Hartmann, Sönke



# Project scheduling with multiple modes: A genetic algorithm

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 435

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 435

# Project Scheduling with Multiple Modes: A Genetic Algorithm

Sönke Hartmann

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 435

# Project Scheduling with Multiple Modes: A Genetic Algorithm

Sönke Hartmann

March 1997

Sönke Hartmann*
Institut für Betriebswirtschaftslehre
Lehrstuhl für Produktion und Logistik
Christian-Albrechts-Universität zu Kiel
Olshausenstraße 40
24118 Kiel, Germany

e-mail: hartmann@bwl.uni-kiel.de
URL: ftp://www.wiso.uni-kiel.de/pub/operations-research

*supported by the *Studienstiftung des deutschen Volkes*

**Abstract**

In this paper we consider the resource-constrained project scheduling problem with multiple execution modes for each activity and makespan minimization as objective. We present a new genetic algorithm approach to solve this problem. The genetic encoding is based on a precedence feasible sequence of activities and a mode assignment. After defining the related crossover, mutation, and selection operators, we describe a local search extension which is employed to improve the schedules found by the basic genetic algorithm. Finally, we present the results of our thorough computational study. We determine the best among several different variants of our genetic algorithm and compare it to three other heuristics that have recently been proposed in the literature. The results that have been obtained using a standard set of instances show that the new genetic algorithm outperforms the other heuristic procedures with regard to a lower average deviation from the optimal makespan.

**Keywords:** Project Management and Scheduling, Multiple Modes, Genetic Algorithms, Local Search, Computational Results.

# 1 Introduction

Within the classical resource-constrained project scheduling problem (RCPSP), the activities of a project have to be scheduled such that the makespan of the project is minimized. Thereby, technological precedence constraints have to be observed as well as limitations of the renewable resources required to accomplish the activities. Once started, an activity may not be interrupted.

This problem has been extended to a more realistic model, the multi-mode resource-constrained project scheduling problem (MRCPSP). Here, each activity can be performed in one out of several modes. Each mode of an activity represents an alternative way of combining different levels of resource requirements with a related duration. Following Slowinski [22], renewable, nonrenewable and doubly constrained resources are dinstinguished. While renewable resources have a limited per-period availability such as manpower and machines, nonrenewable resources are limited for the entire project, allowing to model, e.g., a budget for the project. Doubly constrained resources are limited both for each period and for the whole project. However, since they can simply be incorporated by enlarging the sets of the renewable and nonrenewable resources, we do not consider them explicitly. The objective is to find a mode and a start time for each activity such that the schedule is makespan minimal and feasible w.r.t. the precedence and resource constraints. This problem has been introduced by Elmaghraby [8].

The outlined problem arises within systems for production planning and scheduling as well as project management software. However, as shown by Sprecher and Drexl [27], even the currently most powerful optimization procedures are unable to find optimal schedules for highly resource-constrained projects with more than 20 activities and three modes per activity. Hence, in practice heuristic algorithms to generate near-optimal schedules for larger projects are of special interest.

Several heuristic procedures for solving the MRCPSP have been proposed in the literature: Drexl and Grünewald [6] suggested a regret-based biased random sampling approach. Slowinski et al. [23] described a single-pass approach, a multi-pass approach, and a simulated annealing algorithm based on a precedence feasible activity list and priority rules.

Kolisch and Drexl [15] presented a local search procedure. The only genetic algorithm currently available for the MRCPSP has been proposed by Özdamar [20] and is based on a priority rule encoding. Sprecher and Drexl [26] developed a branch-and-bound procedure which is, according to the results obtained by Hartmann and Drexl [11], the currently most powerful algorithm for exactly solving the MRCPSP. Sprecher and Drexl [27] suggested to use it as a heuristic by imposing a time limit. Finally, Boctor [2], [3], [4] presented heuristics for instances without nonrenewable resources.

This paper introduces a new genetic algorithm (GA) approach for solving the MRCPSP. The search space, i. e. the set of the genotypes, consists of the precedence feasible activity sequences and all mode combinations. The phenotype, i. e. schedule, related to a genotype is generated using a serial scheduling scheme. After defining the genetic operators, we extend the procedure by a local search component which systematically improves the solutions found by the GA. Then we compare our GA to three heuristic approaches proposed in the literature. For this computational comparison, we use a standard set of project instances that has been generated using the problem generator ProGen developed by Kolisch et al. [16].

The remainder of the paper is organized as follows: Section 2 contains the description of the problem. Section 3 gives a brief indroduction into the basic ideas of the theory of evolution that are relevant for our GA. Section 4 describes the new GA approach including different variants of the genetic operators and a local search extension. Section 5 provides the results of our computational experiments. Finally, Section 6 states some conclusions.

## 2 Problem Description

We consider a project which consists of $J$ activities (jobs) labeled $j = 1, \ldots, J$. Due to technological requirements the activities are partially ordered, that is, there are precedence relations between some of the jobs. These precedence relations are given by sets of immediate predecessors $P_j$ indicating that an activity $j$ may not be started before all of its predecessors are completed. The transitive closure of the precedence relations is given by sets of (not necessarily immediate) predecessors $\overline{P}_j$. The precedence relations can be represented by an activity-on-node network which is assumed to be acyclic. We consider additional activities $j = 0$ representing the only source and $j = J + 1$ representing the unique sink activity of the network.

With the exception of the (dummy) source and (dummy) sink activity, each activity requires certain amounts of resources to be performed. The set of renewable resources is referred to as $R$. For each renewable resource $r \in R$ the per-period-availability is constant and given by $K_r^\rho$. The set of nonrenewable resources is denoted as $N$. For each nonrenewable resource $r \in N$ the overall availability for the entire project is given by $K_r^\nu$.

Each activity can be performed in one of several different modes of accomplishment. A mode represents a combination of different resources and/or levels of resource requests with a related duration. Once an activity is started in one of its modes, it is not allowed to be interrupted, and its mode may not be changed. Activity $j$ may be executed in $M_j$ modes labeled $m = 1, \ldots, M_j$. The duration of job $j$ being performed in mode $m$ is given by $d_{jm}$. We assume the modes to be labeled w.r.t. to non-decreasing duration, that is, $d_{jm} \leq d_{j,m+1}$ for all activities $j = 1, \ldots, J$ and modes $m = 1, \ldots, M_j - 1$. Furthermore, activity $j$ executed in mode $m$ uses $k_{jmr}^\rho$ units of renewable resource $r$ each period it is in

process, where we assume w.l.o.g. $k^{\rho}_{jmr} \leq K^{\rho}_{r}$ for each renewable resource $r \in R$. Note, otherwise activity $j$ could not be performed in mode $m$. Moreover, it consumes $k^{\nu}_{jmr}$ units of nonrenewable resource $r \in N$. W.l.o.g., we assume that the dummy source and the dummy sink activity have only one mode each with a duration of zero periods and no request for any resource.

