IIE Transactions (1997) **29**, 987–999

Local search for nonpreemptive multi-mode resource-constrained project scheduling

RAINER KOLISCH and ANDREAS DREXL

Institut für Betriebswirtschaftslehre, Lehrstuhl für Produktion und Logistik, Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, D-24118 Kiel, Federal Republic of Germany

This paper addresses a general class of nonpreemptive resource-constrained project scheduling problems in which activity durations are discrete functions of committed renewable and nonrenewable resources. We provide a well known 0–1 problem formulation and stress the importance of the model by giving applications within production and operations management. Furthermore, we prove that the feasibility problem is already NP-complete. Solution procedures proposed so far have the following shortcomings: exact methods can solve only very small instances to optimality; heuristic solution approaches fail to generate feasible solutions when problems become highly resource-constrained. Hence, we propose a new local search method that first tries to find a feasible solution and secondly performs a single-neighborhood search on the set of feasible mode assignments. To evaluate the new procedure we perform a rigorous computational study on two benchmark sets. The experiment includes a comparison of our procedure with other heuristics.

1. Introduction

We consider a project consisting of $j = 1, \ldots, J$ activities, a set R of renewable resources and a set N of nonrenewable resources. Each renewable resource $r \in R$ has a limited capactiy of $K_r \geq 0$ units per period; each nonrenewable resource $r \in N$ is restricted to $K_r \geq 0$ units within the whole planning horizon. An activity j is executable in one of its $m = 1, \ldots, M_j$ modes. The mode m determines uniquely the duration d_{jm}, the resource usage per period with respect to each renewable resource $k_{jmr} \geq 0$, and the resource consumption with respect to each nonrenewable resource $k_{jmr} \geq 0$, respectively. Activities are interrelated by technological or so-called precedence relations as well as by the conflicting demand of scarce resources. A feasible solution of the problem has to schedule each activity in one of its modes such that the precedence constraints and all resource constraints are maintained.

The problem outlined is a very general one and embodies a wide range of scheduling problems. In particular, the multi-mode and the single-mode version of the resource-constrained project scheduling problem (RCPSP), job shop, and flow shop type problems as well as scheduling problems with one and multiple parallel machines are included. Furthermore, the problem is important within automated manufacturing systems (see, for example, [1,2]), where tasks with alternative process plans are given. Each process plan specifies machines, tools, material handling carriers as well as their sequences. Finally, Kolisch [3] shows that the outlined problem also arises within production planning and control systems and in Leitstand systems.

Unfortunately, the problem is intrinsically hard to solve. Incorporating different modes into a given problem setting enlarges the solution space [4,5], which makes it more difficult to find good solutions. In addition, we will see in the next section that answering the question of whether there exists a feasible solution is an NP-complete problem when there are two or more (highly) constrained nonrenewable resources.

On account of the urgent need for solution strategies capable to generate feasible and good solutions with a polynomially bounded effort, we present a general local search method designed for problems with highly constrained nonrenewable resources.

The remainder of this paper is organized as follows: in the next section we provide a 0–1 programming model from the literature and prove the NP-completeness of the feasibility problem for $|N| \geq 2$. Afterwards, we give a review of several solution procedures. In Section 4 we propose our new solution method. Section 5 reports the results of a detailed computational study in which we benchmarked our procedure and two heuristics from the literature on two different instance sets. Section 6 is reserved for final conclusions.

2. Model and complexity results

To model the multi-mode multiple resource-constrained project scheduling problem (MRCPSP), we assume

without loss of generality that activities are topologically ordered, i.e., each activity j has an activity number that is larger than the number of all its immediate predecessors $i \in P_j$. We also assume that we have unique dummy start and finish activities, $j = 1$ and $j = J$, each only performable in a single mode associated with zero duration and zero resource demand, respectively. For all other activities we assume that modes are sorted in the order of non-decreasing duration. Given an upper bound T for the project's makespan, e.g., the sum of the maximum activity durations, earliest and latest finish times, EFT_j and LFT_j, can be calculated as follows. First, we assign to each activity its mode with shortest duration. Then we perform a traditional forward recursion to compute the earliest finish times EFT_j. Finally, we set $LFT_j = T$ and perform a traditional backward recursion to calculate the latest finish times LFT_j. Introducing the binary decision variables x_{jmt}, which equal 1 if activity j is scheduled in mode m to finish at the end of period t, 0 otherwise, we can now formulate the following model (see, for example, [6]):

$$\sum_{m=1}^{M_j} \sum_{t=EFT_j}^{LFT_j} x_{jmt} = 1, \quad j = 1, \cdots, J; \tag{1}$$

$$\sum_{m=1}^{M_i} \sum_{t=EFT_i}^{LFT_i} t\, x_{imt} \leq \sum_{m=1}^{M_j} \sum_{t=EFT_j}^{LFT_j} (t - d_{jm}) x_{jmt}, \quad j = 2, \ldots, J, \quad i \in P_j; \tag{2}$$

$$\sum_{j=2}^{J-1} \sum_{m=1}^{M_j} k_{jmr} \sum_{\tau=t}^{t+d_{jm}-1} x_{jm\tau} \leq K_r, \quad r \in R, \quad t = 1, \ldots, T; \tag{3}$$

$$\sum_{j=2}^{J-1} \sum_{m=1}^{M_j} k_{jmr} \sum_{t=EFT_j}^{LFT_j} x_{jmt} \leq K_r, \quad r \in N; \tag{4}$$

$$x_{jmt} \in \{0, 1\}, \quad j = 1, \ldots, J, \quad m = 1, \ldots, M_j, \quad t = EFT_j, \ldots, LFT_j. \tag{5}$$

Constraint set (1) ensures that each activity $j = 1, \ldots, J$ is performed in one of its modes and is finished within its time window $[EFT_j, LFT_j]$. Constraints (2) represent the precedence relations. The period capacity of the renewable resource types is maintained by constraint set (3). Constraints (4) limit the total resource consumption of nonrenewable resources to the available amount. Finally, constraints (5) define all decision variables to be binary. The commonly used objective function for (1)–(5) is the minimization of the makespan

$$\text{minmize } \phi = \sum_{t=EFT_J}^{LFT_J} t\, x_{J|t}. \tag{6}$$

Other objective functions, e.g., the minimization of the mean flow time or the maximization of the net present value, can be found in Slowinski [7].

The problem (1)–(6) is one of the most general and most difficult (project) scheduling problems. As a generalization of the well-known job shop problem it belongs to the class of NP-hard problems [8]. In addition, we now prove that answering the question of whether there exists a feasible solution for (1)–(5) is an NP-complete problem.

Theorem. *The feasibility problem* (1)–(5) *is* NP-*complete for* $|N| \geq 2$ *and* $M_j \geq 2,\ 1 < j < J$.

Proof. The problem is in NP because the feasibility of any solution x can obviously be checked in polynomial time. To prove that the problem is NP-complete, we polynomially transform the knapsack problem, which is known to be NP-complete [9], to (1)–(5).

The knapsack problem can be stated as follows: given a finite set U with elements u, $1 \leq u \leq |U|$, for each $u \in U$ a size $s(u) \in \boldsymbol{Z}^+$ and a value $v(u)$, a size $B \in \boldsymbol{Z}^+$ and a size $D \in \boldsymbol{Z}^+$, is there a subset $U' \subseteq U$ such that $\sum_{u \in U'} s(u) \leq B$ and $\sum_{u \in U'} v(u) \geq D$ hold?

