12-1998

Local search methods for the discrete time/resource trade-off problem in project networks

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

Erik DEMEULEMEESTER

Willy HERROELEN

Local Search Methods for the Discrete Time/Resource Trade-off Problem in Project Networks

Bert De Reyck, Erik Demeulemeester, Willy Herroelen
Operations Management Group, Department of Applied Economics, Katholieke Universiteit Leuven, Naamsestraat 69, B-3000 Leuven, Belgium

Received March 1997; revised January 1998; accepted 2 March 1998

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), in which the duration of the activities is a discrete, nonincreasing 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 a decomposition of 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. © 1998 John Wiley & Sons, Inc. Naval Research Logistics 45: 553–578, 1998

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 [25] and Özdamar and Ulusoy [36]). 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 [24] and Elmaghraby [19] 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., total

Correspondence to: W. Herroelen
Contract grant sponsor: National Science Foundation; contract grant number FWO G.0131.96

 CCC 0894-069X/98/060553-26

amount of labor) 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, nonincreasing function of the amount of a single *renewable* resource committed to it. Given the specified work content W_i for activity i ($1 \le i \le n$), all M_i efficient execution modes for its execution are determined based on time/resource trade-offs. Activity i, when performed in mode m ($1 \le m \le M_i$), 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 every other mode has either a higher duration or a higher resource requirement. Without loss of generality, we assume that the modes of each activity are sorted in the order of nondecreasing 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:

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

Subject to

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

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

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

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

where e_i (l_i) denotes the earliest (latest) start time of activity i based on the modes with the smallest duration, T is an upper bound on the project duration, and E represents the set of precedence relations. The objective function (1) minimizes the project makespan. The constraints given by (2) ensure that each activity is assigned exactly one mode and exactly

one start time. Equations (3) denote the precedence constraints. Equations (4) secure that the per-period availability of the renewable resource is met. Finally, Eqs. (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. [9]). 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 (De et al. [8]). Optimal procedures and computational experience have been presented by Demeulemeester et al. [12].

The DTRTP is also a subproblem of the *multimode resource-constrained project scheduling problem (MRCPSP)*, which includes time/resource and resource/resource trade-offs and renewable, nonrenewable, and doubly constrained resource types. As a generalization of the RCPSP, the MRCPSP is NP-hard (Kolisch [28]). *Optimal procedures* have been presented by Talbot [47], Patterson et al. [37, 38], Speranza and Vercellis [42], Sprecher [43], Sprecher, Hartmann, and Drexl [45], Sprecher and Drexl [44], Ahn and Erengüç [1], and Nudtasomboon and Randhawa [34]. All of them use implicit enumeration with branch-and-bound.

Heuristic solution procedures for the MRCPSP have been developed by Talbot [47], Drexl and Grünewald [18], Slowinski, Soniewicki and Weglarz [41], Boctor [3–5], Özdamar and Ulusoy [35], Kolisch [28], Yang and Patterson [51], Ahn and Erengüç [2], Kolisch and Drexl [30], and Sung and Lim [46]. Talbot [47] recommends the use of a truncated version of his exact enumeration procedure. Drexl and Grünewald [18] 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 [28] compared his multimode heuristic (MMH) to the heuristics of Talbot [47], Drexl and Grünewald [18], Boctor [3], and the truncated method of Sprecher [43]. He reached the conclusion that MMH outperforms every other heuristic except the truncated branch-and-bound procedure. Boctor [4] 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 [3]. Özdamar and Ulusoy [35] present a constraint-based heuristic with an exponential time complexity.

Simulated annealing has been tried by Slowinski, Soniewicki, and Weglarz [41], Yang and Patterson [51] and Boctor [5]. Boctor [5] concludes that the simulated annealing algorithm outperforms the heuristics presented in Boctor [3, 4]. Slowinski, Soniewicki,

and Weglarz [41] 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 [51] conclude that simulated annealing outperforms the backtracking algorithm of Patterson et al. [37, 38], in that it obtains the smallest mean project duration with less computational effort. Kolisch and Drexl [30] present a biased random sampling approach which outperforms a truncated version of the enumeration procedure of Talbot [47] and the procedure of Drexl and Grünewald [18]. Sung and Lim [46] have developed a branch-and-bound procedure using two lower bounds, which is incorporated in a two-phase heuristic method.

Despite the fact that excellent results have been reported using tabu search on (generalized) job shop scheduling problems (Vaessens [49] and Vaessens, Aarts, and Lenstra [50]), efforts to develop tabu search procedures for the RCPSP are rather sparse (Icmeli and Erengüç [26], Pinson, Prins, and Rullier [39], and Lee and Kim [31]) 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 (neighborhood) 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.* A similar local search methodology using biased random sampling has been devised by Kolisch and Drexl [30].

In most applications of so-called intelligent heuristic procedures (local search, Tabu Search, Simulated Annealing, Genetic Algorithms, etc.) for the RCPSP, a large amount of effort is required to check whether moves are precedence- and resource-feasible. The main advantage of using such a decomposition approach is that the neighborhood search only operates on the level of mode assignments. The difficult problem of obtaining precedence- and resource-feasible heuristic solutions (of high quality) for the resource-constrained project scheduling problem is not tackled with a neighborhood search procedure, but using a dedicated algorithm. Moreover, for any arbitrary mode assignment, a feasible schedule can be found.

