International Journal of Operations Research

International Journal of Operations Research Vol. 4, No. 1, 50–62 (2007)

Resource Leveling in Make-to-Order Production: Modeling and Heuristic Solution Method

Francisco Ballestín[1,*], Christoph Schwindt[2], and Jürgen Zimmermann[2]

[1]Department of Statistics and OR, Public University of Navarra, Campus Arrosadia, 31006 Pamplona, Spain

[2]Institute of Management and Economics, Clausthal University of Technology, Julius-Albert-Str. 2, 38678 Clausthal-Zellerfeld, Germany

Received July 2006; *Revised November 2006*; *Accepted November 2006*

Abstract—We consider the mid-term production planning problem in make-to-order production, which consists in scheduling production orders so that the variability in the resource utilization over time is minimized. The problem is modeled as a resource leveling project scheduling problem, where production orders for final products and main components correspond to activities of a project. We present a new population-based resource leveling procedure of the iterated greedy type. The results of an experimental performance analysis including projects with up to 1000 activities indicate that the new method outperforms state-of-the-art resource leveling heuristics from literature.

Keywords—Make-to-order production, Resource leveling, Project scheduling, Stochastic local search, iterated greedy heuristic

1. INTRODUCTION

In make-to-order production, which is typical of single-item or small-batch production, all products are manufactured on the basis of a given set of customer orders with individual delivery dates. Due to increasing customer standards, make-to-order production becomes more and more important in practice. In this paper we consider the mid-term level of production planning, i.e., the determination of the master production schedule (MPS). The MPS specifies the exact amounts of all final products and some main components (together called main products) and the corresponding times of production. Since in make-to-order production the production quantities of the final products are given by the customer orders, the required amounts of all main products belonging to a customer order can be easily computed by performing a bill of materials explosion. The production orders for a main product arise from combining those requirements into a lot for which a joint production seems advantageous due to similar delivery dates of the customer orders and savings in machine setup or materials handling costs. Hence, mid-term production planning in make-to-order production essentially boils down to the timing of the production orders under consideration.

Practical experience shows that leveling over time the resource requirements of the resources involved at the MPS level is crucial to providing good feasible solutions at the subsequent planning levels. Fluctuations in the utilization of resources like manpower or machinery are undesirable because they often cause extra labor or financial expenditure (see Easa (1989) and Tavares (1987)). In make-to-order production, where products are only produced on demand and hence the workload distribution over time cannot be balanced via inventory holding, resource leveling amounts to scheduling the production orders in such a way that a smooth resource utilization is achieved. The first resource leveling procedures have been developed for smoothing manpower requirements in make-to-order environments like naval shipyards (cf. Levy et al. (1962)) or aircraft construction plants (cf. Dewitte (1964)). The particular importance of resource leveling in make-to-order production is due to the non-repetitive character of the production orders, which makes it impossible to avoid bottlenecks by appropriately configuring the capacities of the production system. The importance of resource leveling in practice—not only in the context of make-to-order production—also becomes apparent by examining a number of widely used software tools for project management (e.g., Primavera or SuperProject), which offer simple shifting heuristics or activity splitting techniques for solving resource leveling problems (cf. Meredith and Mantel (2002) and Son and Mattila (2004)).

In make-to-stock production, the mid-term production planning problem is usually formulated as a linear program. The formulations are based on the decomposition of the planning horizon into individual time periods and contain decision variables for the production quantities, the subcontracted amounts, and the workforce levels in the periods of the planning horizon. Generally, it is assumed

* Corresponding author's email: francisco.ballestin@unavarra.es

that each production order can be completed within one period. Consequently, resource requirements are modeled on the aggregate level of period requirements without considering the sequencing of production orders within one period (see, e.g., Nahmias (2001), Sect. 3.5 for a basic version of such an aggregate planning model). Since in make-to-order production the production quantities are typically small and the lead times of production orders may vary considerably, we require a more detailed view on production planning, which enables us to explicitly take sequencing decisions into consideration. In this paper we model the problem as a resource leveling project scheduling problem where each production order for a main product is regarded as an activity of a customer project (cf. Franck et al. (1997) and Neumann and Schwindt (1998)). Integer programming formulations for the scheduling of production orders in make-to-order environments that take the resource leveling criterion into account can be found in Özdamar and Yazgaç (1997) and Sawik (2006).

Due to prescribed delivery dates as well as technical or organizational temporal constraints arising from the process plans, there are time restrictions for some activities. Thus, each customer project generally contains a number of prescribed minimum and maximum time lags in between the start times of certain activities. Given a maximum project duration, the problem consists in finding an activity schedule which observes the prescribed time lags and minimizes the variability in the resource requirements.

For what follows we assume that each production order must be carried out without interruption. Preemptive schedules often incur additional costs for machine setup, materials handling, and inventory holding or impair the quality of the products (see, e.g., Domschke and Drexl (1991) or Braun and Schmidt (2003)). On principle, the case where interruptions are allowed can be integrated into our model. Provided that all processing times and prescribed time lags are integer-valued, we may decompose a production order into a sequence of unit-time activities. Each activity corresponds to a unit-time portion of the total processing. By introducing a precedence constraint between any two subsequent activities in the sequence we ensure that first, all portions of the processing are executed one after another and second, the production order can be interrupted at each integral point in time. This approach has been used by Demeulemeester and Herroelen (1996) for solving the preemptive resource-constrained project scheduling problem. Note, however, that the decomposition into unit-time activities leads to a model whose size is pseudo-polynomial in the input length of the problem.

As has been shown in Neumann et al. (2003, Sect. 3.4) the resource leveling project scheduling problem is NP-hard even if only one resource is considered. Since in practice we commonly need to deal with a large number of customer orders simultaneously when determining an MPS, each of which consists of several main products, the resulting projects often contain several hundreds of activities. Therefore, in this paper, we devise a heuristic solution method for problem instances of up to 1000 activities.

Exact algorithms based upon implicit enumeration, integer programming, or dynamic programming techniques, which are applicable to instances with up to 20 activities, have been proposed for the case of precedence constraints between activities, e.g., by Ahuja (1976, Ch. 11), Easa (1989), Bandelloni et al. (1994), and Younis and Saad (1996). For general minimum and maximum time lags Neumann and Zimmermann (1999a) and Neumann et al. (2003, Sect. 3.6) present a time-window based approach and a tree-based branch-and-bound procedure, respectively.

Heuristic procedures for the case of precedence constraints are given, e.g., in Burgess and Killebrew (1962), Harris (1978, Ch. 11), Harris (1990), Moder et al. (1983, Ch. 7), and Takamoto et al. (1995). These algorithms are simple shifting heuristics or priority-rule based methods which can be easily adapted to consider general time lags. Some newer modifications of the so-called pack heuristic of Harris (1990) can be found in Hiyassat (2000, 2001), and a genetic algorithm is described in Leu et al. (2000). However, for none of all those heuristics an in-depth performance analysis has been published. For resource leveling problems with different objective functions and general time lags priority-rule based methods as well as a tabu search procedure are discussed in Brinkmann and Neumann (1996), Neumann and Zimmermann (1999b, 2000), and Neumann et al. (2003, Sect. 3.7). For the tabu search algorithm and the multi-start priority-rule based method devised by Neumann and Zimmermann (2000), the data of extensive computational experiments are available. We use these results as a benchmark in the performance analysis of our method.

The remainder of this paper is organized as follows. In Section 2 we develop the project scheduling model for production planning in make-to-order production and briefly summarize some structural issues of the scheduling problem. Section 3 is devoted to the new iterated greedy heuristic. At first, a generic iterated greedy algorithm iterating cycles of so-called destruction and construction phases as well as two postprocessing methods are devised. Then, a population-based extension and further destruction-construction operators are presented. In Section 4 we discuss the results of a comprehensive computational study comparing the heuristic to recent resource leveling algorithms from literature.

