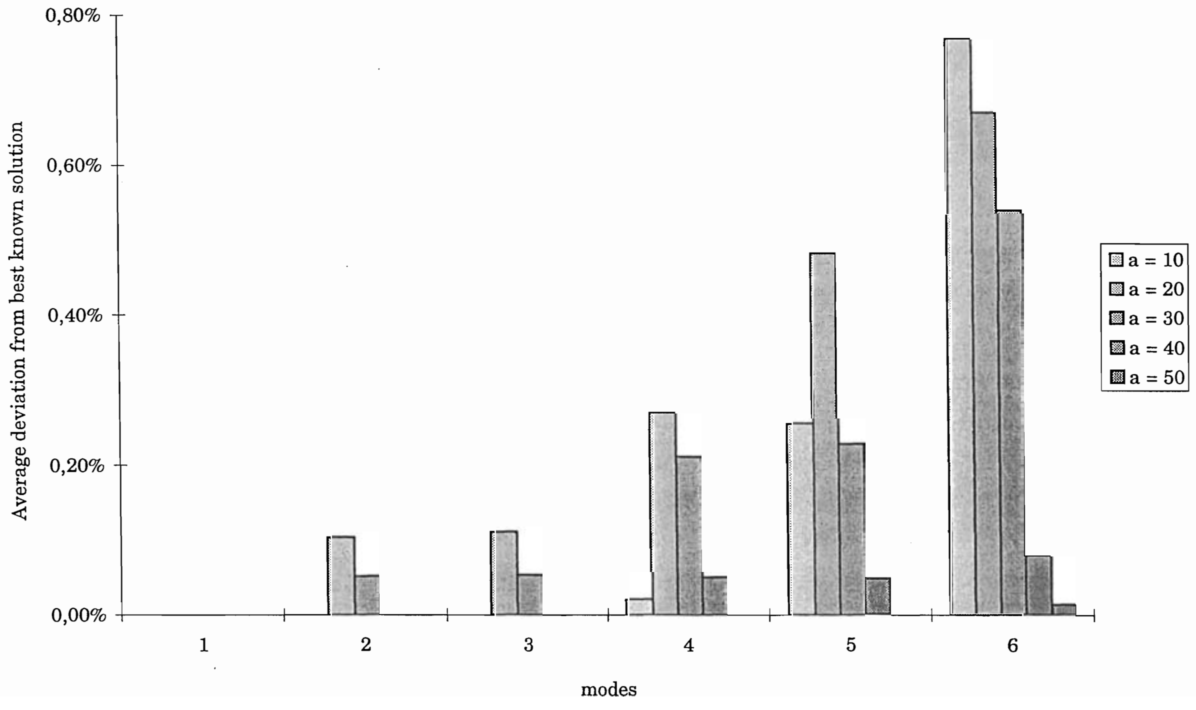

Fig. 2 shows the effect of the resource availability $a$ on the computational complexity of the DTRTP. The resource availability does not seem to have a monotonous impact on the DTRTP complexity. On the contrary, a kind of bell-shaped curve seems to result: when $a$ increases, the average deviations increase up to a certain point after which they decrease again. Only when the number of modes is high enough (namely 6), the resource availability has a negative impact on the DTRTP complexity (because probably the top of the bell-shaped curve occurs for small values of $a$). We did not include the results for an unlimited number of modes since then, the resource availability has a substantial impact on the number of modes (as given in Table 3), thereby polluting the real effect of $a$ on the DTRTP complexity.

5.2.11. Restricting the number of modes

Table 61 reports the average deviation from the best known solution, where the best known solution for each problem instance is defined as the best solution encountered using all the local search algorithms *with an unlimited allowed number of modes*. Therefore, Table 61 indicates how a restriction on the number of modes influences the quality of the obtained solutions relative to the absence of such a restriction. The very high deviations indicate that is not wise to restrict the number of allowed modes a priori without there being a justifiable reason. Specific modes may be excluded from consideration if they represent unrealistic modes (e.g. a duration of 100 with resource requirement 1) or if executing the activity using the proposed mode is not feasible (dictated by the actual conditions). However, limiting the number of modes in order to decrease the computational complexity may have a drastic impact on the quality of the obtained solutions. As Table 62 indicates, the average deviation from the best known solution due to a restriction of the number of modes increases when the resource availability increases. When the resource availability is high, "extreme" modes with a very small duration, which can only be obtained when there is no limit on the number of modes, become very important in finding the optimal schedule. Highly resource-constrained projects do not suffer as much from a restriction on the number of modes.