The objective is to minimize the makespan of the project. We assume the parameters to be nonnegative and integer valued. A mathematical programming formulation of this problem has been given by Talbot [30].

# 3 Evolution and Optimization

## 3.1 The Theory of Evolution

In his book "On the Origin of the Species by Means of Natural Selection", Charles R. Darwin (1809-1882) laid the foundation of the theory of evolution. It has later been extended and confirmed by various researchers such as Gregor Mendel (1822-1884) who developed a theory of genetic inheritance.

The process of evolution can be desribed as follows: An individual, or more precisely, its phenotype, consists of basic characteristics contained in its genes, the genotype, and further acquired characterictics. The individuals of a species are similar but differ both in their genotypes and phenotypes. New individuals are produced by crossover, that is, usually two parent individuals mate. The genotype of a child individual is determined by a recombination of the parents genes and mutation, that is, random changes of some genes.

In the struggle for life, the individuals of a species compete for food and mating partners. The fittest individuals of the population survive and may pass on their genes, the others die before they can reproduce. This principle of selection leads to an increasing level of adaption to the species' environment.

Populations may be separated from each other by mountains, deserts, or water. Isolated populations of the same species may develop differently. Nevertheless, some fit individuals may migrate between populations and spread their genes.

The adaption of a species to its environment is called phylogenetic learning. In addition, each individual of a species may learn individually. This process is referred to as ontogenetic learning. The results of phylogenetic learning are passed on to the following generations by means of recombination of the genes as described above. In contrast, the results of ontogenetic learning are not hereditary, that is, changes within the phenotype do not have an influence on the genotype. This seems to be because there is no mechanism to decide whether a change is useful or not, as e.g. injuries or changes due to old age. Only random changes within the genotype by means of mutation may be passed on; together with recombination and selection this forms the process of phylogenetic learning. This contradicts the theory of Jean-Baptiste de Monet Chevalier de Lamarck (1744-1829) who claimed that physical changes of an individual occur if and because they are useful, and that these changes are passed on to its offspring. His teleological theory of evolution, however, has been disproved.

## 3.2 Genetic Algorithms

Genetic algorithms have been developed by Holland [13], for an introduction into GAs we refer to Goldberg [9]. GAs are inspired by the theory of biological evolution and serve as a meta strategy within the fields of e.g. continuous and discrete optimization, machine learning, and game theory. Many variants of GA techniques have been developed over the years, therefore we restrict the following description to the ideas that are used in the GA proposed in this paper.

Roughly speaking, a GA is based on a problem specific encoding and related unary and binary operators, that is, mutation and crossover. First, an initial population is determined and the fitness of the individuals is computed, reflecting the quality of the individuals w.r.t. a given objective. Then new individuals are produced using mutation and crossover. Finally, the next generation is determined by a selection strategy which allows fit individuals to survive and removes the others from the population, that is, they die. Following the isolation principle in nature, this basic scheme is often extended by considering several independently developing populations on different "islands".

The crossover operator recombines parts of two fit individuals to form new ones. These parts, the genes or gene combinations, are assumed to have contributed to the fitness of the parent individuals and are called building blocks. The mutation operator may produce genes or gene combinations that had not occured in the population before. That is, the result of a mutation can be a genotype that could not have been produced by the crossover operator.

Usually, the basic genetic algorithm described above makes only little use of problem specific knowledge. Therefore, as outlined by Grefenstette [10], it is sometimes extended by a local search component that is used to improve the fitness of an individual. While the genetic algorithm roughly navigates through the whole search space in order to identify promising regions, the local search extension systematically scans the neighborhood of an individual. In analogy to biology, the local search component corresponds to individual or ontogenetic learning. In contrast to nature, the artificial evolution also offers the possibility of inheriting the local search results as described by Lamarck.

Viewing the sequence of generations produced by a GA as a Markov chain, Eiben et al. [7] examine the general GA characteristics that make GAs suitable to solve combinatorial optimization problems. They state some simple sufficient conditions under which a GA "almost surely" finds an optimum (i. e. finds an optimum with probability one within an infinite number of generations).

The only GA that has been proposed for the MRCPSP up to now is that of Özdamar [20]. Lee and Kim [18] have suggested a GA for the RCPSP. Kohlmorgen et al. [14] report their experiences with a parallel implementation of a GA for the RCPSP. For the job shop scheduling problem which is, as shown by Sprecher [25], included in the RCPSP as a special case, GAs have been discussed by e.g. Dorndorf and Pesch [5], Herrmann et al. [12], and Mattfeld [19].

# 4 A Genetic Algorithm

## 4.1 Basic Scheme

In this subsection we outline our genetic algorithm approach. Before the GA itself is executed, we apply a preprocessing procedure which adapts the project data in order to reduce the search space. The GA starts by computing an initial population, i. e. the first generation, containing $POP$ individuals and then determines their fitness values. We assume $POP$ to be an even integer. Then the population is randomly partitioned into pairs of individuals. To each resulting pair of (parent) individuals, we apply the crossover operator to produce two new (children) individuals. Subsequently, we apply the mutation operator to the genotypes of the newly produced children. After computing the fitness of each child individual, we add the children to the current population, leading to a population size of $2 \cdot POP$. Then we apply the selection operator to reduce the population to its former size $POP$ and obtain the next generation to which we again apply the crossover operator. This process is repeated for a prespecified number of generations which is denoted as $GEN$.

We consider a number of $ISL$ islands on which the artificial evolution as described above takes place. On each island, the evolution starts with an independently generated initial population. Let the island currently under consideration be denoted as $i$ with $1 \leq i < ISL$, and let the current generation be denoted as $g$ with $1 \leq g \leq GEN$. We use a prespecified migration probability $p_{\text{migration}}$ and draw a random number $q \in [0, 1]$ to control the migration between the islands: If we have $q \leq p_{\text{migration}}$, then the fittest individual of generation $g$ leaves island $i$ and migrates to island $i+1$ where it is added to the population of generation $g$.

The stopping criterion is either to reach a prespecified number of islands as described above or, alternatively, to meet a given limit on the CPU time without bounding the number of islands. In the latter case, if $GEN$ generations have been completed and the time limit has not yet been met, we skip to the next island and start a new evolution. Clearly, if the number of islands is given by $ISL$, at most $ISL \cdot POP \cdot GEN$ different individuals are calculated.

The remainder of this section is organized as follows: In Subsection 4.2, we briefly summarize the preprocessing concepts. Then, in Subsection 4.3, we introduce the genetic representation of the individuals. In Subsections 4.4, 4.5, and 4.6, the crossover, the mutation, and the selection operators are defined, respectively. Finally, in Subsection 4.7 the local search component is described. Throughout this section, we illustrate the definitions using the project example displayed in Figure 1. For the crossover, selection, and local search extension, several variants are described. They have been evaluated in our computational studies the results of which will be summarized in Section 5.