The general idea of the transformation is to map each element of U into one non-dummy activity with two modes. If an activity is processed in mode $m = 1$ ($m = 2$) the corresponding u is (not) an element of the subset U'. Hence, $J = |U| + 2$, $M_j = 2$, $1 < j < J$, and $M_j = 1$, $j \in \{1, J\}$. The two knapsack constraints are represented by two nonrenewable resource constraints, i.e., $|N| = 2$, while renewable resource constraints do not have to be taken into account, i.e., $R = \varnothing$.

The first knapsack constraint can be considered straightforwardly. Hence, with respect to resource $r = 1$ the availability is $K_1 = B$, and for each activity $1 < j < J$ the resource consumption of the first mode equals $k_{j11} = s(u)$ while the resource consumption of the second mode is zero, i.e., $k_{j21} = 0$.

To consider the second knapsack constraint we have to rewrite it as follows. We multiply the constraint by -1, which gives $\sum -v(u) \leq -D$. Afterwards, we add $\max\{v(z) \mid z \in U\}$ to each summand on the left-hand side and $|U| \cdot \max\{v(z) \mid z \in U\}$ to the right-hand side to obtain non-negative values only. We now can proceed as for the first constraint; i.e., for resource $r = 2$ the resource consumption for each activity $1 < j < J$ is $k_{j12} = \max\{v(z) \mid z \in U\} - v(u)$ when performed in the first mode and $k_{j22} = \max\{v(z) \mid z \in U\}$ when processed in the second mode. The resource availability is $K_2 = |U| \cdot \max\{v(z) \mid z \in U\} - D$.

All other parameters of the MRCPSP are set to 0, i.e., $d_{jm} = 0\ \forall\ 1 < j < J,\ m = 1, \ldots, M_j$ and $P_j = \varnothing$ for $j = 2, \ldots, J$. Consequently, each instance of the knapsack problem can be polynomially transformed into an instance of the MRCPSP. A solution of the latter polynomially converts to a solution of the former as follows: if the first mode has been assigned to activity j, $1 < j < J$, i.e., $\mu(j) = 1$, the corresponding u is in U'. This proves

the NP-completeness of the feasibility problem of the MRCPSP.

3. Review of solution procedures

3.1. *Optimal procedures*

Optimal procedures for solving the MRCPSP have been presented in [6,10–13].

Talbot [6] was the first to present an enumeration scheme to solve the MRCPSP to optimality. Patterson *et al.* [10] refined this method by introducing a precedence tree that allowed a systematic enumeration of mode-assignments and start times. Their procedure solved the 10-activity instances (see Section 5.1) in an average computation time of 74.10 CPU-seconds on an IBM RS/6000 320 workstation [14].

Sprecher [12] enhanced this approach by four dominance criteria and one feasibility bound. This lowered the computational effort to an average computation time of 0.71 CPU-seconds on the same computer.

Speranza and Vercellis [11] proposed a depth-first oriented branch-and-bound procedure with a precedence-based lower bound. It has been shown by Hartmann and Sprecher [15] that for $|R| \geq 2$ this algorithm may not find the optimal solution and, moreover, that for instances with $|N| \geq 1$, the algorithm may even be unable to determine an existing feasible solution.

Finally, Sprecher *et al.* [13] extended the concept of delaying alternatives as introduced by Demeulemeester and Herroelen [16] for the single-mode RCPSP to mode and delaying alternatives. Their implementation solves the 10-activity instances in an average of 0.53 CPU-seconds on an IBM-compatible 80386DX personal computer with 40 MHz clock pulse. Despite these encouraging results it has to be recalled that exact algorithms in general fail to solve problems with more than 15 activities. Hence, even for small problems they are of limited use.

3.2. *Heuristic procedures*

Heuristic solution procedures for the MRCPSP have been provided in [6,17,18]. Özdamar and Ulusoy [19] propose an approximate procedure for the MRCPSP with $|N| = 1$. Solution methods for the MRCPSP with $|N| = 0$, i.e., without nonrenewable resources, were addressed in [20–23].

Talbot [6] recommends his exact method as a truncated enumeration procedure, which basically results in a priority-rule-driven serial scheduling method. Drexl and Grünewald [17] propose a regret-based biased random sampling approach that jointly employs a serial scheduling scheme and the SPT priority rule. Noteworthy is the fact that the start time of a chosen activity is determined with respect to precedence constraints only. Feasibility with respect to resource constraints is checked solely for the final schedule. The heuristic has been tested extensively on 100 instances with 10 activities, a network complexity of 1.5, three renewable and one nonrenewable resource types as well as different measures of resource scarcity. The best results were obtained with a probability mapping parameter of 2. The sample size, relying on the scarcity of resources, varied between 202 and 3779 in order to generate 100 feasible solutions for each problem. Depending on the scarcity of resources, an average performance of no more than 3.5% deviation from the optimal objective function value has been documented. The solution time varied between 17 and 321 CPU-seconds on an IBM-compatible personal computer with 8086 processor and 10 MHz clock-pulse.

Slowinski *et al.* [18] proposed a decision support system for a multiple objective MRCPSP embodying three different solution strategies: a single-pass approach, a multi-pass approach, and simulated annealing. The core of all solution strategies is a precedence-feasible activity list derived by one of 12 priority rules. The single-pass approach deterministically selects the next activity on the list and schedules it in the shortest resource-feasible mode at the earliest period possible. In contrast, the multi-pass approach randomly selects one of the next c precedence-feasible activities of the list for scheduling. Finally, the simulated annealing heuristic employs the activity list to represent a solution. The objective function is then calculated by applying the single-pass approach. A new neighbor is derived by interchanging the position of two activities that are not precedence-related. The DSS has been mainly proposed as a prototype and hence no rigorous computational experiments are reported.

Recently, Özdamar and Ulusoy [19] presented a constraint-based approach to solve the MRCPSP with $|N| = 1$. They employ a parallel scheduling method to decide via so-called essential conditions which activity–mode pairs have to be scheduled. More precisely, at each schedule time, all combinations of feasible activity–mode sets are enumerated. A combination consists of precedence-feasible activities performed in exactly one resource-feasible mode. For each combination, the increase over a lower bound of the project's makespan is calculated. The combination that induces the smallest increase over the lower bound is then scheduled by applying the essential conditions. The procedure has been tested on a set of 95 instances with 20–57 activities, one to six renewable and one nonrenewable resource types. The constraint-based procedure yielded an average increase over the precedence-based lower bound of 59%. The average computational requirement varied between 20 and 25 CPU-seconds on an IBM 70/386 PS/2 computer. Plain priority rules which were applied for comparative purposes yielded an average deviation of 65% within 7–9 CPU-seconds. The constraint-based approach has two disadvantages. First, the worst-case time complexity of

the procedure is exponential. Secondly, it is not suited for solving the MRCPSP with multiple scarce nonrenewable resource types.

Boctor [21–23] developed three solution strategies for the MRCPSP with renewable resource types only. The first heuristic, given in [21] is a single-pass approach that employs a parallel scheduling scheme. An activity is in the decision set if all its predecessors are finished and it can be started in at least one of its modes at the current schedule time. Activities are selected from the decision set in the order given by the MSLK priority rule. A chosen activity is scheduled in the mode with shortest duration. In [22], all possible activity–mode combinations that can be started at the schedule time are evaluated by applying a lower bound on the increase of the makespan. Finally, Boctor [23] performs a simulated annealing procedure similar to that proposed by Slowinski *et al.* [18]. On his own set of 240 test problems, with 50 and 100 activities and up to four renewable resource types, Boctor reports an average percentage deviation from the precedence-based lower bound of 36.8% for the single-pass procedure presented in [20], of 34.4% for the heuristic presented in [22], and of 26.5% for the simulated annealing method given in [23]. Computation times are not reported. Nevertheless, although the running time of the single-pass heuristic should be quite modest, the enumeration and evaluation of all possible activity–mode combinations within [22] should take substantially more time. For simulated annealing it is theoretically well known and has been experimentally shown by van Laarhoven *et al.* [24] that convergence to optimality may be achieved at the price of exponential running times.