**Table 61.** Restriction of number of modes: average deviation from best known solution

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $n = 10$ | 78.52% | 55.27% | 54.09% | 34.61% | 34.03% | 17.96% | 0.19% |
| $n = 15$ | 91.42% | 65.66% | 64.78% | 42.28% | 41.79% | 22.64% | 0.73% |
| $n = 20$ | 102.53% | 75.84% | 74.99% | 51.04% | 50.66% | 30.11% | 1.21% |
| $n = 25$ | 85.06% | 60.03% | 59.32% | 37.80% | 37.48% | 20.03% | 1.70% |
| $n = 30$ | 93.40% | 67.27% | 66.55% | 43.55% | 43.21% | 24.09% | 1.83% |

**Table 62.** Restriction of number of modes: average deviation from best known solution

| | 1 mode | 2 modes | 3 modes | 4 modes | 5 modes | 6 modes | unlimited |
|---|---|---|---|---|---|---|---|
| $a = 10$ | 17.46% | 7.69% | 6.27% | 3.96% | 2.72% | 1.42% | 1.29% |
| $a = 20$ | 31.33% | 16.90% | 14.41% | 6.16% | 5.45% | 2.52% | 1.04% |
| $a = 30$ | 79.61% | 53.98% | 53.63% | 30.11% | 29.99% | 11.91% | 1.18% |
| $a = 40$ | 134.56% | 100.03% | 99.94% | 65.89% | 65.84% | 35.32% | 1.22% |
| $a = 50$ | 187.98% | 145.48% | 145.48% | 103.17% | 103.17% | 63.65% | 0.95% |

### 5.2.12. Summary

Tables 63 and 64 summarize the results for all local search procedures. Table 63 reports the global averages over all problem instances, whereas Table 64 only indicates the results when the number of modes is unlimited. The proposed TS procedure clearly outperforms all other local search methods by a large margin. For more than 99% of all problem instances (5200 out of 5250), tabu search is able to match or improve upon the best solution obtained with all other local search procedures. The truncated complete enumeration procedure performs very badly, even worse than the fastest and steepest descent methods without random restarts. The random procedure, although based on generating hundreds of thousands of mode assignments (and sometimes more than 1 million!), performs the worst. Therefore, based on this evidence, it is dangerous to only use random solutions as a benchmark, even when a very large number of such random solutions are considered. The relative difference in performance between iterated descent procedures and tabu search is in line with the results obtained by Mooney and Rardin (1992), who developed local search procedures for task assignment problems under resource constraints. They conclude that, although iterated descent procedures obtain high diversification levels as a result of the restarting procedure, they perform rather poorly compared to a TS procedure. Therefore, diversification appears to be a necessary but not a sufficient condition for obtaining high-quality solutions.

The results for the branch-and-bound procedure are quite promising when compared to the local search methods. Not only the number of problems solved to optimality but also the quality of the obtained heuristic solutions are significantly higher when compared to all other procedures. There are, however, three drawbacks. First, the maximum deviations from the best known solution and the lower bound are worse than for the TS procedure (and the iterated descent methods), which is mainly due to the fact that the branch-and-bound procedure is sometimes truncated upon finding a first feasible solution, which cannot always be guaranteed to be of high quality. Second, the memory requirements of the branch-and-bound procedure are much higher (some 15 Mb versus 1.3 Mb for the local search algorithms). Third, a feasible solution cannot always be guaranteed after a small CPU-time limit. Sometimes more than 1500 seconds are needed for finding a first feasible solution, whereas the local search procedures can be truncated already after a few seconds of running time. However, it is clear that the branch-and-bound procedure is a viable alternative for solving the DTRTP, even for fairly large problems with up to 30 activities and 15 modes per activity.

**Table 63.** Global summary

| | Truncated complete enumeration | Random procedure | Fastest descent | Steepest descent | Fastest iterated descent | Steepest iterated descent | Tabu search | Branch-and-bound |
|---|---|---|---|---|---|---|---|---|
| Avg. dev. from best sol. | 3.20% | 6.69% | 2.49% | 2.47% | 0.84% | 0.97% | 0.27% | 0.12% |
| Max. dev. from best sol. | 43.18% | 186.67% | 27.78% | 27.78% | 10.00% | 10.00% | 8.33% | 21.05% |
| Best solution | 3,315 (±63%) | 2,159 (±41%) | 2,920 (±56%) | 2,983 (±57%) | 3,746 (±71%) | 3,684 (±70%) | 4,532 (±86%) | 5,141 (±98%) |
| Avg. dev. from *lb* | 5.79% | 9.41% | 5.05% | 5.03% | 3.35% | 3.48% | 2.76% | 2.60% |
| Max. dev. from *lb* | 48.84% | 186.67% | 50.00% | 50.00% | 42.86% | 42.86% | 42.86% | 42.86% |
| Optimal | 2709 (±52%) | 1767 (±34%) | 2642 (±50%) | 2687 (±51%) | 2879 (±55%) | 2855 (±54%) | 2933 (±56%) | 4922 (±94%) |
| Avg. RCPSPs solved | 787,224 | 608,882 | 127 | 204 | 65,603 | 168,355 | 32,114 | — |
| Avg. CPU-time | 44.40 | 67.58 | 1.01 | 1.71 | 28.83 | 34.13 | 31.39 | 9.74 |