## 4.2 Preprocessing

Before the genetic algorithm itself is executed, the project data is adapted by preprocessing in order to reduce the search space. The reduction procedure has been introduced by Sprecher et al. [29] in order to accelerate a branch-and-bound algorithm for the MRCPSP. We briefly summarize the definitions and results: Sprecher et al. define a mode to be non-executable if its execution would violate the renewable or nonrenewable resource constraints in any schedule. A mode is called inefficient if its duration is not shorter and its resource requests are not less than those of another mode of the same activity. A nonrenewable

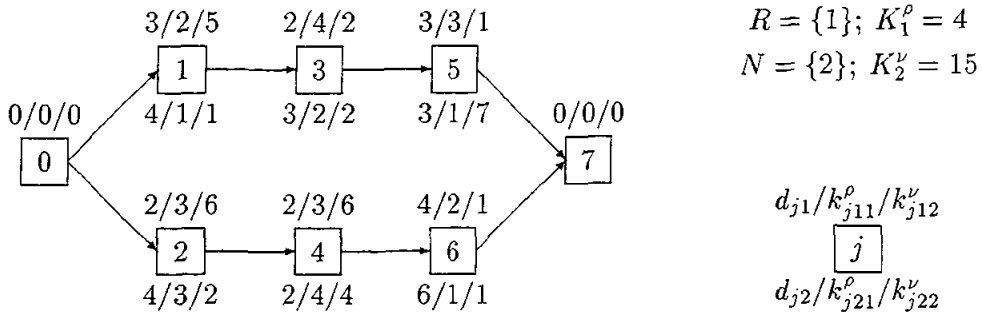


Figure 1: Project instance

resource is called redundant if the sum of the maximal requests of the activities for this resource does not exceed its availability. Clearly, redundant nonrenewable resources as well as non-executable and inefficient modes may be deleted from the project data without affecting the set of the optimal solutions. As there are interaction effects between the elimination of modes and nonrenewable resources, the project data is adapted as follows: First, all non-executable modes are deleted. Second, all redundant renewable resources and, subsequently, all inefficient modes are removed. The second step is repeated until no redundant renewable resources are left. The result of this adaption is a reduction of the number of feasible as well as infeasible solutions.

Consider the project instance given in Figure 1. If activity 5 was performed in mode 2, the whole project would require at least 17 units of the nonrenewable resource whereas only 15 units are available. Consequently, mode 2 of activity 5 is non-executable w.r.t. to the nonrenewable resource and may therefore be deleted.

### 4.3 Individuals and Fitness

In our GA, an individual $I$ is represented by a pair of an activity sequence and a mode assignment and is denoted as

$$I = ((j_1^I, \ldots, j_J^I), m^I).$$

The job sequence $j_1^I, \ldots, j_J^I$ is assumed to be a precedence feasible permutation of the set of activities, that is, we have $\{j_1^I, \ldots, j_J^I\} = \{1, \ldots, J\}$ and $P_{j_i^I} \subseteq \{j_1^I, \ldots, j_{i-1}^I\}$ for $i = 1, \ldots, J$. The mode assignment $m^I$ is a mapping which assigns to each activity $j \in \{1, \ldots, J\}$ a mode $m^I(j) \in \{1, \ldots, M_j\}$. In the examples of this section, we use the equivalent notation for the individuals which is displayed in Figure 2.

$$I = \begin{array}{|c|c|c|} \hline j_1^I & \cdots & j_J^I \\ \hline m^I(j_1^I) & \cdots & m^I(j_J^I) \\ \hline \end{array}$$

Figure 2: Genotype

Each genotype is related to a uniquely determined schedule (phenotype) which is computed as follows: First, the dummy source activity is started at time 0. Then we schedule the activities in the order that is prescribed by the sequence $j_1^I, \ldots, j_J^I$. Thereby, activity $j_i^I$ is scheduled in mode $m^I(j_i^I)$ and assigned the earliest feasible start time. Note that the result is an active schedule, that is, no activity can be left shifted without violating the constraints (for a formal definition of active schedules cf. Sprecher et al. [28]).

Clearly, the schedule related to an individual is feasible with respect to the precedence relations and the renewable resource constraints, but not necessarily w.r.t. the nonrenewable resource constraints. However, it is useful to include schedules that are infeasible w.r.t. the nonrenewable resources into the search space because, as proven by Kolisch and Drexl [15], already finding a feasible schedule is an NP-complete problem if at least two nonrenewable resources are given.

The fitness of an individual $I$ is computed as follows: Let $T$ be the upper bound on the project's makespan given by the sum of the maximal durations of the activities. Moreover, let $L_r^\nu(I)$ denote the leftover capacity of nonrenewable resource $r \in N$ w.r.t. the modes selected by the genotype of individual $I$, that is,

$$L_r^\nu(I) = K_r^\nu - \sum_{j=1}^{J} k_{jm^I(j)r}^\nu.$$

If there is a nonrenewable resource $r \in N$ with $L_r^\nu < 0$, then the mode assignment of individual $I$ is infeasible w.r.t. the nonrenewable resource constraints. In this case, the fitness of $I$ is given by

$$f(I) = T + \sum_{\substack{r \in N \\ L_r^\nu(I) < 0}} |L_r^\nu(I)|.$$

Otherwise, if individual $I$ is feasible w.r.t. the nonrenewable resources, the fitness $f(I)$ of individual $I$ is given by the makespan of the related schedule. From the definitions given above it is clear that a lower fitness of an individual implies a better quality of the related schedule. Observe that a feasible individual always has a lower fitness than an infeasible one.

For illustration, we consider the project instance given in Figure 1 and the two individuals $M$ and $F$ displayed in Figure 3. Clearly, the mode assignment of individual $M$ is feasible as 15 units of the nonrenewable resource are requested, that is, the capacity is not exceeded. Now we derive a schedule from the genotype of $M$ which can be found in Figure 4, where $j(m)$ stands for activity $j$ being performed in mode $m$. The fitness of $M$ is equal to the makespan of the schedule, that is, we have $f(M) = 15$. Individual $F$ induces a nonrenewable resource requirement of 19 units which exceeds the availability by 4 units. Computing $T = 22$, we obtain a fitness value of $f(F) = 26$.

It should be noticed that the permutation based genetic encoding includes some redundancy. This can be seen in genotype of indvidual $M$ of Figure 3: Interchanging activities 1 and 6 in this genotype obviously results in another genotype with a precedence feasible job sequence. The schedule related to this other genotype, however, is the same as that of the original genotype of $M$. Figure 4 shows that this is due to the fact that activities 1 and 6 start at the same time. In other words, different genotypes, i. e. elements of the search space, may be related to the same schedule.