3.2. Truncated Complete Enumeration

Enumerating all mode assignments and solving each resulting RCPSP instance to optimality leads to the optimal solution of the DTRTP. However, such an approach proves intractable because of (a) 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] and (b) the fact that the RCPSP is an NP-hard problem. Nevertheless, we will report computational experience with a truncated complete enumeration procedure which enumer-

ates a number of mode assignments and solves each RCPSP instance using a fast heuristic algorithm.

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 neighborhood of μ (all mode assignments μ_k in which exactly one activity is assigned another mode) and selects one of them for further exploration. The procedure then performs a *move* from one mode assignment to another one in which exactly one activity is assigned another mode. 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 [13] (an enhanced version of their original code presented in [11]) as a fast heuristic for solving the RCPSP. The procedure 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 1 *milli*second. 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 [13], which, if truncated after 1 s of CPU-time, provides high quality, near-optimal solutions.

3.3.1. Descent Methods

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 neighborhood of μ and selects the one with the smallest project makespan. Then, again the neighborhood is determined, and the best possible mode assignment selected. The steepest descent procedure terminates when no improving mode assignment can be found. A *fastest descent* algorithm (*first-fit*/*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.

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 (further referred to as *iterated* fastest or steepest descent). Since these types of local search methods are known to be highly sensitive to the initial solution, incorporating random restarts will undoubtedly produce better results.

3.3.2. 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 nonimproving 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.

3.3.3. Randomized Search

Another way of crossing boundaries of local optimality is provided by randomized search methods such as biased random sampling or simulated annealing. For the DTRTP, a randomized procedure based on random sampling behaves as follows. Given an initial mode assignment μ, a new mode assignment is randomly generated (possibly restricted to the neighborhood of μ) which is enforced or discarded based on the associated value of the objective function. Improving moves are immediately enforced. Nonimproving moves are accepted with a probability that decreases with the deterioration with respect to the incumbent solution.

Randomized procedures can also be implemented using a full scan of the neighborhood of μ as in a steepest descent approach. In that case, the acceptance probability of a move is determined based on the associated value of the objective function relative to the objective function values of the other eligible moves. Whereas the sampling approach described above will typically be used in simulated annealing procedures, a full neighborhood scan approach is typical for biased random sampling procedures.

In the case of simulated annealing, the probability of accepting a deteriorating move (determined by the parameter α) 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, in subsequent iterations, 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 nonimproving 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 (Vaessens, Aarts, and Lenstra [50]). Moreover, several research results indicate tabu search to outperform randomized procedures such as simulated annealing (Pinson, Prins, and Rullier [39] and Dell'Amico and Trubian [10]). As a benchmark, however, Section 5.3 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 RCPSP procedure of Demeulemeester and Herroelen [13].

4. A TABU SEARCH PROCEDURE

4.1. Neighborhood

We define the neighborhood of a specific mode assignment μ as consisting of all mode assignments μ_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 neighborhood structure, the *connectivity property* holds, i.e., there exists, for each solution, a finite sequence of moves leading to the global optimal solution.

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 as 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 4.5.2) will ensure that no indefinite cycling will occur, even if previously visited solutions are revisited. 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. 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. 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 [20, 21] and Dammeyer and Voss [6]). Preliminary experimentation has revealed that the best performance is achieved with a tabu list length which depends on the size of the problem.

However, the length should not increase linearly with the problem size. Therefore, we set the average length to $2\sqrt{n}$, which corresponds to a tabu list length varying from 6 to 11 when the number of activities ranges from 10 to 30 as in the experiments reported on in Section 5 (similar values have been reported to work well for other problem types). Moreover, we allow the length to vary randomly in the interval $[\sqrt{n}, 3\sqrt{n}]$. This allows the tabu list length to decrease when it is too high (thereby eliminating too many moves), and to increase when it is too small (leading to cycling). When 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 or remains the same (each with the same probability).

4.3. Aspiration Criteria

Aspiration criteria are introduced to determine which tabu restrictions should be overridden, thereby revoking 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. 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 [23]), which means that the move is eligible for selection if no other non-tabu improving move exists. Several aspiration criteria have been proposed in the literature (for an overview, see Glover and Laguna [23]), of which the following are implemented in our TS procedure.

4.3.1. 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 as 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 until an admissible move exists. When the length of the tabu list is set too long (such that every possible move is classified as tabu), aspiration by default will automatically decrease the tabu list length.

4.3.2. Aspiration by Objective

If a move that is set tabu would lead to the best solution obtained so far, the tabu status is revoked, 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 tabu move would lead to the best solution obtained 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.

4.3.3. Aspiration by Influence

Moves can be classified according to their *influence*, i.e., their 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 favored 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 favor 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 as tabu and the current time (iteration), a move of higher influence has been chosen.

We also favor 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. Moreover, 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.