2. MODELING AND ANALYSIS OF THE PROBLEM

In this section we at first explain how to construct the project network belonging to a given set of customer orders. Based on this project network, we then provide a formal statement of the resource leveling project scheduling problem. Finally, we discuss some properties of the feasible region and characterize candidates for an optimal solution to the problem.

2.1 Construction of the project network

In make-to-order production, each customer order consists of a set of final products, corresponding order quantities, and a common delivery date. From the order quantities and the product structure, we compute the requirements for the main components by a bill of materials explosion and determine appropriate production orders for the manufacture or assembly of all main products involved. We consider each production order as a project activity *i*. The processing time p_i of activity *i* includes the setup time and the total processing time for the entire lot of the main product and its components at lower levels of the product structure. Due to waiting times, which are known only after shop floor scheduling has taken place, p_i is generally obtained by adding a surcharge to the sum of the setup and processing times. Practical experience shows that the lead times increase heavily with growing utilization. Elementary results from queueing theory can be used for estimating a surcharge that accounts for the dependency of lead times on the utilization of resources (see Karmarkar (1987) or Schneeweiß and Söhner (1995)). For what follows we assume that $p_i \in \mathbb{Z}_{\geq 0}$ for all activities *i*.

The execution of each individual customer order can be viewed as a project, which is in turn represented by an activity-on-node network. Accordingly, each activity *i* is associated with a node *i*, and the arcs of the network correspond to minimum and maximum time lags between the activities. A minimum time lag $d_{ij}^{\min} \in \mathbb{Z}_{\geq 0}$ between the start time S_i of activity *i* and the start time S_j of activity *j* means that activity *j* cannot be begun earlier than $d_{ij}^{\min}$ time units after the start of activity *i*, i.e., $S_j - S_i \geq d_{ij}^{\min}$. Such minimum time lags arise from assembly relationships between main products at lower and upper levels of the product structure. Neumann and Schwindt (1997) describe how to compute those time lags for the general case where the execution of production orders for products on different levels in the product structure may overlap in time. Sometimes, also maximum time lags between the start times of activities are given. A maximum time lag $d_{ij}^{\max} \in \mathbb{Z}_{\geq 0}$ between the start of two different activities *i* and *j* requires that $S_j - S_i \leq d_{ij}^{\max}$, i.e., activity *j* must be started $d_{ij}^{\max}$ time units after the start of activity *i* at the latest. Maximum time lags may, for example, be used to model upper bounds on the total flow times of products, which can serve to limit the work-in-progress inventory (see Neumann and Schwindt (1995) for further examples of minimum and maximum time lags arising in production scheduling). In a customer project network, a minimum time lag $d_{ij}^{\min}$ is modeled by an arc (i, j) with weight $\delta_{ij} := d_{ij}^{\min}$ and a maximum time lag $d_{ij}^{\max}$ is represented by a backward arc (j, i) with weight $\delta_{ji} := -d_{ij}^{\max}$. Note that a maximum time lag $d_{ij}^{\max}$ between activities *i* and *j* can be regarded as a negative minimum time lag $d_{ji}^{\min} = -d_{ij}^{\max} = -(-\delta_{ji}) = \delta_{ji}$ between activities *j* and *i*.

The project network for the entire set of customer orders is obtained by superposing all individual customer project networks. To this end, we introduce a supersource α and a supersink ω representing the beginning and the completion of the overall project. Furthermore, we add an arc with weight zero from node α to the sources of each customer project as well as an arc from each sink of a customer project to node ω whose weight equals the processing time of the sink. In this way we ensure that no activity is started before the project beginning and that all activities are completed by the end of the project. A delivery date $\overline{d}_i$ for some final product can be interpreted as a maximum time lag of $\overline{d}_i - p_i$ between the project beginning and the start of the activity *i* corresponding to production order for that product. In the project network, this maximum time lag gives rise to an arc from node *i* to α with weight $p_i - \overline{d}_i$. If a production order includes the production quantities of several customer orders with different delivery dates, the maximum time lag is determined by the earliest delivery date. Finally, we insert an arc from node ω to node α with weight $-\overline{d}$, which ensures that the maximum project duration $\overline{d}$ is not exceeded. This deadline $\overline{d}$ for the completion of the project can be chosen to be equal to the length of the planning horizon or equal to the largest delivery date. The construction of the project network is illustrated in the following example.

Example. We consider the three customer orders for final products *I*, *II*, and *III* given in Table 1. The main components of the final products are the subassemblies *A* and *B* as well as the parts *a* and *b*. The product structures of the final products and the corresponding processing and setup times are shown in Figure 1. We suppose that due to high setup costs, the total requirements for parts *a* and *b* and for subassembly *B* are each combined into one lot, whereas the units of *A* required for products *I* and *II* are processed in two separate lots. Moreover, we assume that the production orders for part *b* and subassembly *B* may overlap in time, the transfer batch size being equal to two units of *b*. This means that whenever two units of *b* have been completed, those units are released to the production order for *B*. For product *III* we limit the total flow time to 110% of the sum of the processing times. The deadline for the completion of all three customer projects is equal to the largest delivery date, i.e., $\overline{d} = 55$.

Table 1. Customer orders

Customer	Final product	Order quantity	Delivery date
X	*I*	2	30
Y	*II*	3	55
Z	*III*	1	50

Main product	Processing time per unit	Setup time
I	1	2
II	3	1
III	2	1
A	2	2
B	1	1
a	2	2
b	3	1

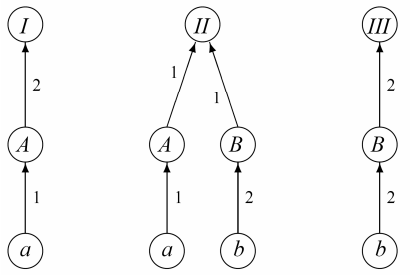

Figure 1. Processing and setup times as well as product structures for final products and main components.

Figure 2 shows the project network N belonging to the three customer orders. The durations of the project activities include the setup and processing times of the corresponding lots. For example, since we need seven units of part *a* to meet the demands for final products *I* and *II*, the activity corresponding to the processing of *a* has a duration of 16 = 2 + 7 · 2 time units. Since the assembly of *A* cannot start before all units of *a* have been manufactured, we have minimum time lags of 16 time units between the production order for *a* and the two production orders for *A*. The production orders for *b* and *B* may overlap in time. The minimum time lag between those production orders can be calculated as follows. Given that the processing time of one unit of *b* is greater than the processing time per unit of *B*, we have to synchronize the transfer of the last batch to ensure that all units of *B* can be processed without interruption. Accordingly, the minimum time lag is equal to the total processing time of the production order for *b* (1 + 10 · 3 = 31 time units) minus the time needed to assemble the (10 − 2)/2 = 4 units of *B* before the arrival of the last transfer batch (1 + 4 · 1 = 5 time units). The maximum flow time for product *III* equals 110% of the processing times of the production orders for *III* and the main components *B* and *b* at lower levels, i.e., 1.1 · (31 + 6 + 3) = 44 time units. By subtracting the processing time of the production order for *III* we obtain the corresponding maximum time lag of 41 time units between the starts of the production orders for *b* and *III*.

2.2 The resource leveling project scheduling model

Suppose that the project consists of $n + 2$ activities $i = 0, 1, ..., n + 1$ where the fictitious activities 0 and $n + 1$ correspond to supersource α and supersink ω of project network N. Then $V := \{0, 1, ..., n + 1\}$ is the set of all activities. Furthermore, let E be the set of all arcs of N. If we establish the convention that $S_0 := 0$, which means that the project always begins at time zero, S_{n+1} coincides with the project duration. A sequence $S = (S_0, S_1, ..., S_{n+1})$ with $S_i \geq 0$ $(i \in V)$ and $S_0 = 0$ is called a schedule. A schedule S satisfying the temporal constraints

$$S_j - S_i \geq \delta_{ij} \quad ((i, j) \in E)$$

is called *feasible*. The set of all feasible schedules is denoted by $\mathcal{S}$.