$M =$

| 2 | 4 | 1 | 6 | 3 | 5 |
|---|---|---|---|---|---|
| 2 | 2 | 1 | 1 | 1 | 1 |

$F =$

| 1 | 3 | 2 | 5 | 4 | 6 |
|---|---|---|---|---|---|
| 1 | 2 | 1 | 1 | 2 | 2 |

Figure 3: Example individuals

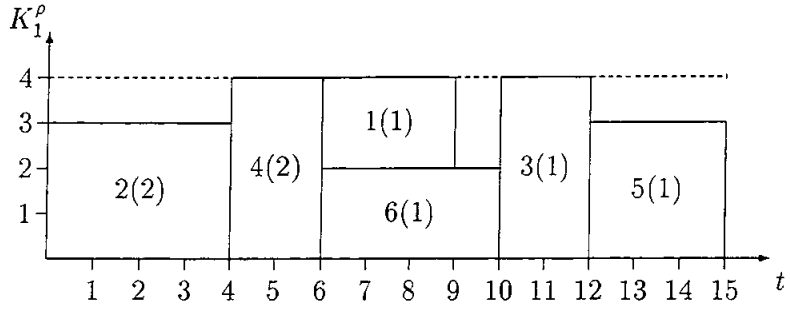


Figure 4: Schedule of individual $M$

### 4.4 Crossover

We consider two individuals selected for crossover, a mother $M$ and a father $F$. Then we draw two random integers $p_1$ and $p_2$ with $1 \leq p_1, p_2 \leq J$. Now two new individuals, a daughter $D$ and a son $S$, are produced from the parents. We first consider $D$ which is defined as follows: In the sequence of jobs of $D$, the positions $i = 1, \ldots, p_1$ are taken from the mother, that is,

$$j_i^D := j_i^M.$$

The job sequence of positions $i = p_1 + 1, \ldots, J$ in $D$ is taken from the father. However, the jobs that have already been taken from the mother may not be considered again. Following the general crossover technique described by Reeves [21] for permutation based genotypes, we obtain:

$$j_i^D := j_k^F \text{ where } k \text{ is the lowest index such that } j_k^F \notin \{j_1^D, \ldots, j_{i-1}^D\}.$$

This definition ensures that the relative positions in the parents' job sequences are preserved. Observe that the resulting job sequence is precedence feasible.

The modes of the activities on the positions $i = 1, \ldots, p_2$ in daughter $D$ are defined by the mother's mode assignment $m^M$, that is,

$$m^D(j_i^D) := m^M(j_i^D).$$

The modes of the remaining jobs on the positions $i = p_2 + 1, \ldots, J$ in $D$ are derived from the father's mode assignment $m^F$:

$$m^D(j_i^D) := m^F(j_i^D).$$

The son $S$ of the individuals $M$ and $F$ is computed similarly. However, the positions $1, \ldots, p_1$ of the son's job sequence are taken from the father and the remaining positions are determined by the mother. Analogously, the first part up to position $p_2$ of the mode assignment of $S$ is taken from $F$ while the second part is derived from $M$.

The above definitions are illustrated by the following example. Using the project instance of Figure 1 and the individuals shown in Figure 3, we set $p_1 := 3$ and $p_2 := 4$ and compute the children displayed in Figure 5. We consider the daughter $D$. The first three positions of the job sequence are equal to those of $M$. The order of the remaining activities is taken from $F$. According to the value of $p_2$, the modes of the first four activities in the job sequence of $D$ are determined by $M$ while the last two activities get their modes from $F$. Observe that, as we have $p_1 < p_2$ in this example, the fourth activity of the daughter's job sequence, activity 3, is determined by the father's job sequence. The mode of activity 3, however, is taken from the mother.

$D =$

| 2 | 4 | 1 | 3 | 5 | 6 |
|---|---|---|---|---|---|
| 2 | 2 | 1 | 1 | 1 | 2 |

$S =$

| 1 | 3 | 2 | 4 | 6 | 5 |
|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 1 | 1 |

Figure 5: Children

We have considered two variants of this crossover operator: First, the general variant as defined above and, second, a simplified version in which we randomly draw only $p_1$ and set $p_2 := p_1$.

### 4.5 Mutation

The mutation included in our GA is applied to each newly generated child individual and is defined as follows: Given an individual $I$ of the current population, we draw two random integers $q_1$ and $q_2$ with $1 \leq q_1 < J$ and $1 \leq q_2 \leq J$. $q_1$ is used to modify the job sequence by exchanging activties $j^I_{q_1}$ and $j^I_{q_1+1}$ if the result is a job sequence which fulfills the precedence constraints (cf. Subsection 4.3). Note that each of the changed activities keeps its assigned mode, that is, this modification does not change the mode assignment. Then we randomly choose a new mode for the activity on position $q_2$, that is, we redetermine $m^I(j^I_{q_2})$ by drawing a random integer out of $\{1, \ldots, M_{j^I_{q_2}}\}$. While the first step may create job sequences (i. e. gene combinations) that could not have been procuced by the crossover operator, the second step may introduce a mode (i. e. gene) that has not occured in the current population.

It should be noted that performing a mutation on an individual does not necessarily change the related schedule. This is due to the redundancy in the genetic representation that has been described in Subsection 4.3. We consider again the project instance shown in Figure 1, individual $M$ of Figure 3, and the related schedule displayed in Figure 4. Choosing $q_1 = 3$ induces an exchange of activities 1 and 6 in the job sequence. However, activities 1 and 6 are assigned a start time of 6 no matter if they are changed in this job sequence or not.

### 4.6 Selection

We consider two variants of the selection operator. The first variant is a simple survival-of-the-fittest method: We restore the original population size by keeping the $POP$ best individuals and removing the remaining ones from the population (ties are broken arbitrarily).

The second variant is a randomized version of the previously described survival-of-the-fittest technique. Let $\mathcal{P}$ denote the current population, that is, a list containing the individuals. Note that we use a list of individuals instead of a set because we explicitly allow two (or more) distinct individuals with the same genotype in a population. We restore the original population size by successsively removing individuals from the population until $POP$ individuals are left, using the following probability: Denoting with $f_{\text{best}} = \min\{f(I) \mid I \in \mathcal{P}\}$ the best fitness in the current population, the probability to die for an individual $I$ is given by

$$p_{\text{death}}(I) = \frac{(f(I) - f_{\text{best}} + 1)^2}{\sum_{I' \in \mathcal{P}} (f(I') - f_{\text{best}} + 1)^2} .$$

### 4.7 Improvement by Local Search

In this subsection we discuss a problem specific local search method to improve the schedule related to an individual. The approach is based on the definition of a multi-mode left shift which has been introduced by Sprecher et al. [29] in order to accelerate their branch-and-bound algorithm for the MRCPSP. A multi-mode left shift of an activity $j$ is an operation on a given schedule which reduces the finish time of activitity $j$ without changing the modes or finish times of the other activities and without violating the constraints. Thereby, the mode of activity $j$ may be changed.