4.3.4. Aspiration by Search Direction

In some cases, revisiting an earlier examined solution can be advantageous, provided that another search path leading out of that solution might be chosen (thereby preventing cycling). *Aspiration by search direction* provides a mechanism for ensuring that another path out of a previously encountered solution is chosen. For each implemented move, we store whether it was improving or not. We revoke the tabu status of a tabu move if (a) it improves upon the incumbent solution and (b) if the most recent move out of the new mode assignment also was an improving move. Consequently, we allow a revisit of an earlier examined solution. We revisit a local optimum examined earlier but we now choose another (nonimproving) path leading out of that local optimum.

4.3.5. Aspiration by Strong Admissibility

A move is labelled *strongly admissible* if it is eligible 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 [22]). If such a strongly admissible move was made prior to the most recent iteration during which a nonimproving 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 as tabu.

4.4. Termination Criteria

The procedure is terminated when (a) 10,000 iterations are performed, or (b) 1000 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 s 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 $lb_r = \lceil (\sum_{i=1}^{n} W_i)/a \rceil$, 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 referred to as the *redundant* work content).

4.5. Medium and Long-Term Frequency-Based Memory

4.5.1. Frequency Information

The core of a TS procedure 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, often outperforms 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 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 can be used in unison to achieve a better performance.

4.5.2. Diversification

Although the tabu list prevents the procedure from cycling between a number of solutions, it cannot prevent the search process from being confined to a small region of the solution space. We use frequency-based memory to detect when this occurs 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.

A first possibility is to modify the attractiveness of moves 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 measure that stores f_i, the frequency that a new mode was assigned to activity i. Moves concerning activities with small f_i are favored against moves pertaining to activities with large f_i-values. 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 improving moves exist.

A second possibility of applying frequency-based memory for the purpose of diversification is to divide the search process in different phases for the purpose of diversifying or intensifying the search. After an initial data collection phase, a diversification phase is started. This can be accomplished using residence frequencies which store information about the frequency f_{im} that, in the previously encountered mode assignments, an activity was executed in a specific designated 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.

4.5.3. 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 f_{im}. 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 (by setting those moves tabu that would alter the mode of such activities to another mode not belonging to the associated subset). This type of intensification strategy is often referred to as *reinforcement by restriction* (Glover and Laguna [23]), 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 can 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 4.5.2 was also based on reinforcement by restriction, albeit to diversify the search (often referred to as *selective diversification*) rather than intensify it.

4.5.4. 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 [23]). 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 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 the associated modes for that activity (reinforcement by restriction), thereby serving the goals of both intensification and diversification.

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

Control parameter	Value
Number of activities (n)	10; 15; 20; 25; 30
Activity work content (W_i)	[10, 100]
Number of initial and terminal activities	[2, 5]
Maximal number of predecessors and successors	3
Order strength (OS[a])	0.25; 0.50; 0.75

[a] Schwindt [40] uses an estimator for the restrictiveness (Thesen [48]) as a network complexity measure. However, De Reyck [16] has shown that this measure is identical to the order strength (Mastor [32]), the flexibility ratio (Dar-El [7]), and the density (Kao and Queyranne [27]). We will use *order strength* when referring to this measure.

4.5.5. 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. Set x equal to the number of the current iteration. 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 them to be used on computers with little available memory.

5.1. Benchmark Problem Set

Schwindt [40] extended ProGen, the problem generator for the RCPSP developed by Kolisch, Sprecher, and Drexl [29], 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

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

Control parameter	Value
Number of activities (n)	10; 15; 20; 25; 30
Order strength (OS)	0.25; 0.50; 0.75
Resource availability (a)	10; 20; 30; 40; 50
Number of modes (M)	1; 2; 3; 4; 5; 6; unlimited

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 efficient modes (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 5250 problem instances results.

An unlimited number of modes means that for each activity i, every duration/resource requirement pair (d_{im}, r_{im}) is allowed as long as their product is at least as large as its work content W_i. In that case, the number of modes for activity i depends on its work content W_i and the resource availability a. The higher W_i or a, the higher the number of modes for activity i. Table 3 indicates for the different resource availability settings, the average number of modes for the activities in the corresponding problem instances, as well as their theoretical minimum and maximum. For the entire problem set, the average number of modes when there is no restriction imposed equals 11.82.

When a restriction is imposed on the number of modes, it is enforced as follows. The procedure first generates the mode m with duration $d_{im} = \max\{\lfloor\sqrt{W_i}\rfloor, \lceil W_i/a\rceil\}$ and resource requirement $r_{im} = \lceil W_i/d_i\rceil$, where $\lfloor x\rfloor$ denotes the highest integer number equal to or smaller 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, respectively 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 actual working conditions. Contrary to the case of an unlimited number of modes, restricting the number of modes to 1, 2, 3, 4, 5, or 6 results in exactly the specified amount for each activity, except for $a = 10$ and $r_i = 11$, which only allows for five 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.