4. A local search-based heuristic

4.1. *Outline of the solution method*

It has been shown above that the feasibility problem of the MRCPSP belongs to the class of NP-complete problems. Thus, solving a feasible MRCPSP instance heuristically does not guarantee a feasible solution. Drexl and Grünewald [17] pointed out that obtaining feasible solutions for hard MRCPSP instances is a formidable task. Currently, there are no heuristics developed especially to handle problems with multiple highly constrained nonrenewable resources. This knowledge was the primary motivation to design the solution method presented here.

We encode a solution for the MRCPSP by a mode-assignment M and a schedule S, denoted by (M,S). The mode-assignment M is a J-tuple $M=(\mu(1),\ldots,\mu(J))$ that assigns to every activity j a unique mode $\mu(j)$; the schedule S is a J-tuple $S=(FT_1,\ldots,FT_J)$ that assigns to every activity j a unique finish time FT_j. Following [25], we can outline our solution method as follows (see Table 1): In phase I we generate an initial mode assignment M. If M is feasible, we obtain the first *solution* (M,S) by applying any fast heuristic for the single-mode RCPSP to M. The *cost function* $\phi(M,S)$ is the makespan of the resulting schedule S. Phase II generates and evaluates a single neighbor of M and stores it as the current solution. This is repeated for a number of prescribed iterations. Finally, in phase III we use M^*, the best mode assignment of phase II, to derive an improved cost function for M^* with a near-optimal RCPSP heuristic.

The solution method described so far uses a general local search strategy. Given this basic framework, we can now apply many local-search-based solution strategies to the MRCPSP, e.g., simulated annealing, tabu search or biased random sampling. In what follows we apply the last concept. Let us add more detail to the two crucial points of our method: first, the generation of a feasible mode-assignment M within phase I and, secondly, the move from a feasible mode assignment M to a (feasible) neighbor M' within phase II. We discuss the former in Section 4.2 and the latter in Section 4.3.

4.2. *The construction phase: generation of an initial solution*

An initial mode assignment M is generated in two basic steps. First, to each of the activities is assigned the mode with the smallest relative resource consumption. Sec-

Table 1. Outline of the solution method

I. Construction phase: generation of an initial solution
Generate an initial mode assignment M.
If M is feasible then:
  Given M, generate a schedule S with a fast heuristic for the RCPSP.
  Save the solution (M,S) as the so-far best solution (M^*,S^*).
Else
  Goto IV
II. Local search phase: improvement of the mode-assignment
For a prescribed number of iterations do:
  Move from the feasible mode assignment M to a feasible neighbor M'.
  Given M', generate a schedule S' with a fast heuristic for the RCPSP.
  If $\phi(M',S')<\phi(M^*,S^*)$ then:
    Save it as the so-far best solution, $(M^*,S^*)=(M',S')$.
  Set $M=M'$.
III. Intensification Phase: Improvement of the Schedule
Given M^*, generate a schedule S with a near-optimal heuristic for the RCPSP.
If $\phi(M^*,S)<\phi(M^*,S^*)$ then:
  Save it as the so-far best solution, $(M^*,S^*)=(M^*,S)$.
IV Stop.

ondly, the left-over capacity of nonrenewable resources is allocated to the most promising activities to reduce the makespan. Let us use the following notation:

SJ	set of activities to which a mode has already been assigned;
NJ	set of activities to which a mode has not been assigned;
πK_r	left-over capacity of the nonrenewable resource r;
RC_{jm}	relative (nonrenewable) resource consumption of activity j in mode m;
SRC_j	smallest relative resource consumption of activity j;
LRC	largest relative resource consumption;
$\lambda(j)$	ordinal position of activity j within priority list λ.

The left-over capacity and the relative resource consumption of nonrenewable resources are defined as

$$\pi K_r = K_r - \sum_{j \in SJ} k_{j\mu(j)r} \,, \tag{7}$$

$$RC_{jm} = \sum_{r \in N} \frac{k_{jmr}}{\pi K_r} \,. \tag{8}$$

We define the difference in resource usage or consumption entailed by performing activity j in mode m instead of $\mu(j)$ as

$$\Delta k(j,m,r) = k_{jmr} - k_{j\mu(j)r},\ 1 \le m,\ \mu(j) \le M_j,\ m \ne \mu(j) \,. \tag{9}$$

Now, the formal description of the initial mode-assignment procedure (Init-MA) is as follows

Procedure Init-MA:
Initialization:
$SJ = \varnothing; NJ = \{1,\ldots,J\}; \pi K_r = K_r \forall r \in N;$ Calculate RC_{jm}, $j \in NJ, m \in \{1,\ldots,M_j\}$;
1. For $n = 1$ to J do:
   $SRC_j = \min\{RC_{jm} \mid 1 \le m \le Mj\},\ j \in NJ$;
   $LRC = \max\{SRC_j \mid j \in NJ\}$;
   $j^* = \min\{j \mid SRC_j = LRC, j \in NJ\}$;
   $\mu(j^*) = \min\{m \mid RC_{j^*m} = SRC_{j^*}, 1 \le m \le M_{j^*}\}$;
   $SJ = SJ \cup \{j^*\},\ NJ = NJ \backslash \{j^*\}$;
   Update $\pi K_r\ \forall\ r \in N,\ RC_{jm}, j \in NJ, m \in \{1,\ldots,M_j\}$;
2. If $\pi K_r < 0\ \exists r \in N$ then Stop.
   Perform forward recursion; $LFT_J = EFT_J$; perform backward recursion; $s_j = LFT_j - EFT_j,\ 1 \le j \le J$; sort activities according to non-decreasing s_j;
   Store the activity order in the list $\lambda(j), 1 \le j \le J$;
3. For $n = 1$ to J do:
   $j = \lambda(n); m = \mu(j-1)$;
   While $m \ge 1$ do :
      If $\pi K_r - \Delta k(j,m,r) \ge 0\ \forall r \in N$ then $\mu(j) = m$;
      $m = m - 1$;
   Update $\pi K_r\ \forall r \in N$;
Stop: if $\pi K_r \ge 0\ \forall r \in N$ then: a feasible mode-assignment M has been generated.

At the beginning of Init-MA, modes have not yet been fixed. Consequently, SJ is empty and the left-over capacity of each nonrenewable resource equals its overall capacity. Afterwards, we perform step 1 J times. Within each iteration we select one activity, assign to it one mode, and eliminate it from NJ, the set of available activities. To select an activity out of NJ, we calculate for every available activity and for each of its modes the relative resource consumption according to (8). We select the activity that maximizes the minimum relative resource consumption of its modes. To this activity is assigned the mode with the smallest relative resource consumption. In case of ties, the lowest labelled activity and mode are selected. Finally, the set of activities to which a mode has been assigned and the left-over capacities of nonrenewable resources are updated. The reasoning behind selecting the activity that maximizes the minimum relative resource consumption is as follows. From constraint (1) we know that each activity has to be performed in exactly one of its modes. Obviously, the mode with the minimum relative resource consumption is the best way of performing an activity to save nonrenewable resource capacity. Hence, by choosing the activity that maximizes the minimum relative resource consumption we decrease the left-over capacity of nonrenewable resources by an unavoidable amount as early as possible. This ensures that scarce resources are reflected properly to direct a feasible assignment of the not-yet assigned activities.