**Table 64.** Summary for unlimited number of modes

| | Truncated complete enumeration | Random procedure | Fastest descent | Steepest descent | Fastest iterated descent | Steepest iterated descent | Tabu search | Branch-and-bound |
|---|---|---|---|---|---|---|---|---|
| Avg. dev. from best sol. | 6.70% | 27.38% | 6.17% | 5.75% | 2.27% | 2.62% | 1.13% | 0.81% |
| Max. dev. from best sol. | 24.00% | 186.67% | 16.67% | 19.05% | 10.00% | 10.00% | 8.33% | 21.05% |
| Best solution | 61 (±8%) | 13 (±2%) | 28 (±4%) | 32 (±4%) | 246 (±33%) | 227 (30%) | 404 (±54%) | 645 (86%) |
| Avg. dev. from *lb* | 10.64% | 32.42% | 10.05% | 9.62% | 6.01% | 6.37% | 4.83% | 4.51% |
| Max. dev. from *lb* | 33.33% | 186.67% | 28.57% | 28.57% | 16.67% | 16.67% | 16.67% | 27.78% |
| Optimal | 20 (±3%) | 7 (±1%) | 4 (±1%) | 6 (±1%) | 49 (±7%) | 51 (±7%) | 79 (±11%) | 450 (60%) |
| Avg. RCPSPs solved | 2,432,985 | 1,116,687 | 404 | 649 | 258,519 | 702,453 | 95,831 | — |
| Avg. CPU-time | 97.83 | 99.39 | 2.30 | 2.60 | 75.89 | 81.44 | 82.07 | 60.74 |

## *6. Conclusions and suggestions for future research*

In this paper we present a tabu search (TS) procedure for the discrete time/resource trade-off problem in project networks. The TS procedure is based on subdividing the problem into a mode assignment phase and a resource-constrained project scheduling phase with fixed mode assignments. The computational results indicate that the TS procedure clearly outperforms other local search methods such as iterated descent, truncated complete enumeration or a random approach. In more than 99% of all problem instances the TS procedure is able to find the best solution obtained by all local search procedures. Even a rather simple TS procedure based on recency-based memory and aspiration criteria only already outperforms other local search methods. Equipping the TS procedure with intensification and diversification strategies substantially improves its performance. The branch-and-bound procedure developed by Demeulemeester et al. (1997) is capable of solving many of the test problems to optimality within an acceptable amount of time. A truncated version is able to outperform all local search methods presented in this paper. However, the branch-and-bound procedure sometimes requires relatively high computation times and memory, even for just finding a feasible solution.

A major advantage of TS versus more rigid procedures of the branch-and-bound type is its flexibility and versatility in the sense that it can easily be adapted to other assumptions and different problem types. One possible extension would be the application of a similar TS procedure to the MRCPSP. The major difference between the DTRTP and the MRCPSP is the fact that the mode assignment problem becomes NP-hard when there are at least two nonrenewable resource types. Therefore, finding a feasible solution for the MRCPSP or, equivalently, a feasible solution for the mode assignment phase in the TS procedure may become a very difficult task. Another possible extension would be to include other than zero-lag finish-start precedence relations, leading to the DTRTP with *precedence diagramming* (minimal start-start, start-finish, finish-start or finish-finish precedence relations) or with *generalized precedence relations* (minimal *and maximal* start-start, start-finish, finish-start or finish-finish precedence relations). This extension would require the use of a procedure for the GRCPSP (Demeulemeester and Herroelen, 1997b) or the RCPSP-GPR (De Reyck and Herroelen, 1997) for evaluating each mode assignment. Finally, the proposed TS procedure can be adapted for other regular and non-regular objective functions such as minimizing project costs, optimizing due-date performance and maximizing the net present value of the project given positive or negative cash flows associated with the activities. Basically, only the move evaluation phase and the lower bound calculations have to be modified. These problem types definitely constitute areas for future research.