	$a = 10$	$a = 20$	$a = 30$	$a = 40$	$a = 50$
Average number of modes	8.84	11.60	12.49	12.92	13.26
Minimum number of modes	5	6	6	6	6
Maximum number of modes	10	15	16	17	18

5.2. A Branch-and-Bound Procedure

Demeulemeester, De Reyck, and Herroelen [15] have developed 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 which corresponds to the completion time of one or more activities, the algorithm evaluates feasible partial schedules PS_t (corresponding 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 partial schedule resulting from a specific activity–mode combination is evaluated by computing a critical path-based and a resource-based lower bound. The node with the smallest lower bound is selected for further branching at the next decision point t'. At time t' again all feasible maximal activity–mode combinations are enumerated under the assumption that earlier scheduled activities can be removed from the partial schedule. If they are not removed from the partial schedule, another mode can be selected provided that they are restarted at time t'. Backtracking occurs when a schedule is completed, when its lower bound exceeds (or equals) a previously determined upper bound on the project makespan or when a branch is to be fathomed by one of the proposed dominance rules. The procedure stops with the optimal solution upon backtracking to level 0 in the search tree.

Computational experience is reported in Demeulemeester, De Reyck, and Herroelen [15]. In order to compare the performance of the branch-and-bound procedure with the local search methods developed in this paper, we have recompiled the branch-and-bound procedure 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 s 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 1567 s, which is the maximum CPU-time required by the branch-and-bound procedure.

5.3. Basic Computational Results

For the local search methods, moves are evaluated by solving the RCPSP instance corresponding to the associated mode assignment using the RCPSP procedure of Demeulemeester and Herroelen [13], which is truncated after a very small amount of time (until 100 backtracking steps have been performed). Upon termination of the local search procedures, the mode assignment which led to the schedule with minimal makespan is further investigated by running the same RCPSP algorithm, truncated after 1 s and using the best solution obtained so far as an upper bound.

Tables 4 and 5 summarize the results. The results reported are the average and maximal deviation with respect to the best known solution, which is obtained using all procedures

Table 4. 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	3315 (63%)	2159 (41%)	2920 (56%)	2983 (57%)	3746 (71%)	3684 (70%)	4532 (86%)	5141 (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
Median CPU-time	0.10	100.00	0.01	0.01	0.12	0.49	1.75	0.01

Table 5. 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
Median CPU-time	100.00	100.00	0.13	0.17	100.00	100.00	100.00	19.51

presented in this paper (4929, i.e., about 94% of those solutions are known to be optimal). Also the number of times the best known solution is obtained is given. The average and maximal deviation with respect to the lower bound *lb* are given, as are the number of problems solved to optimality. For the local search procedures, 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. Finally, the average number of RCPSP instances solved and the average and median values for the required CPU-time are reported. Table 4 reports the global averages over all problem instances, whereas Table 5 only indicates the results when the number of modes is unlimited.

The truncated complete enumeration procedure enumerates all activity–mode combinations and evaluates each resulting RCPSP instance. The activity–mode combinations are enumerated starting from the modes with the smallest duration. For use as a benchmark, we have also tested a fully randomized procedure, in which a (large) number of random mode assignments are generated and the corresponding RCPSP instances solved. Both enumeration procedures are truncated upon finding the optimal solution or when the time limit is exceeded. This explains the rather high average CPU-times when only a few of the instances can be solved to optimality. The difference in the median values for random and complete enumeration is due to the fact that the enumeration procedure solves (just) more than half of the problems to optimality within a small time limit. Therefore, about half of the problem instances can be considered rather easy. When the number of modes is unlimited, the enumeration procedures can solve almost none of the instances to optimality, indicating the high degree of complexity of the problem set.

For the descent algorithms, the initial mode assignment is determined by assigning to each activity its mode with smallest duration (which resulted in the best performance). We also examined the performance of the descent procedures enhanced with a random restart engine. Unless restricted by the time limit of 100 s, 1000 mode assignments are determined randomly, after which the descent procedure is initiated. As was to be expected, equipping a descent procedure with random restarts significantly increases the quality of the obtained solutions, at the expense of increased computational requirements. Contrary to expectations, the overall quality of the solutions obtained with steepest descent do not significantly differ from those obtained using a fastest descent approach. Only for an unlimited amount of modes does steepest descent outperform fastest descent. However, the slight advantage of steepest descent versus fastest descent disappears completely when random restarts are introduced. In fact, iterated fastest descent performs better than iterated steepest descent. The main reason for this is the fact that a fastest descent approach requires less time to perform each iteration, thereby making it possible to perform more iterations and random restarts.

For the TS procedure, the initial mode assignment is determined by assigning to each activity the mode with duration $\max\{\lfloor\sqrt{W_i}\rfloor, \lceil W_i/a\rceil\}$. 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 [4]).

For the TS procedure, the initial solution does not have a significant impact on the quality of the obtained solutions. When the modes according to the smallest activity durations 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 optimality equals 2932 (vs. 2933). In general, the initial solution has a significant impact on the outcome for local search methods that do not perform a global optimization (such as single-pass fastest or steepest descent). However, a well-designed TS procedure that employs long-term memory for diversification purposes will strive for global optimization. Therefore, the initial solution has no significant impact because the diversification process will guide the search away from the initial region in the solution space. Only when no diversification aspects are included in the TS procedure, or when it is truncated after a very small amount of time (thereby preventing the diversification process to work as intended), will the initial solution have an influence on the quality of the obtained solution.