We employ the local search procedure after transforming an individual $I$ into a schedule. If the schedule is feasible w.r.t. the nonrenewable resources, we try to improve it as follows: For each activity $j_1^I, \ldots, j_J^I$, we check whether a multi-mode left shift can be performed. Thereby, the modes of an activity $j_i^I$ are tested w.r.t. non-decreasing duration, that is, in the order given by $1, \ldots, M_{j_i^I}$. For each activity, the first feasible multi-mode left shift found (if any) is applied to the schedule. Then we skip to the next activity in the job sequence. The result is a feasible schedule with a makespan equal to or lower than the original one. Now the fitness of individual $I$ is set to be equal to the new makespan. This improvement procedure is applied to all individuals of the current population for which we have to compute the fitness.

We consider again the project instance given in Figure 1. The schedule shown in Figure 4 related to individual $M$ of Figure 3 is improved by the procedure described above as follows: While activities 2 and 4 cannot be left-shifted, we can perform a multi-mode left shift on activity 1. Leaving activity 6 unchanged, we can now apply a multi-mode left-shift to activity 3. Finally, left-shifting activity 5 yields a new makespan of 13 periods. The resulting schedule is displayed in Figure 6.

As each activity is considered only once for a multi-mode left shift, we call the above procedure single pass local search algorithm. Note, however, that the schedule derived by this single pass approach is not necessarily tight (a tight schedule is a schedule to which no multi-mode left shift can be applied, cf. Speranza and Vercellis [24] and Sprecher et

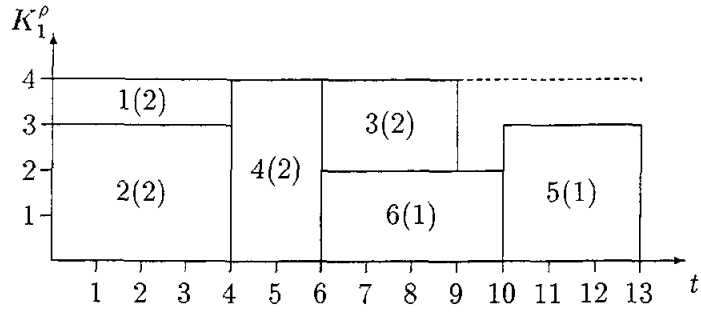


Figure 6: Improved schedule of individual $M$

al. [29]). This is because a multi-mode left shift of some activity $j_i^I$ might allow a multi-mode left shift of some activity $j_k^I$ with $k < i$ that had not been possible before. In the improved schedule of Figure 6, for example, a multi-mode left shift of activity 2 is possible because the total consumption of the nonrenewable resource decreased due to the multi-mode left shift of activity 1. Consequently, we consider a second variant of our local search extension which we call multi pass procedure: We repeatedly apply the above single pass algorithm until we have obtained a tight schedule. In contrast to the single pass algorithm, the multi pass approach always leads to a local optimum as the resulting schedule cannot be further improved using multi-mode left shifts. Note, however, that these procedures cannot improve schedules for the single-mode RCPSP found by the GA, because in this case the set of tight schedules coincides with the set of the active ones (cf. Sprecher et al. [29]), and each schedule computed by the evaluation function of Subsection 4.3 is active.

Both local search procedures can be viewed as a second step of the genotype evaluation as they compute fitness values. But we can do more: Applying one of these procedures to a schedule $S$ related to an individual $I$, we obtain an improved schedule $S'$. Now we can transform the new schedule $S'$ into an individual $I'$, that is, we can find an individual $I'$ which corresponds to schedule $S'$. Clearly, the job sequence of individual $I'$ is given by the sequence of activities ordered w.r.t. non-decreasing start times of schedule $S'$; the mode assignment is straightforward. Subsequently, we can replace individual $I$ with the improved individual $I'$ in the current population. Considering evolution in biology, the single and multi pass procedures above which only change the phenotype (schedule) can be compared to individual or ontogenetic learning. The transformation of their results into a new genotype, i. e. into hereditary information, corresponds to the possibility to inherit the results of ontogenetic learning as described by Lamarck.

# 5 Computational Results

## 5.1 Experimental Design

In this section we present the results of the computational studies concerning the genetic algorithm introduced in the previous section. The experiments have been performed on a Pentium-based IBM-compatible personal computer with 133 MHz clock-pulse and 16 MB RAM. The GA has been coded in ANSI C, compiled with the GNU C compiler and tested under Linux.

We used a set of standard test problems systematically constructed by the project generator ProGen which has been developed by Kolisch et al. [16]. They are available in the project scheduling problem library PSPLIB from the University of Kiel. For detailed information the reader is referred to Kolisch and Sprecher [17]. Some of the instances have been used by Kolisch and Drexl [15] and Özdamar [20] to evaluate their heuristics for the MRCPSP.

In our study, we have used the multi-mode problem sets containing instances with 10, 12, 14, 16, 18, and 20 non-dummy activities. Each of the non-dummy activities may be performed in one out of three modes. The duration of a mode varies between 1 and 10 periods. We have two renewable and two nonrenewable resources. For each problem size, a set of instances was generated by systematically varying four parameters, that is, the resource factor and the resource strength of each resource category. The resource factor is a measure of the average portion of resources requested per job. The resource strength reflects the scarceness of the resources. For each project size, 640 instances were generated. Those instances for which no feasible solution exists have not been considered. Hence, we have 536 instances with $J = 10$, 547 instances with $J = 12$, 551 instances with $J = 14$, 550 instances with $J = 16$, 552 instances with $J = 18$, and 554 instances with $J = 20$. The set with 20 non-dummy activities currently is the hardest standard set of multi-mode instances for which all optimal solutions are known, cf. Sprecher and Drexl [27].

## 5.2 Configuration of the Algorithm

In the numerical investigation reported in this subsection we determined the best configuration of our GA. We selected the instance set with 20 non-dummy activities per project for these experiments.

In the best configuration, we have a population size of 50 individuals and 30 generations on two islands. Moreover, the best configuration consists of the general crossover variant, the deterministic selection operator, and the single pass multi-mode left shift concept without inheritance, but it does not include migration between the islands, that is, $p_{\text{migration}} = 0$. In the following, we summarize the most important computational results which confirm the superiority of this configuration.