*References*

Ahn, T. and S.S. Erengüç, 1995, "Resource-constrained project scheduling with multiple crashable modes: An exact solution method", Paper presented at the INFORMS New Orleans Fall 1995 Meeting, October 29-November 1.

Ahn, T. and S.S. Erengüç, 1996, "The resource-constrained project scheduling problem with multiple crashable modes: A heuristic procedure", Paper presented at the Fifth International Workshop on Project Management and Scheduling, Poznan (Poland), April 11-13.

Boctor, F., 1993, "Heuristics for scheduling projects with resource restrictions and several resource-duration modes", *International Journal of Production Research*, 31, 2547-2558.

Boctor, F., 1996, "A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes", *European Journal of Operational Research*, Special Issue on Project Management and Scheduling (W. Herroelen and E. Demeulemeester (Eds)), 90, 349-361.

Boctor, F., 1997, "An adaptation of the simulated annealing algorithm for solving resource-constrained project scheduling problems", *International Journal of Production Research*, to appear.

Dammeyer, F. and S. Voss, 1993, "Dynamic tabu list management using the reverse elimination method", *Annals of Operations Research*, 41, 31-46.

Dar-El, E.M., 1973, "MALB - A heuristic technique for balancing large single-model assembly lines", *AIIE Transactions*, 5, 343-356.

De, P., E.J. Dunne, J.B. Gosh and C.E. Wells, 1992, "Complexity of the discrete time-cost tradeoff problem for project networks", Tech. Report, Dept. MIS and Dec. Sci., University of Dayton.

De, P., E.J. Dunne, J.B. Gosh and C.E. Wells, 1995, "The discrete time-cost trade-off problem revisited", *European Journal of Operational Research*, 81, 225-238.

De Reyck, B., 1995, "On the use of the restrictiveness as a measure of complexity for resource-constrained project scheduling", Research Report 9535, Department of Applied Economics, Katholieke Universiteit Leuven.

De Reyck, B. and W. Herroelen, 1997, "A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations", *European Journal of Operational Research*, to appear.

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

Demeulemeester, E., W. Herroelen and S.E. Elmaghraby, 1996, "Optimal procedures for the discrete time/cost trade-off problem in project networks", *European Journal of Operational Research*, 88, 50-68.

Demeulemeester, E. and W. Herroelen, 1997a, "New benchmark results for the resource-constrained project scheduling problem", *Management Science*, to appear.

Demeulemeester, E. and W. Herroelen, 1997b, "A branch-and-bound procedure for the generalized resource-constrained project scheduling problem", *Operations Research*, to appear.

Demeulemeester, E., B. De Reyck and W. Herroelen, 1997, "A branch-and-bound procedure for the discrete time/resource trade-off problem in project networks", Research Report, Department of Applied Economics, Katholieke Universiteit Leuven.

Dell'Amico, M. and M. Trubian, 1993, "Applying tabu search to the job-shop scheduling problem", *Annals of Operations Research*, 41, 231-252.

Drexl, A. and J. Grünewald, 1993, "Nonpreemptive multi-mode resource-constrained project scheduling", *IIE Transactions*, 25 (5), 1993, 74-81.

Elmaghraby, S.E., 1977, *Activity networks: Project planning and control by network models*, John Wiley & Sons, New York.

Glover, F., 1989, "Tabu search, part I", *ORSA Journal on Computing*, 1, 190-206.

Glover, F., 1990a, "Tabu search, part II", *ORSA Journal on Computing*, 2, 4-32.

Glover, F., 1990b, "Tabu search: a tutorial", *Interfaces*, 20 (4), 74-94.

Glover, F. and M. Laguna, 1993, "Tabu search", in Reeves, C. (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford.

Hartmann, S. and A. Sprecher, 1993, "A note on 'Hierarchical models for multi-project planning and scheduling', Research report N° 338, Christian-Albrechts-Universität zu Kiel, Germany.

Herroelen, W., 1972, *Heuristische programmatie - Methodologische benadering en praktische toepassing op complexe combinatorische problemen*, Aurelia Books, Leuven.

Herroelen, W. and E. Demeulemeester, 1995, "Recent advances in branch-and-bound procedures for resource-constrained project scheduling problems", Chapter 12 in *Scheduling Theory and Its Applications* (Chrétienne, Ph. et al. (Eds)), John Wiley & Sons, Chichester.

Icmeli, O. and S.S. Erengüç, 1994, "A tabu search procedure for the resource constrained project scheduling problem with discounted cash flows", *Computers and Operations Research*, 21, 841-853.