Assume that a set $\mathcal{R}$ of renewable resources (e.g., machines, manpower, or equipment) are required for carrying out the activities of the project and let $r_{ik} \in \mathbb{Z}_{\geq 0}$ be the amount of resource k used by activity i. Given a schedule $S = (S_i)_{i \in V}$,

$$r_k(S,t) := \sum_{\substack{i \in V: \\ S_i \leq t < S_i + p_i}} r_{ik}$$

is the amount of resource $k \in \mathcal{R}$ used at point in time $t \in [0, \overline{d}]$. The definition of $r_k(S, t)$ implies that each activity $i \in V$ is executed within the interval $[S_i, S_i + p_i[$ of length p_i without interruption. The function $r_k(S, \cdot) : [0, \overline{d}] \rightarrow \mathbb{Z}_{\geq 0}$ is referred to as the *resource profile* of resource k.

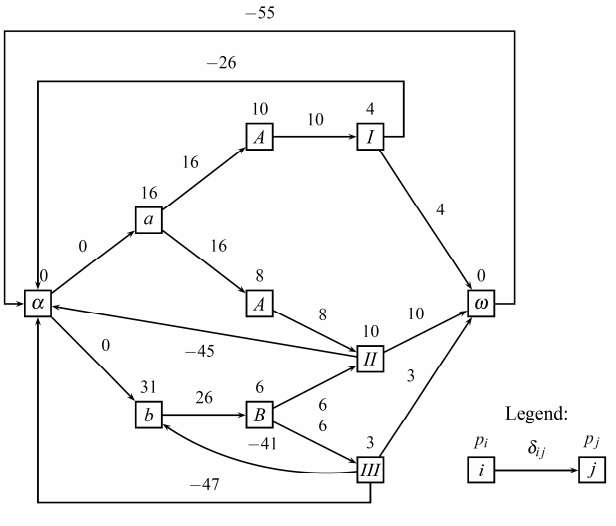

Figure 2. Project network N.

To obtain a balanced utilization of the renewable resources, we have to minimize some measure of the variability in profiles $r_k(S, \cdot)$. Let $c_k \geq 0$ be an adjustment cost incurred per unit of resource $k \in \mathcal{R}$. Then the leveling can, for example, be achieved by considering the total squared utilization cost of schedule S

$$f(S) = \sum_{k \in \mathcal{R}} c_k \int_0^{\overline{d}} r_k^2(S,t)\,dt \qquad (1)$$

Since the total workload $\int_0^{\overline{d}} r_k(S,t)\,dt = \sum_{i \in V} r_{ik}$ of resource k does not depend on schedule S, $f(S)$ equals a weighted sum of the variances of the resource profiles $r_k(S, t)$ plus a constant (cf. Schwindt (2005), Sect. 2.3).

The resource leveling problem can now be formulated as follows:

$$\left.\begin{array}{ll} \text{Minimize } f(S) & \\ \text{subject to } S_j - S_i \geq \delta_{ij} & ((i,j) \in E) \\ S_i \geq 0 & (i \in V) \\ S_0 = 0 & \end{array}\right\} (RLP)$$

A feasible schedule S that minimizes objective function f on the feasible region $\mathcal{S}$ of (RLP) is called optimal.

2.3 Structural issues

It is well-known that set $\mathcal{S}$ is nonempty if and only if network N does not contain any cycle of positive length. As the latter condition can be verified efficiently by applying a label correcting algorithm (see, e.g., Ahuja et al. (1993), Ch. 5), we will assume from now on that $\mathcal{S} \neq \emptyset$. In this case the length of a longest path from node i to node j in N corresponds to the (possibly negative) minimum time lag d_{ij} between activities i and j that is implied by the original minimum and maximum time lags. Due to the absence of positive-length cycles in N, the longest path lengths d_{ij} are equal to the longest walk lengths in N, which are provided by the label correcting algorithm establishing $\mathcal{S} \neq \emptyset$. Set $\mathcal{S}$ possesses exactly one minimal and exactly one maximal point, which coincide with the vectors $ES := (d_{0i})_{i \in V}$ and $LS := (-d_{i0})_{i \in V}$ called *the earliest* and *latest schedules*, respectively. ES_i and LS_i are the earliest and latest start times of activity i given the prescribed minimum and maximum time lags. $TF_i := LS_i - ES_i$ is termed the *total float* or *slack time* of activity i and corresponds to the maximum amount of time by which the start of i can be delayed beyond its earliest start time such that the deadline $\overline{d}$ for the completion of the project is met. If $TF_i = 0$, activity i is called *critical*, and for $0 < TF_i < p_i$ we speak of a *near-critical activity*. For critical and near-critical activities it holds that the time window $[LS_i, ES_i + p_i[$ during which activity i will necessarily be in progress is nonempty.

It is easily seen that the objective function f is a piecewise linear, continuous, and non-convex function. Since set $\mathcal{S}$ is compact, there consequently always exists an optimal schedule. Moreover, as has been shown by Neumann et al. (2000) the set of optimal schedules always contains a *quasistable schedule*. A feasible schedule S is called quasistable if and only if for each activity $j \in V$, $j \neq 0$ there is an activity $i \in V$ such that one of the following four conditions is met:

(a) $S_j = S_i + \delta_{ij}$, i.e., temporal constraint $S_j - S_i \geq \delta_{ij}$ is binding,
(b) $S_j = S_i - \delta_{ji}$, i.e., temporal constraint $S_i - S_j \geq \delta_{ji}$ is binding,
(c) $S_j = S_i + p_i$, i.e., precedence relationship $S_j - S_i \geq p_i$ is binding,
(d) $S_j = S_i - p_j$, i.e., precedence relationship $S_i - S_j \geq p_j$ is binding.

Note that in contrast to the temporal constraints the precedence relationships are not given in advance but induced by the schedule S under consideration. If conditions (a) or (b) are satisfied, activity j is begun at its earliest or latest start time given the start times of the remaining activities. Conditions (c) and (d) say that activity j is started immediately after or before the execution of activity i. It can be shown that schedule S is quasistable precisely if the equation system defined by $S_0 = 0$ and equations (a) to (d) satisfied by S possesses a unique solution, which coincides with schedule S. In this case we may associate an arc (i, j) or (j, i) with each of those conditions (a) to (d) and select a subset E' of these arcs such that they form a spanning tree $T = (V, E')$ with node set V (see Neumann et al. (2003), Proposition 3.2.16). From the latter statement it follows that each quasistable schedule is integer-valued and that the set of all quasistable schedules is finite. The NP-hardness of problem (RLP) implies that the number of quasistable schedules is exponential in the number n of activities. An exact brute-force algorithm for solving (RLP) consists in generating all quasistable schedules via the enumeration of all corresponding spanning trees.

Further objective functions for which the set of quasistable schedules contains an optimal solution are discussed in Demeulemeester and Herroelen (2002, Subsect. 3.1.3), Neumann et al. (2003, Sect. 3.1), and Schwindt (2005, Sect. 2.3). We briefly recall two of those objective functions, which may substitute function (1) in the statement of problem (RLP). When exceeding a given threshold $\overline{r}_k$ for the utilization of resource k incurs a cost of $c_k \geq 0$ per unit time and resource, we may use the leveling objective function

$$f(S) = \sum_{k \in \mathcal{R}} c_k \int_0^{\overline{d}} \max(0,\ [r_k(S,t) - \overline{r}_k])\,dt \qquad (2)$$

If no threshold is given, $\overline{r}_k$ can be replaced by the average resource utilization $1/d \sum_{i \in V} r_{ik} p_i$. Let $t_0, t_1, \ldots, t_\nu$ be the increasing sequence of different start and completion times of activities $i \in V$ given schedule S. Then any jump discontinuity in profiles $r_k(S, \cdot)$ occurs at time t_μ with $0 \leq \mu \leq \nu$, and the weighted sum of the total variations of the profiles $r_k(S, \cdot)$ is

$$f(S) = \sum_{k \in \mathcal{R}} c_k \sum_{\mu=1}^{\nu} \left| r_k(S,t_\mu) - r_k(S,t_{\mu-1}) \right| \qquad (3)$$

where $c_k \geq 0$ is the cost arising from increasing or decreasing the availability of resource $k \in \mathcal{R}$ by one unit. The heuristic procedure developed in Section 3 can also be used for solving problem instances with objective function (2) or (3).