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. Truncated complete enumeration, despite the high computation time, leads to inferior solutions, which are, on the average, even worse than those of a single-pass fastest or steepest descent. The random procedure, although based on generating hundreds of thousands of mode assignments (sometimes millions), performs the worst. Therefore, based on this evidence, it is dangerous to only use randomly generated 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 [33], 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 s 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.

5.4. Detailed Computational Results

5.4.1. Impact of Problem Dimension

Figure 1 shows the effect of the number of modes and the number of activities on the average deviation of the solutions obtained by the TS procedure with respect to the best

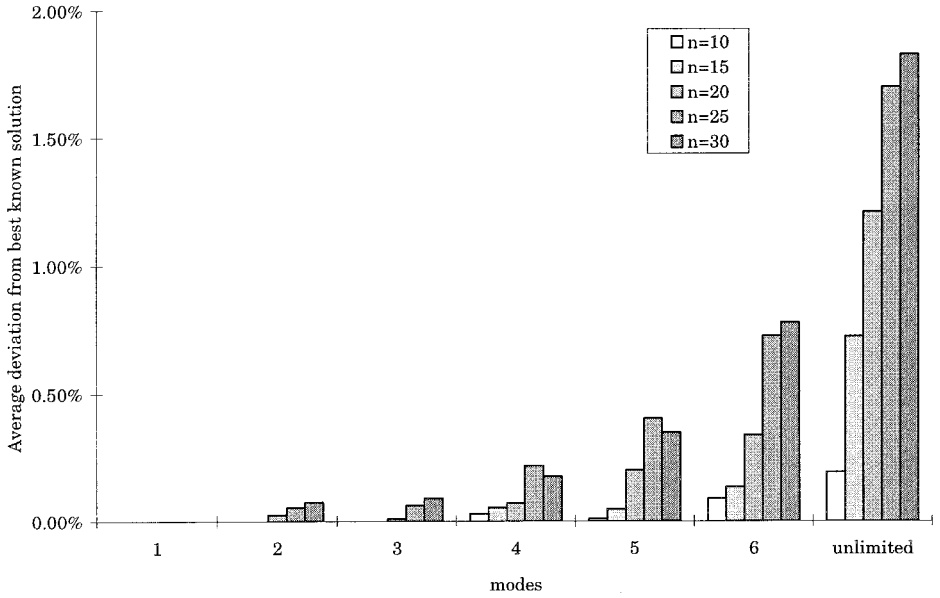

Figure 1. The effect of the number of activities and the number of modes on the DTRTP complexity.

known solution. Clearly, an increasing problem size caused by either an increase in the number of activities or the number of modes leads to a decrease in effectiveness. The main reason for this decrease in efficacy is due to the fact that the time limit of 100 s should be increased when dealing with larger instances. In that case, the decrease in effectiveness cannot be observed anymore.

5.4.2. Effect of Tabu List Management, Aspiration Criteria, Intensification, and Diversification

Tables 6 and 7 indicate the performance of TS starting from a steepest descent/mildest ascent procedure with recency-based memory, up to the full-fledged procedure with a randomly varying tabu list length, all aspiration criteria and intensification and diversification strategies. Clearly, all proposed extensions to the basic TS procedure enhance the overall 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 TS to outperform all other local search procedures. This result conforms to results from the literature in which TS procedures have been developed for solving a wide variety of combinatorial optimization problems. In many computational investigations, rather simple TS procedures based on tabu lists and aspiration criteria only already outperform pure descent procedures.

Equipping a TS procedure with aspiration criteria, however, 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 are reduced to an acceptable level. The

Table 6. Impact of various components of the TS procedure.

	Basic TS procedure	Variable tabu list length	Aspiration criteria	Intensification	Diversification	Solutions evaluated but not visited
Avg. dev. from best sol.	1.35%	1.02%	0.61%	0.52%	0.28%	0.27%
Max. dev. from best sol.	100.00%	100.00%	129.63%	129.63%	7.14%	8.33%
Best solution	3522 (67%)	3776 (72%)	4112 (78%)	4375 (83%)	4514 (86%)	4532 (86%)
Avg. dev. from *lb*	3.86%	3.52%	3.11%	3.01%	2.77%	2.76%
Max. dev. from *lb*	107.69%	107.69%	138.46%	138.46%	42.86%	42.86%
Optimal	2406 (46%)	2521 (48%)	2613 (50%)	2858 (54%)	2932 (56%)	2933 (56%)
Avg. RCPSPs solved	39,489	38,790	38,303	30,562	31,956	32,114
Avg. CPU-time	34.35	34.34	34.64	31.67	30.81	31.39

Table 7. Impact of various components of the TS procedure for unlimited number of modes.