After a mode has been assigned to each of the activities, it may be that there is still left-over capacity of nonrenewable resources. To apportion this capacity efficiently with respect to our objective of minimizing the makespan, we rank the activities in step 2 in nondecreasing order of slack as calculated by traditional forward and backward recursion with each activity in the currently assigned mode. Thus, the first ranking activity has the minimum slack of all activities. Now, in step 3, to each activity on the list we try to assign the next smaller indexed mode until the first labelled mode is reached or a mode becomes infeasible with respect to nonrenewable resource constraints. Clearly, the following two observations hold.

Observation 1. *For $|N| = 1$, Init-MA always constructs a feasible mode assignment if one exists.*

Observation 2. *For the nonrenewable resource-unconstrained case, Init-MA assigns to each activity its mode with shortest duration.*

4.3. *The local search phase: improvement of the mode assignment*

We define the neighborhood of a feasible mode assignment M as follows: for exactly one activity $j, 1 < j < J$, we change the current mode assigned to j from $\mu(j)$ to $m, 1 \le m,\ \mu(j) \le M_j,\ m \ne \mu(j)$. If the new mode assignment is feasible, we stop with the new neighbor M'. Otherwise we carry on changing modes for not-yet selected activities. Thus, to each activity a new mode can be assigned only once. In this way we end up with two possible results: either we obtain a new feasible mode-assignment where to an improper subset of the activities a new mode has been assigned, or we cannot reach a feasible mode assignment.

The question is, now, which mode to assign to which activity in the first step. A quite straightforward way would be to choose the activity–mode pair that will provide the greatest decrease in the objective function. To do so, we would need to optimally solve the single-mode RCPSP for each available activity–mode pair. However, this is not feasible because even the fastest optimal procedures for the single-mode RCPSP can only solve instances with up to 30 activities [26]. Instead, we could derive for each activity–mode pair an upper bound of the objective function by solving the single-mode RCPSP heuristically. But still this is not very practical since we have to evaluate $\sum_{j=2}^{J-1}(M_j - 1)$ different activity–mode pairs and the complexity of the standard heuristic algorithm solving the single-mode RCPSP is $O(J^2 \log J)$ (see for example, [27]). To decrease the computational effort, we calculate an estimate of the effect of an activity–mode change on the objective function. We start with the last schedule generated and calculate for each activity $j,\ 1 < j < J$, a modified slack σ_j, which takes into account precedence as well as resource constraints. The algorithm was introduced in [28]. It runs in $O(J^2 \log J)$ time and is given in Appendix A. Analogously to (9) we then calculate for each activity–mode pair $[j,m]$ the duration difference by changing the current mode $\mu(j)$ into a new mode m:

$$\Delta d(j,m) = d_{jm} - d_{j\mu(j)}, \quad 1 \le m,\ \mu(j) \le M_j,\ m \ne \mu(j). \tag{10}$$

If we now relax constraints (3) we can give the following approximation of ΔT, the difference in the objective function caused by changing mode $\mu(j)$ by m:

$$\Delta T(j,m) = \begin{cases} \Delta d(j,m), & \text{if } \sigma_j = 0; \\ \max\{0, \Delta d(j,m) - \sigma_j\}, & \text{otherwise.} \end{cases} \tag{11}$$

Obviously, (11) gives only an estimate of the makespan reduction. Hence, it need not be beneficial in general to choose the activity–mode pair $[j,m]$ that minimizes (11). Instead, we have used the following mapping to obtain a selection probability for each choosable activity–mode pair. Let $JM1$ be, as defined below, the set of choosable activity–mode pairs. Further, let $\rho[j,m] = \max\{\Delta T(i,n) \mid (i,n) \in JM1\} - \Delta T(j,m)$ be the difference between the estimate of the largest makespan increase and the estimate of the makespan reduction when choosing $[j,m]$. The selection probability $\psi[j,m]$ for the activity–mode pair $[j,m]$ is calculated from

$$\psi[j,m] = \frac{(\rho[j,m]+1)^\alpha}{\sum_{[i,n]\in JM1}(\rho[i,n]+1)^\alpha}, \quad [j,m] \in JM1. \tag{12}$$

Equation (12) gives each activity–mode pair $[j,m]$ a selection probability $\psi[j,m] > 0$ favoring those pairs that are associated with a high estimate of the makespan reduction. The parameter α allows us to vary (12) between deterministically selecting the activity–mode pair with the highest estimate of the makespan reduction (α large) or performing a pure random selection ($\alpha = 0$). Preliminary computational results revealed a significant influence of α on the mean average deviation from a lower bound of the objective function value. Furthermore, $\alpha = 3$ gave the best results.

If the first mode switch results in an infeasible mode assignment, we have to decrease the resource consumption of at least one other activity to regain feasibility. Specifically, activities have to be chosen where a mode switch frees resources currently causing infeasibility and does not create negative left-over capacity with respect to the other nonrenewable resources. We refer to nonrenewable resources that are currently causing infeasibility as 'critical' resources, denoted by $N' = \{r \mid r \in N,\ \pi K_r < 0\}$. Using $\Delta k(j,m,r)$ as given in (9), we can calculate $\Delta k(j,m)$, the consumption difference with respect to critical resources when assigning mode m to activity j as

$$\Delta k(j,m) = \sum_{r\in N'} \Delta k(j,m,r),\ 1 \le m,\ \mu(j) \le M_j,\ m \ne \mu(j). \tag{13}$$

With the additional notation

AJ	the set of available activities,
$AM(j)$	the available modes of activity j,
$JM1$	the set of activity–mode pairs $[j,m]$ that can be chosen to improve the objective function value,
$JM2$	the set of activity–mode pairs $[j,m]$ that can be chosen to regain feasibility,

and the definitions

$$\begin{aligned} AM(j) &= \{1,\ldots,M_j\}\backslash\mu(j), \\ JM1 &= \{[j,m] \mid j \in AJ, m \in AM(j)\}, \\ JM2 &= \{[j,m] \in JM1 \mid \Delta k(j,m,r) \le 0\ \forall r \in N', \\ &\qquad \pi K_r - \Delta k(j,m,r) \ge 0\ \forall r \in N\backslash N'\}, \end{aligned}$$

the procedure Generate New Neighbor (Gen-New-N) can now be presented formally.

Procedure Gen-New-N:
Initialization: *Read* $M = (\mu(1), \ldots, \mu(J))$; $AJ = \{1, \ldots, J\}$; calculate $AM(j)$, $1 \leq j \leq J$; calculate $\pi K_r \; \forall r \in N$; $N' = \varnothing$; calculate $JM1, JM2$;
1. Update $\Delta T(j,m) \; \forall [j,m] \in JM1$;
   Select $[j^*, m^*] \in JM1$ with probability $\psi[j,m]$;
   $\mu(j^*) = m^*, AJ = AJ \backslash \{j^*\}$, update $\pi K_r \; \forall r \in N$,
   update $N', JM1, JM2$;
   If $N' = \varnothing$ then stop ;
2. If $JM2 = \varnothing$ then stop;
   Update $\Delta k(j,m) \; \forall [j,m] \in JM2$;
   Select $[j^*, m^*] \in JM2$ that minimizes $\Delta k(j,m)$;
   $\mu(j^*) = m^*; AJ = AJ \backslash \{j^*\}$; Update $\pi K_r \; \forall r \in N$;
   Update $N', JM2$;
   If $N' = \varnothing$ then stop else goto step 2.

Stop: if $N' = \varnothing$ then a new feasible mode-assignment M has been generated.