3. THE ITERATED GREEDY METHOD

In this section we develop a population-based iterated greedy method for approximately solving resource leveling problem (RLP). Iterated greedy is a metaheuristic belonging to the class of stochastic local search methods. Stochastic local search can be interpreted as a biased random walk in a structured subset $(\mathcal{S}', \mathcal{N})$ of the feasible region $\mathcal{S}$. $\mathcal{N}: \mathcal{S}' \to 2^{\mathcal{S}'}$ is the neighborhood function that defines for each solution $S \in \mathcal{S}'$ a set $\mathcal{N}(S)$ of solutions $S' \in \mathcal{S}'$ being in some sense near to S (see Aarts and Lenstra (2003)). $\mathcal{N}(S)$ is called the neighborhood of S. Starting with an initial solution, stochastic local search iteratively moves from solutions S to neighboring solutions $S' \in \mathcal{N}(S)$ by applying a randomized operator $\mathcal{M}: \mathcal{S}' \to \mathcal{S}'$. Iterated greedy implements $\mathcal{M}$ as the pair of a destruction and a construction phase, where it is assumed that a solution S can be represented as a sequence of solution components S_i. Algorithm 1 shows a generic version of the iterated greedy method. During the destruction phase, some randomly selected components S_i are deleted from S. The construction phase then rebuilds a complete candidate solution S' with a stochastic greedy heuristic. Based on an appropriate acceptance criterion it is decided whether S is replaced by S' or the search is continued from S.

Algorithm 1. *Generic iterated greedy algorithm*

```
generate an initial solution S;
repeat
   reduce S to a partial solution S_; (*destruction phase *)
   expand S_ to a complete candidate solution S';
   (* construction phase *)
   if S' is accepted then put S := S';
until stopping condition is met;
return solution S;
```

Iterated greedy is closely related to two alternative local search schemes that have been proposed in literature. *Iterated local search* substitutes the destruction for a perturbation and the construction for a local search phase. The perturbation phase randomly biases some solution components S_i, whereas the local search phase re-optimizes the resulting perturbed solution (see, e.g., Lourenço et al. (2002) or Hoos and Stützle (2004), Sect. 2.3). In the *large neighborhood search* scheme introduced by Shaw (1997), the destruction phase corresponds to the selection of variables S_i whose values are unfrozen while the remaining variables remain fixed. The construction phase is then replaced with a generally exhaustive search for an optimal solution in the neighborhood defined by the relaxed variables. Hence, the iterated greedy method may be interpreted as a special implementation of the large neighborhood search scheme where the search in the neighborhood consists in randomly selecting some neighbor.

Iterated greedy algorithms have been applied with success, for example, to the set covering problem by Jacobs and Brusco (1995) and Marchiori and Steenbeek (2000). Recent implementations of iterated greedy in the field of scheduling can be found in the papers by Ruiz and Stützle (2005, 2007). Large neighborhood search methods for resource-constrained project and job-shop scheduling problems have been proposed by Palpant et al. (2004) and Godard et al. (2005), respectively. In the former paper, the variables S_i correspond to the start times of the project activities. The freed activities have to be re-scheduled within the time windows defined by the fixed start times of the frozen activities. Our iterated greedy method for the resource leveling problem is based on the same basic principle. In order to enlarge the time windows of the freed activities and hence increase the space for reoptimization, Godard et al. (2005) represent a schedule S by an expanded network $N(S)$ containing arcs for the prescribed time lags as well as for a set of precedence relationships induced by schedule S. The set of precedence relationships is chosen in such a way that any time-feasible schedule with respect to network $N(S)$ is also resource-feasible. We note that this approach offers an increased flexibility but is only applicable to scheduling problems with resource constraints.

In Subsection 3.1 we describe how to configure the building blocks of Algorithm 1 to obtain a basic iterated greedy method for the resource leveling problem. The following subsections treat different enhancements to the basic method. In Subsection 3.2 we develop two postprocessing methods, which are called in each iteration of the iterated greedy method to locally improve the rebuilt complete schedule. In Subsection 3.3 we explain how the method can be expanded to a population-based algorithm where in each iteration one schedule of the population is transformed into a new schedule by performing the destruction and construction phases. In Subsection 3.4 two variants of the standard destruction-construction operator are presented.

3.1 Basic method

When dealing with problem (RLP), a solution corresponds to a schedule $S = (S_i)_{i \in V}$ of start times S_i for the activities i of the project. A partial solution $\underline{S}$ is given by a partial schedule $S^C := (S_i)_{i \in C}$ where only a subset of activities $C \subseteq V$ has been scheduled, C being termed the completed set. We associate a partial schedule with resource profiles $r_k(S^C, \cdot\,)$ for the renewable resources $k \in \mathcal{R}$ in the following way. $r_k(S^C, t)$ takes into account the requirements of all scheduled activities $i \in C$ as well as the unavoidable resource requirements by the critical and near-critical activities $i \in V \setminus C$ that have not been scheduled so far, i.e.,

$$r_k(S^C,t) := \sum_{\substack{i\in C:\\ S_i\le t<S_i+p_i}} r_{ik} + \sum_{\substack{i\in V\setminus C:\\ LS_i\le t<ES_i+p_i}} r_{ik}$$

Roughly speaking, the destruction phase consists in choosing a set U and unscheduling all activities $i \in U$, that is, deleting the start times S_i of all activities $i \in U$ in schedule S. During the construction phase, we reschedule the activities $i \in U$ by determining new start times S_i' and adding them to partial schedule S^C with $C = V\setminus U$.

Before we present the details of the destruction and construction phases, we explain the way in which we construct an initial solution. We use the priority-rule based method by Neumann and Zimmermann (1999b) outlined in Algorithm 2. After the computation of the earliest and latest schedules ES and LS, in each iteration an activity i with minimum total float is selected to be scheduled next, ties being broken on the basis of the greatest resource demand $p_i\sum_{k\in\mathcal{R}} r_{ik}$ (priority rule MST-GRD, where MST stands for minimum slack time first, see Neumann et al. (2003), Sect. 3.7). Then, the so-called *decision set*

$$\mathcal{D}_i(S^C) := \{ES_i, LS_i\} \cup (\{S_j + p_j,\ S_j - p_i \mid j \in C\} \cap [ES_i, LS_i])$$

of tentative start times t for i is determined. The conditions on times t ensure that the resulting schedule S is quasistable. Next, a time $t \in \mathcal{D}_i(S^C)$ minimizing the additional cost

$$\begin{aligned} f_i(S^C,t) &:= \sum_{k\in\mathcal{R}} c_k \int_t^{t+p_i} (r_k(S^C,\tau)+r_{ik})^2\,d\tau \\ &\quad - \sum_{k\in\mathcal{R}} c_k \int_t^{t+p_i} r_k^2(S^C,\tau)\,d\tau \\ &= 2\sum_{k\in\mathcal{R}} c_k r_{ik} \int_t^{t+p_i} r_k(S^C,\tau)\,d\tau + const \end{aligned}$$

incurred by scheduling activity i is selected to be the start time S_i of i, where $const = p_i \sum_{k\in\mathcal{R}} c_k r_{ik}^2$ denotes a constant independent of time t. As has been shown by Neumann and Zimmermann (1999b), set $\mathcal{D}_i(S^C)$ always contains a minimizer of additional-cost function $f_i(S^C, \cdot)$. Finally, the partial schedule is expanded by S_i and the earliest and latest start times of the activities not yet scheduled are updated. These steps are iterated until all activities have been added to the schedule.