Kao, E.P.C. and M. Queyranne, 1982, "On dynamic programming methods for assembly line balancing", *Operations Research*, 30, 375-390.

Kolisch, R., 1995, *Project scheduling under resource constraints - Efficient heuristics for several problem cases*, Physica-Verlag.

Kolisch, R., A. Sprecher and A. Drexl, 1995, "Characterization and generation of a general class of resource-constrained project scheduling problems", *Management Science*, 41 (10), 1693-1703.

Lee, J.-K. and Y.-D. Kim, 1996, "Search heuristics for resource-constrained project scheduling", *Journal of the Operational Research Society*, 47, 678-689.

Mastor, A. A., 1970, "An experimental and comparative evaluation of production line balancing techniques", *Management Science*, 16 (11), 728-746.

Mooney, E.L. and L. Rardin, 1992, "Tabu search for a class of scheduling problems", *Annals of Operations Research*.

Nudtasomboon, N. and S.U. Randhawa, 1997, "Resource-constrained project scheduling with renewable and nonrenewable resources and time/resource trade-offs", *Computers and Industrial Engineering*, 32, 227-242.

Özdamar and G. Ulusoy, 1994, "A local constraint based ananalysis approach to project scheduling under general resource constraints", *European Journal of Operational Research*, 79, 287-298.

Özdamar, L. and G. Ulusoy, 1995, "A survey on the resource-constrained project scheduling problem", *IIE Transactions*, 27, 574-586.

Patterson, J.H., R. Slowinski, F.B. Talbot and J. Weglarz, 1989, "An algorithm for a general class of precedence and resource constrained scheduling problems", Chapter 1 in *Advances In Project Scheduling* (Slowinski, R. and J. Weglarz (Eds.)), Elsevier.

Patterson, J.H., F.B. Talbot, R. Slowinski and J. Weglarz, 1990, "Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems", *European Journal of Operational Research*, 49, 68-79.

Pinson, E., C. Prins and F. Rullier, 1994, "Using tabu search for solving the resource-constrained project scheduling problem", Paper presented at the Fourth International Workshop on project Planning and Scheduling, Leuven, July 12-15.

Schwindt, C., 1995, "ProGen/max: a new problem generator for different resource-constrained project scheduling problems with minimal and maximal time lags", Technical Report WIOR-449, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe.

Slowinski, R., B. Soniewicki and J. Weglarz, 1994, "DSS for multiobjective project scheduling", *European Journal of Operational Research*, 79, 220-229.

Speranza, M.G. and C. Vercellis, 1993, "Hierarchical models for multi-project planning and scheduling", *European Journal of Operational Research*, 64, 312-325.

Sprecher, A., 1994, *Resource-constrained project scheduling - Exact methods for the multi-mode case*, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin.

Sprecher, A. and A. Drexl, 1996a, "Solving multi-mode resource-constrained project scheduling problems by a simple, general and powerful sequencing algorithm. Part I: Theory", Research report 385, Christian-Albrechts-Universität zu Kiel, Germany.

Sprecher, A. and A. Drexl, 1996b, "Solving multi-mode resource-constrained project scheduling problems by a simple, general and powerful sequencing algorithm. Part II: Computation", Research report 386, Christian-Albrechts-Universität zu Kiel, Germany.

Sprecher, A., S. Hartmann and A. Drexl, 1994, "Project scheduling with discrete time-resource and resource-resource trade-offs", Research report 357, Christian-Albrechts-Universität zu Kiel, Germany.

Sung, C.S. and S.K. Lim, 1997, "A scheduling procedure for a general class of resource-constrained projects", *Computers and Industrial Engineering*, 32, 9-17.

Talbot, F.B., 1982, "Resource-constrained project scheduling problem with time-resource trade-offs: The nonpreemptive case", *Management Science*, 28, 1197-1210.

Thesen, A., 1977, "Measures of the restrictiveness of project networks", *Networks*, 7, 193-208.

Vaessens, R.J.M., 1995, Generalized job shop scheduling: complexity and local search, Doctoral thesis, Eindhoven University of Technology.

Vaessens, R.J.M., E.H.L. Aarts and J.K. Lenstra, 1996, "Job shop scheduling by local search", *INFORMS Journal on Computing*, 8, 302-317.

Yang, K.K. and J.H. Patterson, 1995, "Scheduling a resource-constrained project with alternative performance modes using simulated annealing", Paper presented at the INFORMS New Orleans Fall 1995 Meeting, October 29-November 1.