At the beginning of Gen-New-N every currently unassigned activity–mode pair is an element of the activity–mode set $JM1$. In step (1) we calculate for every activity–mode pair of $JM1$ the estimate of the makespan reduction according to (11) and we select the activity–mode pair $[j^*, m^*]$ on the basis of (12). We remove activity j^* from the set of available activities AJ and update the left-over capacities of the nonrenewable resources πK_r, the set of critical nonrenewable resources N', and the activity–mode sets $JM1$ and $JM2$. If the new mode assignment is feasible, Gen-New-N is finished; otherwise we proceed to the next step to regain feasibility.

Here we can choose activity–mode pairs out of the set $JM2$. These are all pairs for which activities have not yet been selected and a mode change fulfils the following two requirements with respect to nonrenewable resources: it does not increase the resource consumption with respect to critical resources and it does not make other resources critical. We select the activity–mode pair $[j^*, m^*]$ that in terms of (13) frees the maximum quantity of critical resource units. In the events of ties, the pair with the lowest activity (mode) number is selected. Activity j^* is removed from AJ and updating of the left-over capacities as well as the sets N', $JM1$, and $JM2$ takes place. Step 2 is continued until either feasibility is regained or the set of choosable activity–mode pairs $JM2$ is empty and hence Gen-New-N has to be finished with an infeasible mode assignment.

4.4. *The complete solution procedure*

In Sections 4.3 and 4.4 we explained the two crucial steps of our solution procedure (see Table 1): *Generate an initial mode assignment M* and *move from the feasible mode assignment M to a feasible neighbor M′*. Therefore, the only elements of our method that remain to be described are the fast and the near-optimal *heuristic for the RCPSP*, respectively. From the many good heuristics available we have chosen the one in [29]. Basically, the heuristic belongs to the sampling variety with the fundamental of selecting the search space taking into account the amount of available processing time and the characterization of the instance. We decided to employ this heuristic because by adjusting the number of iterations one can use the procedure both (see Table 1) in phase II as a fast heuristic and in phase III as a near-optimal heuristic. By applying the algorithm with one iteration in phase II, the procedure 'degenerates' to a simple parallel forward-pass heuristic with the so-called worst-case slack priority rule [29]. In phase III we apply the fully fledged heuristic of [29] to generate and evaluate 30 schedules for the best mode assignment M^*.

4.5. *Numerical example*

We now want to clarify the details of the outlined solution method by solving the example instance provided in Fig. 1 and Table 2. For the sake of clarity we select the activity–mode pair in step 2 of Gen-New-N deterministically, i.e., $[j^*, m^*] = \{[j,m] \mid \Delta T(j,m) = \min\{\Delta T(i,n) \mid [i,n] \in JM1\}\}$.

Table 3 reports step 1 of Init-MA. Each row corresponds to one iteration of step 1 where the values of the variables are given in the columns. Because for the example problem there is $|N| = 1$, the mode with the smallest resource consumption is assigned to each activity (see Observation 1). Hence, step 1 ends with the mode assignment $M = (1,2,1,2,1,2,1)$.

In step 2, the activities are ranked in the order $< 1,4,6,7,2,5,3]$. Table 4 reports the allocation of the remaining three nonrenewable capacity-units within step 3. Again, each row corresponds to one of the J iterations.

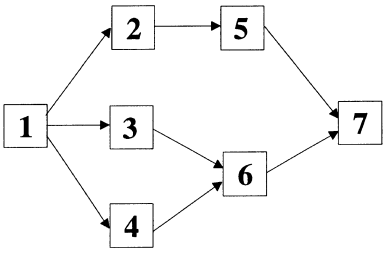

Fig. 1. Network of the example instance.

Table 2. Data for the example instance, where $R = \{1\}$, $N = \{2\}$, $K_1 = 4$, $K_2 = 8$

$j\ldots$	*1*	*2*		*3*	*4*		*5*	*6*		*7*
m	1	1	2	1	1	2	1	1	2	1
d_{jm}	0	4	6	2	3	5	2	2	4	0
k_{jm1}	0	2	1	1	3	1	1	2	1	0
k_{jm2}	0	3	1	0	4	2	0	3	2	0

Table 3. Deriving the initial mode assignment within step 1 of Init-MA

n	SJ	πK_2	RC_{jm} (SRC_j)										LRC	$j*$	$\mu(j^*)$
			$j\ldots$	1	2		3	4		5	6		7			
			$m\ldots$	1	1	2	1	1	2	1	1	2	1			
1	∅	8		(0)	0.37	(0.12)	(0)	0.5	(0.25)	(0)	0.37	(0.25)	(0)	0.25	4	2
2	4	6		(0)	0.5	(0.16)	(0)			(0)	0.5	(0.33)	(0)	0.33	6	2
3	4,6	4		(0)	0.75	(0.25)	(0)			(0)			(0)	0.25	2	2
4	2,4,6	3		(0)			(0)			(0)			(0)	0	1	1
5	1,2,4,6	3					(0)			(0)			(0)	0	3	1
6	1,...,4,6	3								(0)			(0)		5	1
7	1,...,6	3											(0)		7	1

The $\mu(j)$ in the fourth column represents the initial mode assigned to activity j; the $\mu(j)$ in the last column is the new mode assigned to activity j. An empty entry signifies that the inital mode has not been changed. The final mode assignment derived by Init-MA is $M = (1, 2, 1, 1, 1, 1, 1)$. Applying the RCPSP heuristic of [29] to solve the remaining RCPSP we obtain the schedule $S = (0, 6, 2, 5, 8, 7, 8)$. As can be seen in Fig. 2, S has a makespan of 8 periods.

Starting with the mode assignment as obtained by Init-MA, the initialization phase of Gen-New-N terminates with $AJ = \{1, \ldots, 7\}$, $\pi K_2 = 0$, $N' = \emptyset$, $AM(j)\ \forall j, 1 \leq j \leq J$, as given in Table 5, $JM1 = \{[2, 1], [4, 2], [6, 2]\}$, and $JM2 = \{[4, 2], [6, 2]\}$. Tables 6 and 7 report the mode assignments obtained by steps 1 and 2 of Gen-New-N, respectively. The ΔT values for the activity–mode pairs in Table 6 were obtained with (11) by using the precedence- and resource-based slack values as given in Appendix A. Now, step 1 of Gen-New-N assigns the first mode to activity 2. Because this leads to infeasibility, step 2 has to be executed. Herein, one iteration, switching from the first to the second mode of activity 4, is sufficient to regain feasibility. Hence, Gen-New-N stops after both step 1 and step 2 have been processed once. The mode assignment derived is $M = (1, 1, 1, 2, 1, 1, 1)$. Solving the remaining RCPSP with our near-optimal heuristic we obtain the (optimal) schedule $S = (0, 4, 2, 5, 6, 7, 7)$ with a makespan of 7 periods as depicted in Fig. 3.

Table 4. Allocation of the remaining capacity units within step 3 of Init-MA

n	πK_2	$j = \lambda(n)$	$\mu(j)$	m	$\Delta k(j, m, 2)$	$\mu(j) = m$
1	3	1	1			
2	3	4	2	1	−2	1
3	1	6	2	1	−1	1
4	0	7	1			
5	0	2	2	1	−2	
6	0	5	1			
7	0	3	1			

4.6. *Adaptations of the Method*

The proposed solution method can easily be adapted to different objective functions. When seeking for, e.g., the maximization of the net present value, one has to consider the following components of the general procedure as given in Table 1: the fast and the near-optimal problem-specific heuristics employed in phases I and III, respectively, have to be replaced by procedures for the objective function under consideration. Furthermore, one may alter the ranking of the activities within Init-MA as well as the presorting of modes. Finally, one has to employ a criterion different from ΔT for selecting activity–mode pairs within step 1 of Gen-New-N.

