Published in *Naval Research Logistics* 45:733-750 (1998). © Wiley & Sons,
doi:10.1002/(SICI)1520-6750(199810)45:7<733::AID-NAV5>3.0.CO;2-C

A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling

Sönke Hartmann*

Christian-Albrechts-Universität zu Kiel, Lehrstuhl für Produktion und Logistik, D-24098 Kiel, Germany. E-mail: hartmann@bwl.uni-kiel.de

*supported by the *Studienstiftung des deutschen Volkes*

Abstract. In this paper we consider the resource-constrained project scheduling problem (RCPSP) with makespan minimization as objective. We propose a new genetic algorithm approach to solve this problem. Subsequently, we compare it to two genetic algorithm concepts from the literature. While our approach makes use of a permutation based genetic encoding that contains problem-specific knowledge, the other two procedures employ a priority value based and a priority rule based representation, respectively. Then we present the results of our thorough computational study for which standard sets of project instances have been used. The outcome reveals that our procedure is the most promising genetic algorithm to solve the RCPSP. Finally, we show that our genetic algorithm yields better results than several heuristic procedures presented in the literature.

Keywords. Project Management and Scheduling, Resource-Constraints, Genetic Algorithms, 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. This problem arises within project management software as well as within systems for production planning and scheduling. The currently most powerful exact procedures have been presented by Brucker et al. [3], Demeulemeester and Herroelen [6, 7], Mingozzi et al. [24], and Sprecher [32]. However, they are unable to find optimal schedules for highly resource-constrained projects with 60 activities or more. Hence, in practice heuristic algorithms to generate near-optimal schedules for larger projects are of special interest. Recently, an evaluation study (see Kolisch and Hempel [18] and Kolisch [14]) showed that commercial software packages for project management generate schedules with an average deviation of 4.3% to 9.8% from the optimal solution for projects with up to 30 activities. These rather disappointing results indicate the need for (fast) heuristics to obtain better near-optimal schedules which should be implemented in software packages.

Recently proposed heuristic algorithms for the RCPSP include the following: Bell and Han [2] present a two-phase algorithm based on the concept of disjunctive arcs. Sampson and Weiss [31] introduce a local search approach in which a schedule is encoded by a shift vector. Özdamar and Ulusoy [26] propose the so-called local constraint based analysis (LCBA), which is extended to an iterative algorithm in Özdamar and Ulusoy [28]. Kolisch [15] develops three new priority rules for the parallel scheduling scheme and tests them in a single-pass environment. Kolisch [16] compares some good priority rules within both the serial and the parallel scheduling scheme and reports experiences with sampling procedures. Kolisch and Drexl [17] introduce a so-called adaptive search procedure which applies a scheduling scheme and priority rules after analyzing the project instance at hand. Baar et al. [1] develop two tabu search approaches. Lee and Kim [21] compare a genetic algorithm (GA), a simulated annealing heuristic, and a tabu search method. Cho and Kim [4] improve the simulated annealing procedure of Lee and Kim [21]. Leon and Ramamoorthy [22] suggest a GA and two local search procedures which are based on the so-called problem space based representation. Kohlmorgen et al. [13] summarize their experiences with an implementation of a GA for the RCPSP on parallel processors.

The purpose of this paper is to introduce a new GA approach for solving the RCPSP and to compare it to two existing GA concepts for this problem class. The three procedures are based on different genetic encodings and encoding-specific genetic operators. We proceed as follows: Section 2 provides a description of the RCPSP. Section 3 introduces the new GA which makes use of a permutation based representation. Sections 4 and 5 summarize two GA approaches from the literature which employ a priority value and a priority rule based encoding, respectively. Section 6 gives the results of our in-depth computational experiments. Finally, Section 7 states some conclusions.

2 Problem Description

We consider a project which consists of J activities (jobs) labeled $j = 1, \dots, J$. Due to technological requirements, 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. Analogously, S_j is the set of the immediate successors of activity j. The transitive closure of the precedence relations is given by sets of (not necessarily immediate) successors $\overline{S}_j$. 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 single source and $j = J + 1$ representing the single sink activity of the network.