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

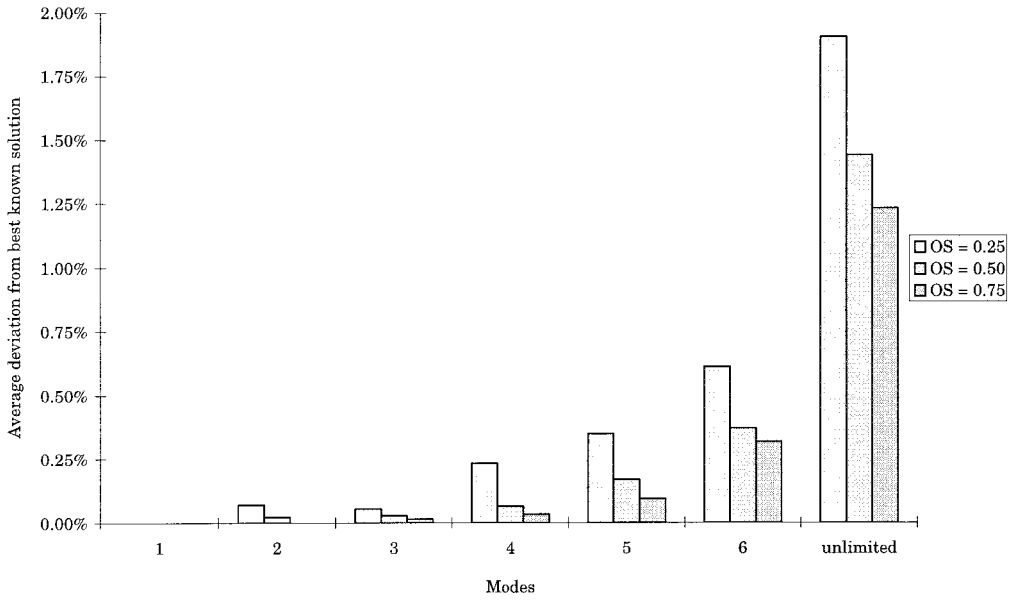

Figure 2. The effect of *OS* on the DTRTP complexity.

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. Notice that the required computational effort does not increase when aspiration criteria, intensification or diversification strategies are introduced.

5.4.3. Effect of Problem Characteristics

Figure 2 shows the effect of *OS* on the effectiveness of the TS procedure, represented by the average deviation of the obtained solutions with respect to the best known solution. Clearly, *OS* has a negative impact on the DTRTP complexity: the higher *OS*, the easier the corresponding DTRTP instance, regardless of the number of modes.

Figure 3 shows the effect of the resource availability *a* on the performance of TS. The resource availability does not seem to have a monotonous impact. 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 (namely six) does the resource availability have a negative impact on the quality of the solutions obtained with TS (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. In that case, 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.

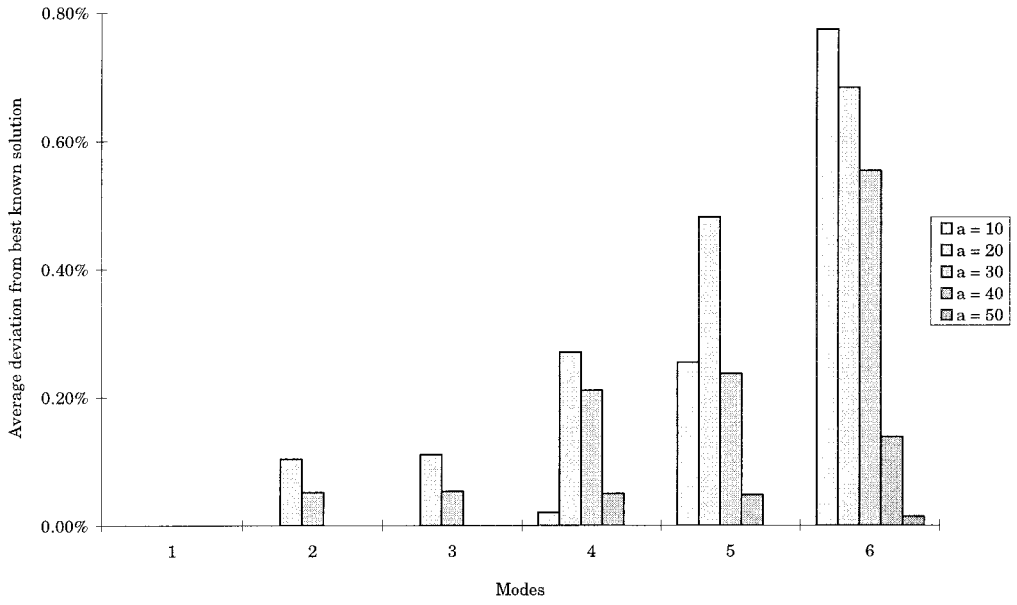

Figure 3. The effect of resource availability *a* on the DTRTP complexity.