In a first step we confirm the relationship between the number of islands, the population size, and the number of generations stated above. We set the number of individuals (i. e. schedules) to be examined to 3000 per instance, resulting in an average computation time of 1.2 seconds per instance. The impact of varying $ISL$, $POP$, and $GEN$ within the best configuration is documented in Table 1. For each tested parameter combination, the average deviation from the optimal makespan, the maximal deviation, the fraction of instances for which a feasible schedule has been found, and the fraction of instances for which an optimal schedule has been found are given. Choosing $POP = 50$, $GEN = 30$, and $ISL = 2$ yields the best results, an average deviation of 1.6 % from the optimal solution. Notice that stopping the evolution of 50 individuals after 30 generations on two islands is more promising than to have 60 generations and only one island. Consider the last row of Table 1: Selecting a population size of 3000, only one generation and one island induces a simple procedure which randomly generates 3000 schedules without employing the genetic operators (note, however, that the single pass local search concept is included). The results show that this random procedure is clearly outperformed by the "real" GA. This confirms

the quality of our genetic encoding and the related operators.

| *POP* | *GEN* | *ISL* | av. dev. | max. dev. | feasible | optimal |
|---|---|---|---|---|---|---|
| 30 | 20 | 5 | 2.3 % | 21.7 % | 100.0 % | 60.3 % |
| 30 | 50 | 2 | 1.9 % | 17.9 % | 100.0 % | 64.1 % |
| 50 | 30 | 2 | 1.6 % | 10.5 % | 100.0 % | 67.7 % |
| 50 | 60 | 1 | 1.8 % | 13.9 % | 100.0 % | 66.3 % |
| 100 | 30 | 1 | 1.7 % | 14.3 % | 100.0 % | 66.6 % |
| 3000 | 1 | 1 | 6.8 % | 60.7 % | 99.1 % | 45.1 % |

Table 1: Impact of the number of generations — 3000 individuals, $J = 20$

The second step to verify the quality of the configuration given above is to determine the best variants of the genetic operators. Here we have selected a population size of 50 and a generation number of 30 and imposed a time limit of one second (without limiting the number of islands). This foregoing is useful because some configuration changes may slightly improve the solution quality, but on the other hand may drastically increase the computation times. Table 2 summarizes the results of the changes of the best configuration. Using the simple crossover variant (i.e. $p_1 = p_2$) instead of the general one, omitting the mutation operator, and employing the randomized selection operator instead of the deterministic one deteriorate the results. Moreover, allowing migration between the islands does not improve the solution quality.

| Configuration | av. dev. | max. dev. | feasible | optimal |
|---|---|---|---|---|
| best | 1.9 % | 11.9 % | 100.0 % | 64.8 % |
| simple crossover | 2.2 % | 15.2 % | 100.0 % | 61.7 % |
| without mutation | 2.8 % | 17.9 % | 100.0 % | 54.5 % |
| randomized selection | 2.4 % | 28.6 % | 100.0 % | 58.0 % |
| $p_{\text{migration}} = 0.25$ | 2.1 % | 17.9 % | 100.0 % | 62.8 % |

Table 2: Impact of alternative genetic operators — 1 second, $J = 20$

Finally, the third step to confirm the configuration stated above as the best one is to examine several alternatives of the local search extension. Again, we have selected a population size of 50, a generation number of 30, and a time limit of one second. Employing the best genetic operators yields the results displayed in Table 3. They show that the single pass multi-mode left shift concept without inheritance is capable of improving the average deviation of the basic GA without local search extension by a factor of approximately 1.4. However, applying the multi-mode left shift procedure until tight schedules are achieved does not have an additional positive impact on the solution quality. This is because the solutions found by the multi pass procedure are only slightly better than those of the single pass concept while it requires more computation time, that is, a smaller number of individuals can be generated within the time limit. Also the inheritance of the local search results, that is, the transformation of the improved phenotype (i. e. schedule) into a related genotype

(i. e. individual), does not yield a further improvement of the computational results.

| Local search | inheritance | av. dev. | max. dev. | feasible | optimal |
|---|---|---|---|---|---|
| none | — | 2.6 % | 21.7 % | 100.0 % | 56.1 % |
| single pass | no | 1.9 % | 11.9 % | 100.0 % | 64.8 % |
| multi pass | no | 1.9 % | 15.0 % | 100.0 % | 64.1 % |
| single pass | yes | 2.3 % | 20.0 % | 100.0 % | 61.2 % |
| multi pass | yes | 2.4 % | 27.3 % | 100.0 % | 60.5 % |

Table 3: Impact of local search variants — 1 second, $J = 20$

### 5.3 Population Analysis

In this subsection we analyze the generations produced by the GA in order to answer a question that has been arising in the previous subsection: Why is it disadvantageous to inherit the local search results?

First we define a measure for the similarity of two individuals $I$ and $I'$. We start with a definition of the similarity of the related activity sequences. Our goal is to check if two activities have the same relative positions in the activity sequences of both $I$ and $I'$. Since it is sufficient to consider only those activities that are not precedence related, we define

$$\mathcal{M} = \{\{i,j\} \mid i,j = 1,\ldots,J;\ i \neq j;\ i \notin \overline{P}_j;\ j \notin \overline{P}_i\}.$$

Given two activities $i$ and $j$ that are not precedence related, i. e. $\{i,j\} \in \mathcal{M}$, we reflect their relative positions within $I$ and $I'$ by

$$\alpha_{\{i,j\}}^{I,I'} = \begin{cases} 1, & \text{if } i \text{ is before } j \text{ in both } I \text{ and } I', \text{ or if } j \text{ is before } i \text{ in both } I \text{ and } I', \\ 0, & \text{otherwise.} \end{cases}$$

Now we are ready to define the following measure for the similarity of the job sequences of $I$ and $I'$: If there are activities that are not precedence related, i. e. $\mathcal{M} \neq \emptyset$, we set

$$\alpha^{I,I'} = \frac{1}{|\mathcal{M}|} \sum_{\{i,j\} \in \mathcal{M}} \alpha_{\{i,j\}}^{I,I'}.$$

Otherwise, if $\mathcal{M} = \emptyset$, we have a serial network structure which implies that there is only one precedence feasible job sequence. In this case, we define $\alpha^{I,I'} = 1$.

The next definition reflects whether an activity $j$ is assigned the same mode by two individuals $I$ and $I'$:

$$\mu_j^{I,I'} = \begin{cases} 1, & \text{if } m^I(j) = m^{I'}(j), \\ 0, & \text{otherwise.} \end{cases}$$

This enables us to define the following measure for the similarity of the mode assignments of $I$ and $I'$:

$$\mu^{I,I'} = \frac{1}{J} \sum_{j=1}^{J} \mu_j^{I,I'}.$$

Combining the above definitions, we obtain a measure $\sigma^{I,I'}$ for the similarity of individuals $I$ and $I'$ in which both the activity sequences and the mode assignments are considered:

$$\sigma^{I,I'} = \frac{\alpha^{I,I'} + \mu^{I,I'}}{2}.$$

Clearly, the higher $\sigma^{I,I'}$, the more identical information is contained in the genotypes of individuals $I$ and $I'$. Observe that we have $\sigma^{I,I'} \in [0,1]$. Especially, we have $\sigma^{I,I'} = 0$ if $I$ and $I'$ do not have any genetic information in common, and $\sigma^{I,I'} = 1$ if they are identical, that is, if we have $I = I'$.