5. Computational investigation

The computational study presented in this section has been performed with two goals: first, to assess the solution quality of the outlined method, and secondly, to compare the method with other heuristics for the MRCPSP that have been published in the open literature. All procedures have been (re-)coded in PASCAL and implemented on an IBM-compatible personal computer with a 80386DX processor and 40 MHz clock pulse. Pseduo-random numbers were drawn by using the generator proposed by Schrage [30].

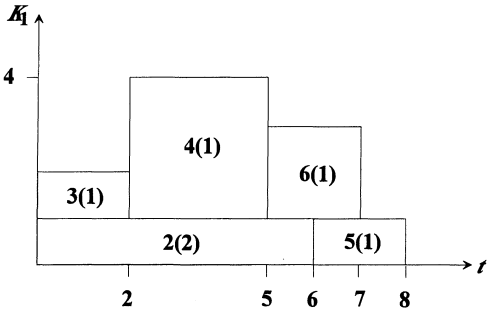

Fig. 2. Schedule associated with the solution of Init-MA.

Table 5. Available modes after the initialization phase of Gen-New-N

j	1	2	3	4	5	6	7
$AM(j)$	$\varnothing$	{1}	$\varnothing$	{2}	$\varnothing$	{2}	$\varnothing$

5.1. *Test instances*

As test instances we have employed two instance sets that were generated with the project scheduling instance generator ProGen [14]. ProGen is currently the only program that can systematically generate instances for scheduling problems with multiple execution modes and different resource categories [12,13]. Both benchmark sets are made of 640 instances that were generated by applying a full factorial experimental design with different problem parameters. The first set consists of 10 non-dummy-activity instances, and the second set of 30 non-dummy-activity instances. Both sets were generated considering three categories of problem parameters, namely, systematically varied parameters, constant parameters, and parameters drawn randomly out of a specified interval.

Systematically varied parameters are the resource factor RF_R, RF_N and the resource strength RS_R, RS_N for renewable and nonrenewable resources, respectively. The resource factor reflects the density of the coefficient matrix given in constraints (3) and (4). The two levels considered for each resource factor are 0.5 and 1. The resource strength measures the degree of resource-constrainedness in the interval [0,1]. Generally, RS is computed as follows: $RS = (K_r - K_r^{\min})/(K_r^{\max} - K_r^{\min})$, where $K_r^{\min}$ and $K_r^{\max}$ are a lower and an upper bound of the resource demand. For the 10-activity instances the following four levels were considered for each resource strength: 0.2, 0.5, 0.7, and 1. The 30-activity instances were generated with the levels 0.25, 0.5, 0.75, and 1.

Constant problem parameters are as follows: $M_j = 3$ for $1 < j < J$ and $M_j = 1$ for $j \in \{1, J\}$, $|R| = |N| = 2$, $|S_1| = |P_J| = 3$, and $NC = 1.5$, where S_1 denotes the set of successor activities of the dummy start activity $j = 1$ and NC stands for the network complexity, which is the number of nonredundant arcs divided by the number of activities.

Finally, Table 8 reports the parameters that were randomly chosen out of the specified interval. Q represents the number of resource types in which an activity–mode pair has a nonzero demand, whereas U denotes the number of units that an activity–mode pair requests of one resource type when having positive demand, i.e., when $k_{jmr} > 0$. More details about the benchmark instances can be found in [14,31].

Table 6. Report of step 1 of Gen-New-N

πK_2	N'	$JM1$	ΔT	$\Delta T^{\min}$	$[j^*, m^*]$
0	$\varnothing$	[2,1], [4,2], [6,2]	−2, 1, 1	−2	[2,1]

Table 7. Report of step 2 of Gen-New-N

πK_2	N'	$JM2$	Δk	$\Delta k^{\min}$	$[j^*, m^*]$
−2	{2}	[4,2], [6,2]	−2, −1	−2	[4,2]

With the use of a full factorial design study, for each level combination of the systematically varied problem parameters 10 instances were randomly generated within the limits given by the other parameter groups. Hence 640 instances emerged for each test set. In the 10-activity set, 536 instances had a feasible solution. Optimal objective function values for each of these instances are reported in [14]. The instances of the 30-activity set are not solvable by the state-of-the-art exact procedures; hence, it is not known how many of these instances have no feasible solution. To benchmark the heuristic solution procedures on the 30-activity instance set, we calculated precedence-based lower bounds on the makespan by fixing for each activity the mode with shortest duration and solving (1), (2) and (5) to optimality via longest path computation. In addition, we selected only the 550 instances for which we obtained a feasible solution by at least one of the three heuristics tested in Section 5.2.

5.2. *Computational results*

Table 9 shows the performance of our local search heuristic for different numbers of iterations within phase II. *DEV* denotes the percentage deviation from the optimal objective function value (precedence-based lower bound) for the instances. *CPU* denotes the computation time in CPU-seconds. Av., Stddev. and Max. denote the average,

Table 8. Intervals for randomly chosen parameter levels

	d_{jm}	Q_R	U_R	Q_N	U_N	S_j	P_j
Min.	1	1	1	1	1	1	1
Max.	10	2	10	2	10	3	3

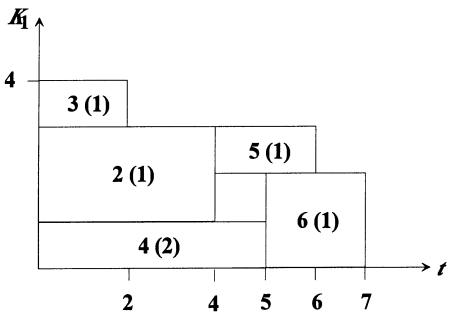

Fig. 3. Final schedule of the example problem.

Table 9. Effect of the number of iterations on the 10-activity instances

Iterations...		*10*	*50*	*100*	*500*
DEV	Av.	10.24	5.72	4.08	1.75
	Stddev.	13.65	9.38	7.68	4.87
	Max.	69.57	57.14	47.50	47.50
CPU	Av.	0.06	0.17	0.29	1.22
	Stddev.	0.04	0.05	0.07	0.31
	Max.	0.17	0.28	0.44	1.92

Table 10. Effect of the number of iterations on the 30-activity instances

Iterations...		*10*	*50*	*100*	*500*
DEV	Av.	27.92	24.46	23.18	21.01
	Stddev.	35.81	32.00	30.87	29.08
	Max.	200.00	176.00	168.00	156.00
CPU	Av.	0.19	0.82	1.61	7.89
	Stddev.	0.04	0.12	0.22	1.10
	Max.	0.33	1.54	2.91	13.79

the standard deviation, and the maximum values of *DEV* and *CPU*, respectively.

To benchmark our procedure further (with 100 iterations) we have solved the two instance sets with the truncated enumeration of [6] and the biased sampling procedure of [17]. We followed the recommendations given by the authors, i.e., for the sampling approach of [17] we set the probability mapping parameter to 2 and chose a sample size of 100. Additionally, the heuristic has been refined by calculating precedence- and resource-feasible start times for selected activities. Scheduling selected activities only at their precedence-feasible start times does not guarantee feasible solutions for the instances where renewable resources are constrained, i.e., where $RS_R < 1$ holds [3]. For the truncated branch-and-bound procedure of [6] we imposed a time limit of 10 CPU-seconds. Branches were selected with the LFT priority rule. We did not consider any of the other heuristics reviewed in Section 3.2 because they are not suited to deal with multiple nonrenewable resource constraints. At the end of this section, however, we give a brief comment on the results to be expected from some of these procedures.