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 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, De Reyck, and Herroelen [15] 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 [14]) or the RCPSP-GPR (De Reyck and Herroelen [17]) for evaluating each mode assignment. Finally, the proposed TS procedure can be adapted for other regular and nonregular objective functions such as minimizing project costs, optimizing due-date performance and maximizing the net present value of a project. Basically, only the move evaluation phase and the lower bound calculations have to be modified.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Ahn, T., and Erengüç, S.S., ''Resource-Constrained Project Scheduling with Multiple Crashable Modes: An Exact Solution Method,'' INFORMS New Orleans Fall 1995 Meeting, 29 October–1 November 1995.

[2] Ahn, T., and Erengüç, S.S., ''The Resource-Constrained Project Scheduling Problem with Multiple Crashable Modes: A Heuristic Procedure,'' in *Proceedings of the Fifth International Workshop on Project Management and Scheduling,* Poznan, Poland, 11–13 April 1996, pp. 23–26.

[3] Boctor, F., ''Heuristics for Scheduling Projects with Resource Restrictions and Several Resource–Duration Modes,'' *International Journal of Production Research,* **31,** 2547–2558 (1993).

[4] Boctor, F., ''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 (1996).

[5] Boctor, F., ''An Adaptation of the Simulated Annealing Algorithm for Solving Resource-Constrained Project Scheduling Problems,'' *International Journal of Production Research,* to appear (1998).

[6] Dammeyer, F., and Voss, S., ''Dynamic Tabu List Management Using the Reverse Elimination Method,'' *Annals of Operations Research,* **41,** 31–46 (1993).

[7] Dar-El, E.M., ''MALB—A Heuristic Technique for Balancing Large Single-Model Assembly Lines,'' *AIIE Transactions,* **5,** 343–356 (1973).

[8] De, P., Dunne, E.J., Gosh, J.B., and Wells, C.E., ''Complexity of the Discrete Time–Cost Tradeoff Problem for Project Networks,'' *Operations Research,* **45,** 302–306 (1997).

[9] De, P., Dunne, E.J., Gosh, J. B., and Wells, C.E., ''The Discrete Time–Cost Trade-off Problem Revisited,'' *European Journal of Operational Research,* **81,** 225–238 (1995).

[10] Dell'Amico, M., and Trubian, M., ''Applying Tabu Search to the Job-Shop Scheduling Problem,'' *Annals of Operations Research,* **41,** 231–252 (1993).

[11] Demeulemeester, E., and Herroelen, W., ''A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem,'' *Management Science,* **38,** 1803–1818 (1992).

[12] Demeulemeester, E., Herroelen, W., and Elmaghraby, S.E., ''Optimal Procedures for the Discrete Time/Cost Trade-off Problem in Project Networks,'' *European Journal of Operational Research,* **88,** 50–68 (1996).

[13] Demeulemeester, E., and Herroelen, W., ''New Benchmark Results for the Resource-Constrained Project Scheduling Problem,'' *Management Science,* **43,** 1485–1492 (1997).

[14] Demeulemeester, E., and Herroelen, W., ''A Branch-and-Bound Procedure for the Generalized Resource-Constrained Project Scheduling Problem,'' *Operations Research,* **45,** 201–212 (1997).