Algorithm 2. *Priority-rule based method*

calculate earliest and latest schedules ES and LS;
initialize completed set $C := \{0\}$ and put $S_0 := 0$;
while $C \neq V$ **do** (* not all activities $i \in V$ have been scheduled *)
  select an activity $i \in V\setminus C$ with minimum total float TF_i;
  compute decision set $\mathcal{D}_i(S^C)$;
  select a time $t \in \mathcal{D}_i(S^C)$ with minimum additional cost $f_i(S^C, t)$, add i to C, and put $S_i := t$; (* schedule i at time t *)
  for all $j \in V\setminus C$ **do** (* update earliest and latest start times *)
    put $ES_j := \max(ES_j, S_i + d_{ij})$ and
      $LS_j := \min(LS_j, S_i - d_{ji})$;
  end for;
end while;
return S;

After the computation of the initial solution, the iterated greedy method performs a stochastic local search in the set $\mathcal{S}$ of feasible schedules by iterating cycles of destruction and construction phases. In our implementation, all random samples are based on a uniform probability distribution over the respective ground set. The destruction phase consists of three steps. At first, we determine a random number $n' \in \{1, \ldots, \lceil n/2 \rceil\}$. Then, we randomly select a set $U \subseteq \{1, \ldots, n\}$ of n' real activities. Finally, we unschedule the activities $i \in U$ from schedule S, obtaining the partial schedule S^C with completed set $C = V\setminus U$. The construction phase extends partial schedule S^C to a complete schedule S'. To this end, we generate a random permutation π of set U and schedule the activities $i \in U$ in that order using the priority-rule based method. This means that we perform the while loop of Algorithm 2 where the total float TF_i is replaced as priority index by the position μ of activity $i = \pi(\mu)$ in permutation π. The acceptance criterion says that we move from S to S' only if S' is better than S, i.e., if $f(S') < f(S)$. We note that the above destruction-construction operator $\mathcal{M}$ typically provides a schedule S' which is not quasistable even if schedule S has been.

3.2 Postprocessing methods

In this subsection we present two postprocessing methods which are used to improve the quality of the complete schedules visited by the iterated greedy method.

3.2.1 Local improvement

When generating the initial solution and during the construction phase, the start times S_i of activities i are chosen to be decision times t at which the additional cost $f_i(S^C, t)$ referring to the current partial schedule S^C is minimized. In the complete schedule S, however, start times S_i need not be minimizers of function $f_i(S^C, \cdot)$ with $C = V\setminus\{i\}$. This means that the iterated schedules S are generally no local minimizers of objective function f on the feasible region $\mathcal{S}$. The reason for this is that start times S_i are fixed in a greedy way where only a part of the resource demand not yet scheduled is anticipated via the consideration of critical and near-critical activities. As a consequence, schedule S can often be improved by using the simple descent method displayed in Algorithm 3. Iterating the real activities i in a random order, we relocate each activity i to a minimum-cost position in the resource profile. Geometrically speaking, in each iteration we move

to a minimizer of the objective function f on the line segment $\ell \subseteq \mathcal{S}$ passing through point S in direction of the i-th unit vector. Since the new start time t minimizes the additional cost $f(S^C, \cdot)$ on interval $[ES_i, LS_i]$, which includes the original start time S_i, the objective function value $f(S)$ is maintained or decreased. We note that the resulting schedule is generally neither quasistable nor a local minimizer of f on $\mathcal{S}$. Experiments with a preliminary version of the iterated greedy algorithm have shown that calling the local improvement function several times until no further decrease in the objective function value is achieved does not provide better results within a fixed time limit than performing only one run of Algorithm 3.

Algorithm 3. *Function local_improvement*

Input: a feasible schedule S
Output: a feasible schedule S with lower or equal objective function value
choose a random permutation π over set V with $\pi(0) = 0$ and $\pi(n + 1) = n + 1$;
for $\mu = 1, \ldots, n$ **do**
put $i := \pi(\mu)$ and $C := V \setminus \{i\}$; (* unschedule activity i *)
determine earliest and latest start times $ES_i = \max_{(j,\, i) \in E}(S_j + \delta_{ji})$ and $LS_i = \min_{(i,\, j) \in E}(S_j - \delta_{ij})$ and compute decision set $\mathcal{D}_i(S^C)$;
select a time $t \in \mathcal{D}_i(S^C)$ with minimum additional cost $f_i(S^C, t)$ and put $S_i := t$; (* reschedule i at time t *)
end for;
return S;

3.2.2 Restoring quasistableness

As we have noticed in Section 2, there always exists an optimal schedule that is quasistable. That is why we can limit our search to the finite set $\mathcal{S}' \subseteq \mathcal{S}$ of quasistable schedules. In what follows we show how the outcome S of the local improvement procedure can be transformed into a quasistable schedule with lower or equal objective function value. Let

$$E(S) := E \cup \{(i,j) \in V \times V \mid i \neq j,\ S_j \geq S_i + p_i\}$$

be the augmented arc set containing the original arcs from project network N as well as an arc (i, j) for each precedence relationship $S_j \geq S_i + p_i$ established by schedule S. The latter arcs (i, j) are weighted by the duration of activity i, i.e., $\delta_{ij} = p_i$ for all $(i, j) \in E(S) \setminus E$. Now recall that if schedule S is quasistable, there exists a spanning tree $T = (V, E')$ with arc set $E' \subseteq E(S)$ such that $S_j = S_i + \delta_{ij}$ for all arcs $(i, j) \in E'$. Otherwise, we can associate schedule S with a corresponding spanning forest $T = (V, E')$ containing a maximum number of arcs $(i, j) \in E(S)$ for which $S_j = S_i + \delta_{ij}$. Let $V' \subset V$ denote the node set of a component of spanning forest T that does not contain node 0. We call V' a free node set since the activities $i \in V'$ may be jointly displaced forwards and backwards in time subject to equations $S_j = S_i + \delta_{ij}$ for all $(i, j) \in E'$. The times $\sigma^-(S, V')$ and $\sigma^+(S, V')$ by which the activities from set V' can be left- or right-shifted, respectively, are

$$\sigma^-(S, V') = \min\{S_i - S_j - \delta_{ji} \mid (j, i) \in E(S),\ i \in V',\ j \in V \setminus V'\}$$
$$\sigma^+(S, V') = \min\{S_j - S_i - \delta_{ij} \mid (i, j) \in E(S),\ i \in V',\ j \in V \setminus V'\}$$

Assume that starting from schedule S we uniformly shift all activities of set V' by some time σ where $-\sigma^-(S, V') \leq \sigma \leq \sigma^+(S, V')$. Interpreting the objective function value $f(S')$ of the destination schedule S' as a function $\varphi(\sigma)$ of the steplength σ, it holds that φ is quasiconcave. This means that the minimum objective function value is attained at one of the two endpoints of the displacement interval $[-\sigma^-(S, V'), \sigma^+(S, V')]$ and hence $f(S)$ can be decreased by left- or right-shifting all activities of V' until a new temporal constraint or precedence relationship from set $E(S)$ becomes binding.

Algorithm 4 shows our procedure for transforming schedule S into a quasistable schedule with lower or equal objective function value. In each iteration the left- and the right-shift of the free node set V_μ with the earliest start time $\min_{i \in V_\mu} S_i$ are considered. The shift providing the greater savings is realized and V_μ is merged with the node set V_λ of the component containing that node j which has determined the displacement time $\sigma^-(S, V_\mu)$ or $\sigma^+(S, V_\mu)$. Note that V_λ may be the non-free node set V_0 containing node 0. We point out that the components of T may also be iterated in a different order and that in general the resulting schedule S depends on the sequence in which the components are shifted.

The quasistability of schedule S can be seen as follows. In each iteration of Algorithm 4 we move to a destination schedule for which an additional inequality $S_j - S_i \geq \delta_{ij}$ with $(i, j) \in E(S)$ becomes binding. Adding all the corresponding arcs (i, j) to spanning forest T leads to a spanning tree with arc set $E'' \supseteq E'$. Since for schedule S it holds that $S_j - S_i = \delta_{ij}$ for all $(i, j) \in E''$ this implies that S is quasistable.