With the exception of the (dummy) source and (dummy) sink activity, each activity requires certain amounts of (renewable) resources to be performed. The set of resources is referred to as K. For each resource $k \in K$ the per-period-availability is constant and given by R_k. The processing time (duration) of an activity j is denoted as p_j, its request for resource k is given by r_{jk}. Once started, an activity may not be interrupted. W.l.o.g., we assume that the dummy source and the dummy sink activity have a duration of zero periods and no request for any resource.

The parameters are assumed to be nonnegative and integer valued. The objective is to determine a schedule with minimal makespan such that both the precedence and resource constraints are fulfilled. Mathematical programming formulations of the RCPSP have been given by e.g. Demeulemeester and Herroelen [6, 7], Mingozzi et al. [24], and Sprecher [32].

3 Permutation based Genetic Algorithm

3.1 Basic Scheme

Introduced by Holland [12], genetic algorithms (GAs) serve as a heuristic meta strategy to solve hard optimization problems. For an introduction into GAs, we refer to Goldberg [9]. In this section we propose a new GA approach for the RCPSP.

The GA starts by computing an initial population, i.e. the first generation. We assume that the initial population contains POP individuals where POP is an even integer. After computing the fitness values of the individuals, 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 determining the fitness of each child individual, we add the children to

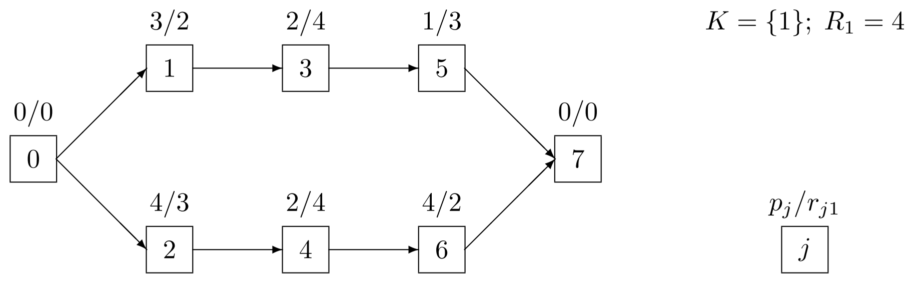

Figure 1: Project instance

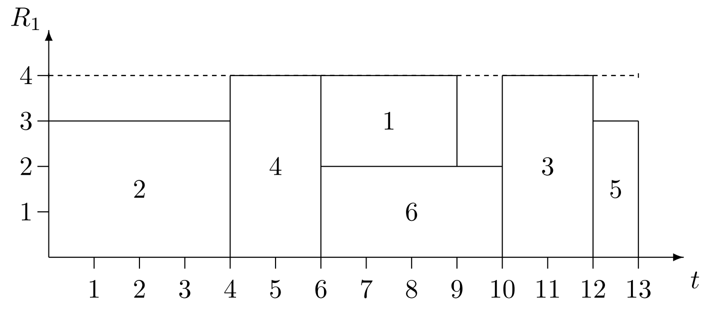

Figure 2: Example schedule

the current population, leading to a population size of $2 \cdot POP$. Finally, applying the selection operator to reduce the population to its former size, we obtain the next generation. The algorithm stops if a prespecified number of generations which is denoted as GEN or a given time limit is reached. Clearly, at most $POP \cdot GEN$ different individuals (and related schedules) are calculated.

3.2 Individuals and fitness

In the first GA variant to be examined, an individual I is given by an activity sequence

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

This job sequence 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 $P_{j_i^I} \subseteq \{0, j_1^I, \ldots, j_{i-1}^I\}$ for $i = 1, \ldots, J$.