[15] Demeulemeester, E., De Reyck, B., and Herroelen, W., ''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, 1997.
[16] De Reyck, B., ''On the Use of the Restrictiveness as a Measure of Complexity for Resource-Constrained Project Scheduling,'' Research Report 9535, Department of Applied Economics, Katholieke Universiteit Leuven, 1995.
[17] De Reyck, B., and Herroelen, W., ''A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations,'' *European Journal of Operational Research,* to appear (1998).
[18] Drexl, A., and Grünewald, J., ''Nonpreemptive Multi-Mode Resource-Constrained Project Scheduling,'' *IIE Transactions,* **25,** 74–81 (1993).
[19] Elmaghraby, S.E., *Activity Networks: Project Planning and Control by Network Models,* Wiley, New York, 1977.
[20] Glover, F., ''Tabu Search, Part I,'' *ORSA Journal on Computing,* **1,** 190–206 (1989).
[21] Glover, F., ''Tabu Search, Part II,'' *ORSA Journal on Computing,* **2,** 4–32 (1990).
[22] Glover, F., ''Tabu Search: A Tutorial,'' *Interfaces,* **20,** 74–94 (1990).
[23] Glover, F., and Laguna, M., ''Tabu Search,'' in C. Reeves (Ed.), *Modern Heuristic Techniques for Combinatorial Problems,* Blackwell Scientific, Oxford, 1993.
[24] Herroelen, W., *Heuristische Programmatie—Methodologische Benadering en Praktische Toepassing op Complexe Combinatorische Problemen,* Aurelia Books, Leuven, 1972.
[25] Herroelen, W., and Demeulemeester, E., ''Recent Advances in Branch-and-Bound Procedures for Resource-Constrained Project Scheduling Problems,'' in Ph. Chrétienne et al. (Eds.), *Scheduling Theory and Its Applications,* Wiley, Chichester, 1995, Chap. 12.
[26] Icmeli, O., and Erengüç, S.S., ''A Tabu Search Procedure for the Resource Constrained Project Scheduling Problem with Discounted Cash Flows,'' *Computers and Operations Research,* **21,** 841–853 (1994).
[27] Kao, E.P.C., and Queyranne, M., ''On Dynamic Programming Methods for Assembly Line Balancing,'' *Operations Research,* **30,** 375–390 (1982).
[28] Kolisch, R., *Project Scheduling under Resource Constraints—Efficient Heuristics for Several Problem Cases,* Physica-Verlag, Heidelberg, 1995.
[29] Kolisch, R., Sprecher, A., and Drexl, A., ''Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems,'' *Management Science,* **41,** 1693–1703 (1995).
[30] Kolisch, R., and Drexl, A., ''Local Search for Nonpreemptive Multi-Mode Resource-Constrained Project Scheduling,'' *IIE Transactions,* **29,** 987–999 (1997).
[31] Lee, J.-K., and Kim, Y.-D., ''Search Heuristics for Resource-Constrained Project Scheduling,'' *Journal of the Operational Research Society,* **47,** 678–689 (1996).
[32] Mastor, A.A., ''An Experimental and Comparative Evaluation of Production Line Balancing Techniques,'' *Management Science,* **16,** 728–746 (1970).
[33] Mooney, E.L., and Rardin, L., ''Tabu Search for a Class of Scheduling Problems,'' *Annals of Operations Research,* **41,** 253–278 (1993).
[34] Nudtasomboon, N., and Randhawa, S.U., ''Resource-Constrained Project Scheduling with Renewable and Nonrenewable Resources and Time/Resource Trade-offs,'' *Computers and Industrial Engineering,* **32,** 227–242 (1997).
[35] Özdamar, L., and Ulusoy, G., ''A Local Constraint Based Analysis Approach to Project Scheduling under General Resource Constraints,'' *European Journal of Operational Research,* **79,** 287–298 (1994).
[36] Özdamar, L., and Ulusoy, G., ''A Survey on the Resource-Constrained Project Scheduling Problem,'' *IIE Transactions,* **27,** 574–586 (1995).
[37] Patterson, J.H., Slowinski, R., Talbot, F.B., and Weglarz, J., ''An Algorithm for a General Class of Precedence and Resource Constrained Scheduling Problems,'' in R. Slowinski and J. Weglarz (Eds.), *Advances in Project Scheduling,* Elsevier, Amsterdam, 1989, Chap. 1.
[38] Patterson, J.H., Talbot, F.B., Slowinski, R., and Weglarz, J., ''Computational Experience with a Backtracking Algorithm for Solving a General Class of Precedence and Resource-Constrained Scheduling Problems,'' *European Journal of Operational Research,* **49,** 68–79 (1990).
[39] Pinson, E., Prins, C., and Rullier, F., ''Using Tabu Search for Solving the Resource-Constrained

Project Scheduling Problem,'' in *Proceedings of the Fourth International Workshop on Project Planning and Scheduling,* Leuven, 12–15 July 1994, pp. 102–106.
[40] Schwindt, C., ''Generation of Resource-Constrained Project Scheduling Problems with Minimal and Maximal Time Lags,'' Technical Report WIOR-489, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe, 1995.
[41] Slowinski, R., Soniewicki, B., and Weglarz, J., ''DSS for Multiobjective Project Scheduling,'' *European Journal of Operational Research,* **79,** 220–229 (1994).
[42] Speranza, M.G., and Vercellis, C., ''Hierarchical Models for Multi-Project Planning and Scheduling,'' *European Journal of Operational Research,* **64,** 312–325 (1993).
[43] Sprecher, A., *Resource-Constrained Project Scheduling—Exact Methods for the Multi-Mode Case,* Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, 1994.
[44] Sprecher, A., and Drexl, A., ''Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm,'' *European Journal of Operational Research,* to appear (1998).
[45] Sprecher, A., Hartmann, S., and Drexl, A., ''An Exact Algorithm for Project Scheduling with Multiple Modes,'' *OR Spektrum,* **19,** 195–203 (1997).
[46] Sung, C.S., and Lim, S.K., ''A Scheduling Procedure for a General Class of Resource-Constrained Projects,'' *Computers and Industrial Engineering,* **32,** 9–17 (1997).
[47] Talbot, F.B., ''Resource-Constrained Project Scheduling Problem with Time-Resource Tradeoffs: The Nonpreemptive Case,'' *Management Science,* **28,** 1197–1210 (1982).
[48] Thesen, A., ''Measures of the Restrictiveness of Project Networks,'' *Networks,* **7,** 193–208 (1977).
[49] Vaessens, R.J.M., ''Generalized Job Shop Scheduling: Complexity and Local Search,'' Ph.D. Dissertation, Eindhoven University of Technology, 1995.
[50] Vaessens, R.J.M., Aarts, E.H.L., and Lenstra, J.K., ''Job Shop Scheduling by Local Search,'' *INFORMS Journal on Computing,* **8,** 302–317 (1996).
[51] Yang, K.K., and Patterson, J.H., ''Scheduling a Resource-Constrained Project with Alternative Performance Modes Using Simulated Annealing,'' INFORMS New Orleans Fall 1995 Meeting, 29 October–1 November 1995.