Now we use this similarity measure for analyzing the different generations produced by our GA. More precisely, each generation is partitioned into sets of similar individuals. Therefore we have implemented a cluster analysis algorithm.[1] Given a generation with $POP$ individuals, we first compute the similarity value for each pair of individuals. Then the cluster analysis algorithm starts with the trivial partition in which each individual forms a cluster, that is, we have $POP$ clusters. After that we unite clusters as follows: Consider two clusters $C$ and $C'$. $C$ and $C'$ are united if we have $\sigma^{I,I'} \geq 0.8$ for all individuals $I \in C$ and $I' \in C'$. This is repeated until there are no clusters left that can be united w.r.t. this similarity criterion. Each resulting cluster contains highly similar individuals.

For the following experiment, we have tested two variants of the GA: first, the best configuration without inheritance of the single pass local search results and, second, the best configuration with inheritance. While the number of individuals is again fixed to 50 per generation, we have observed the development on one island only. We have applied both variants to the hardest of the instances with 20 activities, that is, those with a high resource factor and a low resource strength implying scarce resources. Table 4 lists the average number of clusters that have been obtained by both variants for every fifth generation. As the first generation is randomly determined, there are no similarities of more than 0.8, inducing 50 clusters with one individual each. With an increasing number of generations, the number of clusters in a generation decreases, that is, more similar individuals occur in the population. Clearly, this is due to the crossover and selection operators which tend to copy "fit" and remove "unfit" information. Table 4, however, shows that the inheritance mechanism accelerates the reduction of clusters: While the GA without inheritance leads to 13 different clusters after 30 generations, the GA with inheritance results in only 4 clusters at the same time.

| Inheritance | 1 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| no | 50 | 34 | 27 | 22 | 18 | 15 | 13 | 10 | 9 | 7 | 7 | 7 | 6 |
| yes | 50 | 28 | 19 | 11 | 7 | 6 | 4 | 3 | 3 | 3 | 3 | 2 | 2 |

Table 4: Average number of clusters w.r.t. generation number

These results explain why including the inheritance mechanism into the GA deteriorates the quality of the solutions: As already outlined, the basic strategy of any GA is to gather information about promising regions of the search space. Each of our clusters can be viewed

[1]For a general introduction into cluster analysis the reader is referred to e.g. Backhaus et al. [1].

as such a promising region. At the same time, however, information that is not considered as promising is removed. Clearly, this leads to a loss of genetic variety. If the number of clusters decreases too fast—in other words, if too much information is lost too fast—the GA gets stuck in some promising regions of the search space. As each cluster contains similar information, the GA is likely to fail to construct individuals from previously unsearched regions of the (usually huge) search space if only few clusters are left. Consequently, many regions remain unexplored, and better solutions may be left undetected.

The arguments above also explain why it is better to consider two islands with 30 generations each instead of 60 generations on one island: Table 4 indicates that the GA (without inheritance) results in too few clusters after more than 30 generations.

Basically, one encounters the following difficulty when designing a GA: An evolution proceeding too fast leads to a loss of genetic variety and is thus disadvantageous. On the other hand, an evolution proceeding too slowly may be unable to identify promising regions of the search space. Therefore, the variants and parameters have to be chosen carefully.

### 5.4 Comparison with other Heuristics

In this subsection we summarize the results obtained from a comparison of our GA with three other heuristics for solving the MRCPSP that have recently been proposed in the literature.

For the comparison of our GA with the approaches of Kolisch and Drexl [15] and Özdamar [20], we have selected the standard instance set with 10 non-dummy activities and fixed the number of individuals to 3000 (without imposing a time limit). The results given in Table 5 show that the new GA produces an average deviation of 0.22 % from the optimal makespan. Kolisch and Drexl [15] have suggested a local search procedure which constructs an initial solution and tries to improve it by neighborhood moves based on slack calculations. They report that their approach outperforms the algorithms of Boctor [2] and of Drexl and Grünewald [6]. We have recompiled their original PASCAL code, limited the number of neighborhood moves to 3000 and obtained an average deviation of more than 0.8 %. Özdamar [20] has developed a GA which is based on a genetic representation of a sequence of priority rules that is used within a parallel scheduling scheme. Each individual is evaluated both in a forward and in a backward evaluation of the priority rule sequence. For 3000 individuals, Özdamar [20] reports an average deviation of more than 0.8 %. Hence, the average deviation produced by the new GA is nearly four times lower than those of the two heuristics from the literature.

| Heuristic | av. dev. | feasible | optimal |
|---|---|---|---|
| new GA | 0.22 % | 100.0 % | 96.3 % |
| Kolisch, Drexl | 0.82 % | 100.0 % | 89.0 % |
| Özdamar | 0.86 % | 100.0 % | 88.1 % |

Table 5: Comparison with two other heuristics — 3000 individuals, $J = 10$

Next, we compare the new GA with a truncated version of the branch-and-bound procedure of Sprecher and Drexl [26]. As shown by Hartmann and Drexl [11], this is the currently

most efficient exact approach for solving the MRCPSP. We have coded the branch-and-bound procedure including all available bounding rules in C using the same data structures as in the implementation of our GA when possible. As suggested by Sprecher [25], we have employed the job number rule into the branch-and-bound algorithm, that is, the next eligible activity to be selected is the one with the lowest number. The modes are selected w.r.t. non-decreasing duration.

Table 6 displays the results obtained from both algorithms with a time limit of one second. While the truncated exact procedure solves all instances with 10 activities to optimality within one second, its average deviation for the instances with 20 activities is six times higher than that obtained by the GA. In contrast to the GA which results in moderate maximal deviations of at most 15 %, the maximal deviation of the truncated branch-and-bound algorithm is almost 80 % for $J = 20$. While our GA finds a feasible solution for every instance, the truncated exact procedure fails to do so for instances with more than 12 activities.