Algorithm 4. *Function quasistable*

Input: a feasible schedule S
Output: a quasistable schedule S with lower or equal objective function value
determine a spanning forest T belonging to schedule S and a sorting $V_1, \ldots, V_\nu$ of the free node sets V_μ according to nondecreasing start times $\min_{i \in V_\mu} S_i$;
for $\mu = 1, \ldots, \nu$ **do**
determine arc $(j', i') \in E(S)$ with $S_{i'} - S_{j'} - \delta_{j'i'} = \sigma^-(S, V_\mu)$;
put $S_i' := S_i - \sigma^-(S, V_\mu)$ for all $i \in V_\mu$ and $S_j' := S_j$ for all $j \in V \setminus V_\mu$;
determine arc $(i'', j'') \in E(S)$ with $S_{j''} - S_{i''} - \delta_{i''j''} = \sigma^+(S, V_\mu)$;

put $S_i'' := S_i + \sigma^+(S, V_\mu)$ for all $i \in V_\mu$ and $S_j'' := S_j$ for all $j \in V \setminus V_\mu$;
if $f(S') \le f(S'')$ **then** determine node set V_λ with $j' \in V_\lambda$ and put $S := S'$;
else determine node set V_λ with $j'' \in V_\lambda$ and put $S := S''$;
end if;
put $V_\lambda := V_\lambda \cup V_\mu$;
end for;
return S;

3.3 Population-based method

Instead of working with only one current schedule S our procedure may maintain a schedule population $\mathcal{P}$ of a given size s_p. In each step of the algorithm, a schedule S is chosen from $\mathcal{P}$ and a new schedule S' is generated by applying the destruction-construction operator $\mathcal{M}$ to S and calling the two postprocessing functions. Schedule S is chosen with a probability proportional to $\max_{S' \in \mathcal{P}} f(S') - f(S) + 1$, i.e., the better schedule S the more probable that it is selected. In the terms of evolutionary algorithms this approach is called a roulette-wheel selection strategy, where the slot size of S equals its fitness. The new solution S' replaces the worst solution S'' of population $\mathcal{P}$ provided that $f(S') < f(S'')$, which in the context of evolutionary algorithms corresponds to a $(\mu + \lambda)$ evolution strategy with $\mu = \lambda = 1$ (see, e.g., Mühlenbein (2003)).

The initial population is generated by introducing some random bias in the priority-rule based method (see Algorithm 2), which is then used as a multi-start procedure with s_p passes. More precisely, we employ a method introduced by Valls et al. (2003) under the name of β-biased random sampling. In each iteration we determine an activity $i' \in V \setminus C$ with minimum total float $TF_{i'}$, where the resource demand is used as tie-breaker (MST-GRD rule). With probability β activity i' is selected to be the activity i that is scheduled in this iteration. Else i is chosen among the other activities with roulette-wheel selection, the slot size of i being equal to $\max_{j \in V \setminus C} TF_j - TF_i + 1$. On average, $100(1 - \beta)$ is the percentage of iterations in which we deviate from the MST-GRD rule of Algorithm 2. The computational experiments discussed in Section 4 have been performed with population size $s_p = 50$ and a β value of 0.9. Those values were chosen after initial tests on a training set.

3.4 Further destruction-construction operators

In this subsection we develop two variants, hereinafter referred to as *DC* operators $\mathcal{M}_A$ and $\mathcal{M}_B$, of the basic destruction-construction operator $\mathcal{M}$ presented in Subsection 3.1.

3.4.1 DC operator $\mathcal{M}_A$

The set U of activities to be unscheduled during the destruction phase is determined by choosing two random integers t, t' with $0 \le t < t' \le S_{n+1}$. Set U consists of those real activities i that in schedule S are started before time t or no earlier than time t', i.e.,

$$U = \{i = 1, ..., n \mid S_i < t\} \cup \{i = 1, ..., n \mid S_i \ge t'\}$$

This way we maintain a block of untouched activities, supposing that this part of the schedule is worth being preserved. In the construction phase, we schedule the activities $i \in U$ in the MST-GRD order. Moreover, with a certain probability α the selected activity i is not scheduled at a decision time t with minimum additional cost $f_i(S^C, t)$ but at a random time $t \in \mathcal{D}_i(S^C)$. After some preliminary experiments we have chosen $\alpha = 0.2$ for our implementation.

3.4.2 DC operator $\mathcal{M}_B$

The set U of activities to be unscheduled is constructed stepwise. Starting with $U = \emptyset$, in each iteration we select one real activity i from the set $V \setminus U$ of remaining activities j according to a roulette-wheel strategy and add i to U. The slot sizes of activities j are chosen to be

$$\sum_{k \in \mathcal{R}} c_k \int_{S_j}^{S_j + p_j} r_k(S, t)\,dt$$

The rationale behind this strategy is that the more an activity j contributes to a peak in the resource profiles of schedule S, the larger will generally be the improvement in the objective function value if we succeed in displacing j appropriately. The construction phase again uses the MST-GRD rule. In difference to operator $\mathcal{M}_A$, however, activities $i \in U$ are always scheduled at a minimum-cost decision time, i.e., $\alpha = 0$.

3.4.3 The general iterated greedy algorithm

Algorithm 5 summarizes the main features of our population-based iterated greedy method containing all enhancements from Subsections 3.2 to 3.4. The population $\mathcal{P}$ of schedules is initialized with s_p schedules generated with the β-biased random sampling method from Subsection 3.3. In each iteration we select one schedule S from $\mathcal{P}$ based on a roulette-wheel selection with fitness slot sizes. We then move from S to a neighboring schedule S' by applying, with equal probability, one of the two destruction-construction operators $\mathcal{M}_A$ and $\mathcal{M}_B$. Finally, we try to improve schedule S' by the postprocessing procedures from Subsection 3.2 and replace the schedule S'' with the largest objective function value in $\mathcal{P}$ by S' if $f(S') < f(S'')$.

Algorithm 5. *Population-based iterated greedy algorithm for (RLP)*

generate an initial population $\mathcal{P}$ of size s_p;
repeat
select a schedule S from $\mathcal{P}$;

draw a uniformly distributed random number $\tilde{u} \in [0, 1[$;
if $\tilde{u} < 0.5$ **then** determine neighbor S' by applying DC operator $\mathcal{M}_A$ to S;
else determine neighbor S' by applying DC operator $\mathcal{M}_B$ to S;
end if;
put $S' :=$ quasistable(local_improvement(S'));
(* postprocessing *)
select worst schedule S'' in population $\mathcal{P}$;
if $f(S') < f(S'')$ **then** replace S'' by S' in $\mathcal{P}$;
until stopping condition is met;
return best schedule S in population $\mathcal{P}$;

4. PERFORMANCE ANALYSIS

In this section we discuss the results of an experimental performance analysis, which compares the iterated greedy method to two state-of-the-art leveling procedures from literature.

4.1 Test bed

The performance of the population-based iterated greedy algorithm is tested based on a data set consisting of 630 problem instances. This data set, which has been generated using the network generator ProGen/max (cf. Kolisch et al. (1999)), has been used previously by Franck et al. (2001) and Neumann et al. (2003). The data set is composed of seven subsets, each containing 90 instances with five resources and 10, 20, 50, 100, 200, 500, or 1000 real activities. The problem instances are characterized by various parameters referring to the activities, the network structure, and the resources. Besides the numbers of activities and resources, the prescribed maximum project duration $\overline{d}$ is an important parameter when dealing with resource leveling problems. Deadline $\overline{d}$ determines the latest start times of the activities and thus the corresponding slack times. In the original data set the maximum project duration $\overline{d}$ is not specified. For each instance we have considered four different values for $\overline{d}$, namely $\overline{d} \in \{ES_{n+1}, 1.1ES_{n+1}, 1.25ES_{n+1}, 1.5ES_{n+1}\}$. Hence, we consider a total of 2520 problem instances clustered into 28 subsets. All results refer to a Pentium personal computer with 1.4GHz clock pulse and 512MB RAM.