In addition to the performance measures *DEV* and *CPU*, we denote by *SOL* the percentage of instances for which a feasible solution was derived. Table 11 gives the results of the 10-activity instances. Our procedure is the only heuristic that finds feasible solutions for all 536 instances. The capability of the other two heuristics to derive feasible solutions depends heavily on the two nonrenewable resource parameters RF_N and RS_N. When the average number of requested nonrenewable resources increases from $RF_N = 0.5$ to $RF_N = 1$ or when the scarcity of the nonrenewable resources is increased by lowering RS_N, the percentage of solved instances *SOL* decreases sharply. Both heuristics fail to solve any of the highly nonrenewable resource-constrained instances out of the class $RS_N = 0.2$. Even within the problem classes where the other heuristics find feasible solutions for the easier instances, e.g., $RS_N = 0.5$ or $RS_N = 0.7$, the quality of the solutions in terms of the average deviation from the objective function value is worse than that of the solutions found by our local search procedure on all instances (including the hard ones) of a class. Furthermore, our procedure is faster than the other heuristics. This is because only feasible mode assignments are visited and

Table 11. Comparison of heuristics on the 10-activity instances

		Number of instances	*Talbot [6]*			*Drexl and Grünewald [17]*			*This paper*		
			SOL	*DEV*	*CPU*	*SOL*	*DEV*	*CPU*	*SOL*	*DEV*	*CPU*
RF_R	0.5	232	58.96	14.88		76.83	9.97		100	3.76	
	1	304	57.04	16.89		83.03	12.46		100	4.37	
RS_R	0.2	76	56.30	25.18		67.23	5.88		100	7.98	
	0.5	153	60.43	19.40		84.17	14.40		100	4.54	
	0.7	156	59.42	13.12		83.33	14.91		100	2.36	
	1	151	55.00	6.98		83.57	7.04		100	1.98	
RF_N	0.5	259	87.07	14.82		97.41	8.42		100	2.24	
	1	277	35.53	17.93		66.78	13.99		100	5.48	
RS_N	0.2	119	0.00	—		0.00	—		100	9.60	
	0.5	139	32.68	22.74		86.27	10.02		100	4.91	
	0.7	138	69.87	22.12		95.51	11.03		100	3.08	
	1	140	100	9.16		98.01	6.56		100	1.48	
Av.			57.83	15.91	4.40	80.00	10.88	0.45	100	4.08	0.29
Stddev.				18.59	4.90		17.07	0.03		7.68	0.07
Max.				86.67	10.06		116.6	0.55		47.50	0.44

Table 12. Comparison of heuristics on the 30-activity instances

			Talbot [6]			*Drexl and Grünewald [17]*			*This paper*		
		Instances	*SOL*	*DEV*	*CPU*	*SOL*	*DEV*	*CPU*	*SOL*	*DEV*	*CPU*
RF_R	0.5	270	7.03	20.48	0.41	74.07	14.74	2.33	100.00	17.53	1.53
	1	280	6.78	20.43	0.33	71.42	29.38	2.34	100.00	28.63	1.68
RS_R	0.25	140	5.71	59.27	0.01	71.42	55.26	2.34	100.00	52.00	1.81
	0.5	140	8.57	14.38	0.00	71.42	19.45	2.34	100.00	18.73	1.57
	0.75	140	5.71	1.78	0.97	71.42	7.47	2.33	100.00	12.13	1.53
	1	130	7.69	11.63	0.63	76.92	6.06	2.34	100.00	8.85	1.49
RF_N	0.5	240	15.83	20.45	0.37	100	20.56	2.34	100.00	11.92	1.59
	1	310	0.00	—	—	51.61	24.30	2.33	100.00	31.90	1.62
RS_N	0.25	70	0.00	—	—	0.00	—	—	100.00	70.07	1.72
	0.5	160	13.75	27.30	0.64	50.00	26.75	2.28	100.00	25.85	1.63
	0.75	160	10.00	11.03	0.01	100.00	24.64	2.33	100.00	13.47	1.58
	1	160	0.00	—	—	100.00	17.14	2.37	100.00	9.72	1.56
Av.			6.90	20.45	0.37	72.72	22.06	2.34	100.00	23.18	1.61
Stddev.				28.07	1.58		26.52	0.15		30.87	0.22
Max.				108.0	7.80		121.0	2.69		168.0	2.91

hence no computational time is wasted by generating infeasible solutions.

The results on the 30-activity instances as given in Table 12 support the findings obtained on the 10-activity instance set. Our procedure generates feasible solutions for all 550 instances. In comparison, the truncated branch-and-bound procedure of [6] derives only 38 feasible solutions and the sampling approach of [17] yields 400 solved instances. Table 13 reports the performance of our procedure on these two subset. Clearly, it can be seen that our procedure produces lower *DEV* values than each of the other heuristics.

We can conclude that our solution method fulfils two important tasks: it derives feasible solutions, and the solutions are reasonably close to optimality. Because all other heuristics proposed so far for the MRCPSP do not explicitly take into account multiple scarce nonrenewable resources, we conjecture that they will fail to derive feasible solutions for such instances.

6. Conclusions

A very general project scheduling problem with renewable and nonrenewable resource types as well as activities with multiple execution modes has been considered. By transformation of the knapsack problem we showed that the feasibility problem is NP-complete. We proposed a

Table 13. Performance of the local search procedure on different subsets of instances

		Subset of 38 instances				*Subset of 400 instances*			
		Instances	*SOL*	*DEV*	*CPU*	*Instances*	*SOL*	*DEV*	*CPU*
RF_R	0.5	19	100.00	9.83	1.51	200	100.00	7.86	1.49
	1	19	100.00	11.25	1.64	200	100.00	16.93	1.67
RS_R	0.25	8	100.00	37.91	1.77	100	100.00	38.84	1.82
	0.5	12	100.00	7.02	1.65	100	100.00	6.84	1.56
	0.75	8	100.00	0.00	1.43	100	100.00	2.33	1.49
	1	10	100.00	1.35	1.45	100	100.00	2.07	1.45
RF_N	0.5	38	100.00	10.56	1.58	240	100.00	11.92	1.59
	1	0	100.00	—	—	160	100.00	13.10	1.57
RS_N	0.25	0	100.00	—	—	0	100.00	—	—
	0.5	22	100.00	12.77	1.62	80	100.00	15.60	1.63
	0.75	16	100.00	7.50	1.50	160	100.00	13.47	1.58
	1	0	100.00	—	—	160	100.00	9.72	1.56
Av.			100.00	10.56	1.58		100.00	12.40	1.59
Stddev.				17.98	0.21			19.60	0.24
Max.				78.95	2.25			100	2.91

local search method consisting of a construction phase that tries to reach an initial solution, a local search phase that performs a single neighborhood search on the set of feasible mode-assignments, and finally, an intensification phase where on the basis of the best mode assignment it tries to find a schedule with an improved objective function value. An in-depth computational investigation, which included a comparison with other heuristics, revealed that our local search approach is the only heuristic that solves all 10-activity instances and generated the most feasible solutions for the 30-activity instances. Futhermore, the performance in terms of the deviation from the optimal objective function value is consistently lowest.

Acknowledgement

We thank two anonymous referees for detailed comments that distinctly improved the presentation of the paper

References

[1] Kusiak, A. (1990) A knowledge- and optimization-based approach to scheduling in automated manufacturing systems, in, *Operations Research and Artificial Intelligence*: *The integration of Problem-solving strategies*, Brown, D.E. and White, C.C. (eds), Kluwer, Boston, pp. 453–479.