| Heuristic | $J$ | av. dev. | max. dev. | feasible | optimal |
|---|---|---|---|---|---|
| new GA | 10 | 0.15 % | 10.5 % | 100.0 % | 97.6 % |
| truncated b&b | 10 | 0.00 % | 0.0 % | 100.0 % | 100.0 % |
| new GA | 12 | 0.17 % | 7.1 % | 100.0 % | 96.3 % |
| truncated b&b | 12 | 0.12 % | 17.9 % | 100.0 % | 98.2 % |
| new GA | 14 | 0.68 % | 15.0 % | 100.0 % | 86.6 % |
| truncated b&b | 14 | 1.46 % | 33.3 % | 99.6 % | 85.7 % |
| new GA | 16 | 1.00 % | 12.9 % | 100.0 % | 78.6 % |
| truncated b&b | 16 | 3.81 % | 52.4 % | 99.5 % | 69.5 % |
| new GA | 18 | 1.47 % | 13.0 % | 100.0 % | 71.4 % |
| truncated b&b | 18 | 7.48 % | 77.4 % | 98.0 % | 57.4 % |
| new GA | 20 | 1.91 % | 11.9 % | 100.0 % | 64.8 % |
| truncated b&b | 20 | 11.51 % | 78.6 % | 96.4 % | 47.3 % |

Table 6: Comparison with truncated branch-and-bound — 1 second, all instances

# 6 Conclusions

We have presented a genetic algorithm for solving project scheduling problems with multiple modes. The numerical results show that the GA is successful in finding feasible schedules which is remarkable as already the feasibility problem is NP-complete if at least two nonrenewable resources are given. A computational comparison revealed that our GA outperforms other heuristics proposed in the literature for the MRCPSP. Of special interest, possibly also for other scheduling problems, is that the permutation based genetic representation yields better results than a priority rule based encoding. Moreover, the problem specific local search extension is capable of improving the solutions found by the basic GA.

These results are encouraging for future research on other scheduling problems which include hard feasibility problems such as e.g. the RCPSP with maximal time lags: These problems may also be solved using a (possibly permutation based) genetic encoding which

includes infeasible schedules into the search space. Infeasibility can then be penalized with disadvantageous fitness values. A local search component using problem specific knowledge can be added for supporting the process of finding feasible or near-optimal solutions.

**Acknowledgement:** I am indebted to Rainer Kolisch and Andreas Drexl for making the source code of their heuristic available.

## References

[1] BACKHAUS, K.; B. ERICHSON, W. PLINKE, AND R. WEIBER (1996): Multivariate Analysemethoden: Eine anwendungsorientierte Einführung. Springer, Berlin et al.

[2] BOCTOR, F.F. (1993): Heuristics for scheduling projects with resource restrictions and several resource-duration modes. International Journal of Production Research, Vol. 31, pp. 2547-2558.

[3] BOCTOR, F.F. (1994): A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes. European Journal of Operational Research, Vol. 90, pp. 349-361.

[4] BOCTOR, F.F. (1994): An adaption of the simulated annealing algorithm for solving resource-constrained project scheduling problems. International Journal of Production Research (to appear).

[5] DORNDORF, U. AND E. PESCH (1995): Evolution based learning in a job shop scheduling environment. Computers & Operations Research, Vol. 22, pp. 25-40.

[6] DREXL, A. AND J. GRÜNEWALD (1993): Nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions, Vol. 25, No. 5, pp. 74-81.

[7] EIBEN, A.E., E.H.L. AARTS, AND K.M. VAN HEE (1990): Global convergence of genetic algorithms: A Markov chain analysis. Lecture Notes in Computer Science, Vol. 496, pp. 4-12.

[8] ELMAGHRABY, S.E. (1977): Activity networks: Project planning and control by network models. Wiley, New York.

[9] GOLDBERG, D.E. (1989): Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, Massachusetts.

[10] GREFENSTETTE, J.J. (1987): Incorporating problem specific knowledge into genetic algorithms. In: Davis, L. (Ed.): Genetic algorithms and simulated annealing, pp. 42-60. Pitman, London.

[11] HARTMANN, S. AND A. DREXL (1997): Project scheduling with multiple modes: A comparison of exact algorithms. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 430, University of Kiel, Germany.

[12] HERRMANN, J.W., C.-Y. LEE, AND J. HINCHMAN (1995): Global job shop scheduling with a genetic algorithm. Production and Operations Management, Vol. 4, No. 1, pp. 30-45.

[13] HOLLAND, H.J. (1975): Adaption in natural and artificial systems. Univeristy of Michigan Press, Ann Arbor.

[14] KOHLMORGEN, U., H. SCHMECK, AND K. HAASE (1996): Experiences with fine-grained parallel genetic algorithms. Working Paper, University of Karlsruhe.

[15] KOLISCH, R. AND A. DREXL (1997): Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions (to appear).

[16] KOLISCH, R.; A. SPRECHER AND A. DREXL (1995): Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science, Vol. 41, pp. 1693-1703.

[17] KOLISCH, R. AND A. SPRECHER (1996): PSPLIB - A project scheduling problem library. European Journal of Operational Research, Vol. 96, pp. 205-216.

[18] LEE, J.-K. AND Y.-D. KIM (1996): Search heuristics for resource-constrained project scheduling. Journal of the Operational Research Society, Vol. 47, pp. 678-689.

[19] MATTFELD, D.C. (1996): Evolutinary search and the job shop. Physica, Heidelberg.

[20] ÖZDAMAR, L. (1996): A genetic algorithm approach to a general category project scheduling problem. Research Report, Marmara University, Istanbul.

[21] REEVES, C.R. (1995): Genetic algorithms and combinatorial optimization. In: Rayward-Smith, V.J. (Ed.): Applications of modern heuristic methods. Alfred Waller Ltd., Henley-on-Thames.

[22] SLOWINSKI, R. (1980): Two approaches to problems of resource allocation among project activities: A comparative study. Journal of the Operational Research Society, Vol. 31, pp. 711-723.

[23] SLOWINSKI, R., B. SONIEWICKI, AND J. WEGLARZ (1994): DSS for multiobjective project scheduling subject to multiple-category resource constraints. European Journal of Operational Research, Vol. 79, pp. 220-229.

[24] SPERANZA, M.G. AND C. VERCELLIS (1993): Hierarchical models for multi-project planning and scheduling. European Journal of Operational Research, Vol. 64, pp. 312-325.

[25] 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 et al.

[26] SPRECHER, A. AND A. DREXL (1996): Solving multi-mode resource-constrained project scheduling poblems by a simple, general and powerful sequencing algorithm. Part I: Theory. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 385, University of Kiel, Germany.

[27] SPRECHER, A. AND A. DREXL (1996): Solving multi-mode resource-constrained project scheduling poblems by a simple, general and powerful sequencing algorithm. Part II: Computation. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 386, University of Kiel, Germany.

[28] SPRECHER, A.; R. KOLISCH AND A. DREXL (1995): Semi-active, active and non-delay schedules for the resource-constrained project scheduling problem. European Journal of Operational Research, Vol. 80, pp. 94-102.

[29] SPRECHER, A.; S. HARTMANN AND A. DREXL (1997): An exact algorithm for project scheduling with multiple modes. OR Spektrum (to appear).

[30] TALBOT, F.B. (1982): Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. Management Science, Vol. 28, pp. 1197-1210.