4.2 Computational results

At first, we investigate the performance of the population-based iterated greedy algorithm in comparison to procedures from literature. In particular, we consider the priority-rule based method devised in Neumann and Zimmermann (1999b), which is summarized in Algorithm 2, as well as the tabu search algorithm of Neumann and Zimmermann (2000).

The priority-rule based method has been implemented as a multi-start procedure with 10 passes applying priority rule MST in combination with one of the rules

- GRD greatest resource demand $p_i \sum_{k \in \mathcal{R}} r_{ik}$,
- GRU greatest resource utilization $\sum_{k \in \mathcal{R}} r_{ik}$,
- MPA minimum number of parallel activities $|\{j \in V \setminus C \mid [ES_j, LS_j + p_j[\cap [ES_i, LS_i + p_i[\neq \emptyset\}|$,
- RTF minimum relative total float, TF_i/p_i, and
- LST minimum latest start time LS_i

as tie-breakers as well as the corresponding converse priority-rule combinations GRD-MST, GRU-MST, MPA-MST, RTF-MST, and LST-MST. The total running time of the priority-rule based method has always been less than 10 seconds per instance.

For the tabu search procedure, which works with a dynamic tabu list of length 10 to 50 and a candidate list containing 100 schedules, we have imposed a computing time limit of $2n$ seconds. The population-based iterated greedy method (Algorithm 5) has been run with the parameter settings given in Subsections 3.3 and 3.4. As stopping criterion we have used the number of iterations of the repeat-loop, which has been set to 1000. The corresponding computation times in seconds are shown in Table 2.

Table 3 shows the average relative deviations of the objective function values for the best schedules found by the multi-start priority-rule based method from the objective function value yielded by the iterated greedy algorithm. The iterated greedy heuristic is able to improve on the results of the priority-rule based method for each combination of n and $\overline{d}$. The mean gain in accuracy increases as the projects and float times become larger. If we delete the unavoidable resource consumption of the critical and near-critical activities in the objective function, the average accuracy gain increases from 7.5% to 7.9%.

Table 4 shows the average relative deviations between the objective function values provided by the tabu search and the iterated greedy algorithms. The iterated greedy algorithm obtains solutions which are on average about 4.2% better than the solutions generated by the tabu search procedure (the mean deviation equals 4.4% when deleting the unavoidable resource consumption). Again the iterated greedy method compares particularly favorably when the total floats and hence the spaces for leveling are large. Interestingly, the greatest improvements are obtained for medium-sized projects with 50 to 200 activities.

Eventually, we investigate how the different components of the population-based iterated greedy heuristic contribute to the performance of the method. Our analysis is based on the instances with 100 real activities and a run time limit of 3 seconds. Table 5 shows the total improvements obtained compared to the basic version of Algorithm 1 by successively including the different enhancements described in Subsections 3.2 to 3.4. The greatest gain in accuracy is due to the local improvement function (see Subsection 3.2). In case of a large leveling space, restoring the quasistableness of the schedules often leads to an additional improvement, even if the effect remains rather small. The population-based approach (see Subsection 3.3) as well as the more elaborate destruction-construction operators $\mathcal{M}_A$ and $\mathcal{M}_B$ (see Subsection 3.4) again allow for substantial improvements.

Table 2. Computation times in seconds of the iterated greedy method for different numbers n of activities

$n=10$	$n=20$	$n=50$	$n=100$	$n=200$	$n=500$	$n=1000$
0.12	0.28	1.07	2.94	9.54	58.90	459.70

Table 3. Mean relative deviations between objective function values of the priority-rule based method and the iterated greedy algorithm for different numbers n of activities and different deadlines $\overline{d}$

	$\overline{d}=ES_{n+1}$	$\overline{d}=1.1ES_{n+1}$	$\overline{d}=1.25ES_{n+1}$	$\overline{d}=1.5ES_{n+1}$
$n=10$	0.49%	1.71%	3.41%	3.47%
$n=20$	0.62%	3.39%	6.16%	8.22%
$n=50$	2.66%	6.89%	10.31%	13.36%
$n=100$	3.47%	7.70%	11.18%	15.22%
$n=200$	4.47%	7.79%	11.55%	16.19%
$n=500$	4.66%	7.30%	10.42%	15.19%
$n=1000$	3.93%	6.51%	9.39%	13.49%

Table 4. Mean relative deviations between objective function values of the tabu search method and the iterated greedy algorithm for different numbers n of activities and different deadlines $\overline{d}$

	$\overline{d}=ES_{n+1}$	$\overline{d}=1.1ES_{n+1}$	$\overline{d}=1.25ES_{n+1}$	$\overline{d}=1.5ES_{n+1}$
$n=10$	0.26%	0.91%	1.20%	0.65%
$n=20$	0.27%	1.61%	2.52%	3.52%
$n=50$	1.87%	4.83%	7.06%	9.26%
$n=100$	2.21%	5.60%	8.01%	10.63%
$n=200$	2.32%	4.38%	6.87%	10.66%
$n=500$	1.87%	3.25%	5.53%	8.27%
$n=1000$	0.60%	1.98%	3.99%	6.83%

Table 5. Cumulative improvements on objective function values by the different enhancements of Algorithm 1 for $n=100$ activities and different deadlines $\overline{d}$

	$\overline{d}=ES_{n+1}$	$\overline{d}=1.1ES_{n+1}$	$\overline{d}=1.25ES_{n+1}$	$\overline{d}=1.5ES_{n+1}$
local improvement	13.50%	15.50%	18.61%	21.27%
quasistable	13.55%	15.48%	18.53%	21.63%
population-based	19.07%	24.70%	31.14%	38.82%
DC op.'s $\mathcal{M}_A$, $\mathcal{M}_B$	22.48%	32.23%	42.76%	54.82%

5. CONCLUSIONS

In this paper we have presented a project scheduling approach to mid-term production planning in make-to-order production. We have considered the objective of leveling over time the resource usage of the production orders for the main products in the product structure. The problem has been modeled as a resource leveling project scheduling problem. For solving this scheduling problem we have developed an efficient population-based iterated greedy method. An experimental performance analysis including projects with up to 1000 activities shows that the new method compares favorably to leveling heuristics from literature with respect to accuracy and computation time.

An important area of future research is, for example, the adaptation of the iterated greedy method to scheduling problems with alternative objective functions like changeover or resource leasing costs. Moreover, the approach should be generalized to the presence of resource constraints arising from the limited availability of manpower or equipment. Such resource constraints have to be taken into account for the short-term scheduling of production orders on the shop floor.

ACKNOWLEDGEMENTS

This research has been partially supported by the Ministerio de Ciencia y Tecnología (Spain) under contract TIC2002-02510. Part of the work has been developed during the first author's visit at the University of Karlsruhe (Germany). The authors would like to thank four anonymous referees for their valuable comments, which helped to improve the presentation of this paper.

REFERENCES

1. Aarts, E. and Lenstra, J.K. (2003). Introduction. In: E. Aarts and J.K. Lenstra (Eds.), *Local Search in Combinatorial Optimization*, Princeton University Press, Princeton, pp. 1-17.
2. Ahuja, H.N. (1976). *Construction Performance Control by Networks*, John Wiley, New York.
3. Ahuja, R.K., Magnanti, T.L., and Orlin, J.B. (1993).