[2] Blazewicz, J. and Finke, G. (1994) Scheduling with resource management in manufacturing systems. *European Journal of Operational Research*, **76**, 1–14.

[3] Kolisch, R. (1995) Project scheduling under resource constraints – efficient heuristics for several problem classes. Ph.D. dissertation, Christian-Albrechts-University zu Kiel.

[4] Ahn, J. and Kusiak, A. (1991) Scheduling with alternative process plans, in *Modern Production Concepts*, Fandel, G. and Zäpfel, G. (eds), Springer, Berlin, pp. 386–403.

[5] Li, R.K.-Y. and Willis, J. (1991) Alternative resources in project scheduling. *Computers and Operations Research*, **18**, 663–668.

[6] Talbot, B. (1982) Resource-constrained project scheduling with time-resource tradeoffs: the nonpreemptive case. *Management Science*, **28**, 1197–1210.

[7] Slowinski, R. (1989) Multiobjective project scheduling under multiple-category resource constraints, in *Advances in Project Scheduling*, Slowinski, R. and Weglarz, J. (eds), Elsevier, Amsterdam, pp. 113–134.

[8] Blazewicz, J., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1983) Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, **5**, 11–24.

[9] Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability – A Guide to the Theory of NP-completeness*, Freeman, San Francisco.

[10] Patterson, J.H., Slowinski, R., Talbot, F.B. and Weglarz, J. (1989) An algorithm for a general class of precedence and resource constrained scheduling problems, in, *Advances in Project Scheduling*, Slowinksi, R. and Weglarz, J. (eds), Elsevier, Amsterdam, pp. 3–28.

[11] Speranza, M.G. and Vercellis, C. (1993) Hierarchical models for multi-project planning and scheduling. *European Journal of Operational Research*, **64**, 312–325.

[12] Sprecher, A. (1994) Resource-constrained project scheduling: exact methods for the multi-mode case. *Lecture Notes in Economics and Mathematical Systems*, Vol. 409, Springer, Berlin.

[13] Sprecher, A., Hartmann, S. and Drexl, A. (1997) An exact algorithm for project scheduling with multiple modes. *OR Sprektrum*, in the press.

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

[15] Hartmann, S. and Sprecher, A. (1996) A note on 'Hierarchical models for multi-project planning and scheduling'. *European Journal of Operational Research* **94**, 377–383.

[16] Demeulemeester, E. and Herroelen, W.S. (1992) A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, **38**, 1803–1818.

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

[18] Slowinski, R., Soniewicki, B. and Weglarz, J. (1994) DSS for multiobjective project scheduling. *European Journal of Operational Research*, **79**, 220–229.

[19] Özdamar, L. and Ulusoy, G. (1994) A local constraint based analysis approach to project scheduling under general resource constraints. *European Journal of Operational Research*, **79**, 287–298.

[20] Dell'Amico, M. (1990) Un algoritmo euristico per la pianificazione delle risorse nei progetti software, in *Proceedings of the Annual Conference of the Operational Research, Society of Italy*, pp. 211–226.

[21] Boctor, F.F. (1993) Heuristics for scheduling projects with resource restrictions and several resource-duration modes. *International Journal of Production Research*, **31**, (11), 2547–2558.

[22] Boctor, F.F. (1996) A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes. *European Journal of Operational Research*, **90**, 349–361.

[23] Boctor, F.F. (1997) An adaptation of the simulated annealing algorithm for solving resource-constrained project scheduling problems. *International Journal of Production Research*, in press.

[24] van Laarhoven, P.J.M., Aarts, E.H.L. and Lenstra, J.K. (1992) Job shop scheduling by simulated annealing. *Operations Research*, **40**, 113–125.

[25] Papadimitriou, C.H. and Steiglitz, K. (1982) *Combinatorial Optimization – Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.

[26] Demeulemeester, E. and Herroelen, W.S. (1997) New benchmark results for the resource-constrained project scheduling problem. *Management Science*, in the press.

[27] Leon, V.J. and Balakrishnan, R. (1995) Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling. *OR Spektrum*, **17**, 173–182.

[28] Wiest, J.D. (1967) A heuristic model for scheduling large projects with limited resources. *Management Science*, **13**, B359–B377.

[29] Kolisch, R. and Drexl, A. (1996) Adaptive search for solving hard project scheduling problems. *Naval Research Logistics*, **43**, 23–40.

[30] Schrage, L. (1979) A more portable Fortran random number generator. *ACM Transactions on Mathematical Software*, **5**, 132–138.

[31] Kolisch, R. and Sprecher, A. (1996) PSPLIB – A project scheduling problem library. *European Journal of Operational Research*, **36**, 205–216.

Appendix A. Calculation of precedence-and resource-based slack

Given a schedule S, the procedure 'slack calculation' (Sl-Ca) due to Wiest [28] calculates a precedence- and resource-based slack σ_j for each activity j:

Procedure Sl-Ca:
Initialization: Given a feasible schedule $S = (FT_1, \ldots, FT_J)$.
Sort activities according to non-increasing FT_j; with first tie-breaker min $d_{j\mu(j)}$, second tie-breaker min $\sum_{r \in R} k_{jr\mu(j)}$, and final tie-breaker min (activity number).
Store the activity order in the list $\lambda(j),\ 1 \leq j \leq J$;
For $n = 1$ TO J do:
  Begin:
    $j = \lambda(n)$;
    Locally right-shift activity j as far as possible;
    Assign to activity j the (new) finish time FT'_j;
    $\sigma_j = FT'_j - FT_j$;
  End.
Stop: the precedence- and resource-based slack $SL = (\sigma_1, \ldots, \sigma_J)$ have been calculated.

Applying Sl-Ca to the mode assignment $M = (1, 2, 1, 1, 1, 1, 1)$ and the schedule $S = (0, 6, 2, 5, 8, 7, 8)$ as obtained for the example instance by phase I, yields the results given in Tables A1 and A2.

Table A1. Priority list obtained by the initialization of Sl-Ca

$j \ldots$	*1*	*2*	*3*	*4*	*5*	*6*	*7*
FT_j	0	6	2	5	8	7	8
$d_{j\mu(j)}$	0	6	2	3	2	2	0
$\lambda(j)$	7	4	6	5	2	3	1

Table A2. Report of Sl-Ca

$n \ldots$	*1*	*2*	*3*	*4*	*5*	*6*	*7*
$j = \lambda(n)$	7	5	6	2	4	3	1
FT_j	8	8	7	6	5	2	0
FT'_j	8	8	8	6	6	3	0
σ_j	0	0	1	0	1	1	0

Biographies

Andreas Drexl is Full Professor of Production and Operations Management in the Department of Management Science at the University of Kiel. He received degrees in Management Science and Operations Research from University Augsburg and his doctoral degree from University Hamburg, He obtained his habilitation from Technical University Darmstadt. His main teaching and research interests are in quantitative/computer science approaches to production planning and scheduling. Recently he has published several articles in, among others, *European Journal of Operational Research*, *Journal of the Operational Research Society*, *Operations Research Spektrum*, and *Management Science*. He is a member of DGOR, IIE, and INFORMS.

Rainer Kolisch is Associate Professor of Production and Operations Management in the Department of Management Science at the University of Kiel. He received his Master's degree in Industrial Engineering from the Technical University of Darmstadt and his doctoral degree in Production and Operations Management from the University of Kiel. Dr. Kolisch's main teaching and research interests are in quantitative/computer science approaches to management science in general and production and operations management in particular. He has published articles in, among others, *European Journal of Operational Research*, *Journal of Operations Management*, *Management Science*, *Naval Research Logistics*, and *Production and Inventory Management*. He is a member of the German Operations Research Society, DGOR.