Network Flows, Prentice Hall, Englewood Cliffs.
4. Bandelloni, M., Tucci, M., and Rinaldi, R. (1994). Optimal resource leveling using non-serial dynamic programming. *European Journal of Operational Research*, 78: 162-177.
5. Braun, O. and Schmidt, G. (2003). Parallel processor scheduling with limited number of preemptions. *SIAM Journal of Computing*, 32: 671-680.
6. Brinkmann, K. and Neumann, K. (1996). Heuristic procedures for resource-constrained project scheduling with minimal and maximal time lags: The resource-levelling and minimum project-duration problems. *Journal of Decision Systems*, 5: 129-156.
7. Burgess, A.R. and Killebrew, J.B. (1962). Variation in activity level on a cyclic arrow diagram. *The Journal of Industrial Engineering*, 13: 76-83.
8. Demeulemeester, E.L. and Herroelen, W.S. (1996). An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. *European Journal of Operational Research*, 90: 334-348.
9. Demeulemeester, E.L. and Herroelen, W.S. (2002). *Project Scheduling: A Research Handbook*, Kluwer, Boston.
10. Dewitte, L. (1964). Manpower leveling of PERT networks. *Data Processing for Science/Engineering*, March–April.
11. Domschke, W. and Drexl, A. (1991). Kapazitätsplanung in Netzwerken. *OR Spektrum*, 13: 63-76.
12. Easa, S.M. (1989). Resource leveling in construction by optimization. *Journal of Construction Engineering and Management*, 115: 302-316.
13. Franck, B., Neumann, K., and Schwindt, C. (1997). A capacity-oriented hierarchical approach to single-item and small-batch production planning using project-scheduling methods. *OR Spektrum*, 19: 77-85.
14. Franck, B., Neumann, K., and Schwindt, C. (2001). Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR Spektrum*, 23: 297-324.
15. Godard, D., Laborie, P., and Nuijten, W. (2005). Randomized large neighborhood search for cumulative scheduling. *Proceedings of the International Conference on Automated Planning and Scheduling ICAPS-05*, Monterrey, pp. 81-89.
16. Harris, R.B. (1978). *Precedence and Arrow Networking Techniques for Construction*, John Wiley and Sons, New York.
17. Harris, R.B. (1990). Packing method for resource leveling (pack). *Journal of Construction Engineering and Management*, 116: 39-43.
18. Hiyassat, M.A.S. (2000). Modification of minimum moment approach in resource levelling. *Journal of Construction Engineering and Management*, 126: 278-284.
19. Hiyassat, M.A.S. (2001). Applying modified minimum moment method to multiple resource levelling. *Journal of Construction Engineering and Management*, 127: 192-198.
20. Hoos, H.H. and Stützle, T. (2004). *Stochastic Local Search: Foundations and Applications*, Morgan Kaufmann, San Francisco.
21. Jacobs, L.W. and Brusco, M.J. (1995). A local search heuristic for large set-covering problems. *Naval Research Logistics Quarterly*, 42: 1129-1140.
22. Karmarkar, U.S. (1987). Lot sizes, lead times and in-process inventories. *Management Science*, 33: 409-418.
23. Kolisch, R., Schwindt, C., and Sprecher, A. (1999). Benchmark instances for project scheduling problems. In: J. Węglarz (Ed.), *Project Scheduling: Recent Models, Algorithms and Applications*, Kluwer, Boston, pp. 197-212.
24. Leu, S.-S., Yang, C.-H., and Huang, J.-C. (2000). Resource leveling in construction by genetic algorithm-based optimization and its decision support application. *Automation in Construction*, 10: 27-41.
25. Levy, F.K., Thompson, G.L., and Wiest, J.D. (1962). Multi-ship multi-shop workload smoothing program. *Naval Research Logistics Quarterly*, 9: 37-44.
26. Lourenço, H.R., Martin, O., and Stützle, T. (2002). Iterated local search. In: F. Glover and G. Kochenberger (Eds.), *Handbook of Metaheuristics*, Kluwer, Boston, pp. 321-353.
27. Marchiori, E. and Steenbeek, A. (2000). An evolutionary algorithm for large set covering problems with applications to airline crew scheduling. In: S. Cagnoni (Ed.), *Lecture Notes in Computer Science, Real-World Applications of Evolutionary Computing*, Springer, Berlin, 1803: 367-381.
28. Meredith, J.R. and Mantel, S.J. (2002). *Project Management: A Managerial Approach*, John Wiley and Sons, New York.
29. Moder, J.J., Phillips, C.R., and Davis, E.W. (1983). *Project Management with CPM, PERT, and Precedence Diagramming*, Van Nostrand Reinhold Company, New York.
30. Mühlenbein, H. (2003). Genetic algorithms. In: E. Aarts and J.K. Lenstra (Eds.), *Local Search in Combinatorial Optimization*, Princeton University Press, Princeton, pp. 137-171.
31. Nahmias, S. (2001). *Production and Operations Analysis*, Irwin, Homewood.
32. Neumann, K. and Schwindt, C. (1995). Projects with minimal and maximal time lags: Construction of activity-on-node networks and applications. Report WIOR-447, University of Karlsruhe.
33. Neumann, K. and Schwindt, C. (1997). Activity-on-node networks with minimal and maximal time lags and their application to make-to-order production. *OR Spektrum*, 19: 205-217.
34. Neumann, K. and Schwindt, C. (1998). A capacitated hierarchical approach to make-to-order production. *European Journal of Automation*, 32: 397-413.
35. Neumann, K. and Zimmermann, J. (1999a). Methods for resource-constrained project scheduling with regular and nonregular objective functions and schedule-dependent time windows. In: J. Węglarz (Ed.), *Project Scheduling: Recent Models, Algorithms and Applications*, Kluwer, Boston, pp. 261-287.
36. Neumann, K. and Zimmermann, J. (1999b). Resource

levelling for projects with schedule-dependent time windows. *European Journal of Operational Research*, 117: 591-605.

37. Neumann, K. and Zimmermann, J. (2000). Procedures for resource levelling and net present value problems in project scheduling with general temporal and resource constraints. *European Journal of Operational Research*, 127: 425-443.
38. Neumann, K., Nübel, H., and Schwindt, C. (2000). Active and stable project scheduling. *Mathematical Methods of Operations Research*, 52: 441-465.
39. Neumann, K., Schwindt, C., and Zimmermann, J. (2003). *Project Scheduling with Time Windows and Scarce Resources*, Springer, Berlin.
40. Özdamar, L. and Yazgaç, T. (1997). Capacity driven due date settings in make-to-order production systems. *International Journal of Production Economics*, 49: 29-44.
41. Palpant, M., Artigues, C., and Michelon, P. (2004). LSSPER: Solving the resource-constrained project scheduling problem with large neigborhood search. *Annals of Operations Research*, 31: 237-257.
42. Ruiz, R. and Stützle, T. (2005). An iterated greedy algorithm for the flowshop problem with sequence dependent setup times. *Proceedings of the 6th Metaheuristics International Conference*, Vienna, pp. 817-823.
43. Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177: 2033-2049.
44. Sawik, T. (2006). Hierarchical approach to production scheduling in make-to-order assembly. *International Journal of Production Research*, 44: 801-830.
45. Schneeweiß, C. and Söhner, V. (1995). Hierarchically integrated lot size optimization. *European Journal of Operational Research*, 86: 73-90.
46. Schwindt, C. (2005). *Resource Allocation in Project Management*, Springer, Berlin.
47. Shaw, P. (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report, University of Strathclyde.
48. Son, J. and Mattila, K.G. (2004). Binary resource leveling model: Activity splitting allowed. *Journal of Construction Engineering and Management*, 130: 887-894.
49. Takamoto, M., Yamada, N., Kobayashi, Y., and Nonaka, H. (1995). Zero-one quadratic programming algorithm for resource leveling for manufacturing process schedules. *Systems and Computers in Japan*, 26: 68-76.
50. Tavares, L.V. (1987). Optimal resource profiles for program scheduling. *European Journal of Operational Research*, 29: 83-90.
51. Valls, V., Quintanilla, S., and Ballestín, F. (2003). Resource-constrained project scheduling: A critical activity reordering heuristic. *European Journal of Operational Research*, 149: 282-301.
52. Younis, M.A. and Saad, B. (1996). Optimal resource leveling of multi-resource projects. *Computers and Industrial Engineering*, 31: 1-4.