Each genotype is related to a uniquely determined schedule (phenotype) which is computed using the following serial scheduling scheme: 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, each activity is assigned the earliest feasible start time. Note that the result is an active schedule, cf. Kolisch [16]. That is, no activity can be left shifted without violating the constraints (for a formal definition of active schedules cf. Sprecher et al. [33]). The fitness of an individual I is given by the makespan of the related schedule. Consider the project instance shown in Figure 1. Applying the serial scheduling procedure described above to example individual

$$I = (2, 4, 6, 1, 3, 5)$$

leads to the schedule given in Figure 2. The fitness of this individual is 13.

The initial population is computed as follows: Starting with the empty job sequence, we obtain a precedence feasible job sequence by repeatedly applying the following step: The next activity in the job sequence is randomly taken from the set of those currently unselected activities all non-dummy predecessors of which have already been selected for the job sequence. In addition to this random algorithm, we have tested another variant which determines the initial population with a sampling procedure as described by Kolisch [16]. More precisely, we employ a good priority rule, in our case the latest finish time rule (LFT), to derive probabilities which are used to select the next activity for the job sequence.

Notice that, while each individual is related to a unique schedule, a schedule can be related to more than one individual. In other words, there is some redundancy in the search space as distinct elements of the search space (i.e. genotypes) may be related to the same schedule. Consider again the project instance of Figure 1 and the above example individual I. Obviously, exchanging activities 1 and 6 in this job sequence, we obtain a different precedence feasible genotype, i. e. $(2,4,1,6,3,5)$. However, both genotypes are related to the same schedule, i.e. the schedule displayed in Figure 2.

3.3 Crossover

We consider three different crossover variants for the permutation based encoding. They are similar to the general crossover technique described by Reeves [30] for permutation based genotypes, with the difference that our encoding takes precedence relations into account.

The first crossover operator is called **one-point crossover**. We consider two individuals selected for crossover, a mother M and a father F. Then we draw a random integer q with $1 \le q < J$. 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 job sequence of D, the positions $i = 1, \ldots, q$ are taken from the mother, that is,

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

The job sequence of positions $i = q+1, \ldots, J$ in D is taken from the father. However, the jobs that have already been taken from the mother may not be considered again. We obtain

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

As a result, the relative positions in the parents' job sequences are preserved. This way, the parents contribute to the child's fitness. For illustration, we consider again Figure 1 and, given $q = 3$, the individuals

$$M = (1,3,2,5,4,6),\ F = (2,4,6,1,3,5),\ D = (1,3,2,4,6,5).$$

While the above definition obviously ensures that each activity appears exactly once in the resulting job sequence, the following theorem shows that also the precedence assumption is fulfilled.

Theorem 1 *If applied to precedence feasible parent individuals, the one-point crossover operator for the permutation based genetic encoding results in a precedence feasible offspring genotype.*

Proof. Let the genotypes of the parents M and F fulfill the precedence assumption. We assume that the child individual D produced by the crossover operator is not precedence feasible. That is, there are two activities j_i^D and j_k^D with $1 \le i < k \le J$ and $j_k^D \in P_{j_i^D}$. Three cases can be distinguished:

Case 1: We have $i, k \le q$. Then activity j_i^D is before activity j_k^D in the job sequence of M, a contradiction to the precedence feasibility of M.

Case 2: We have $i, k > q$. As the relative positions are maintained by the crossover operator, activity j_i^D is before activity j_k^D in the job sequence of F, contradicting the precedence feasibility of F.

Case 3: We have $i \le q$ and $k > q$. Then activity j_i^D is before activity j_k^D in the job sequence of M, again a contradiction to the precedence feasibility of M. $\square$

The son S of the individuals M and F is computed similarly. However, the positions $1, \ldots, q$ of the son's job sequence are taken from the father and the remaining positions are determined by the mother. Obviously, Theorem 1 holds for both offspring individuals D and S.

The second crossover operator is an extension of the one-point variant, called **two-point crossover**. Here, we draw two random integers q_1 and q_2 with $1 \leq q_1 < q_2 \leq J$. Now the daughter individual D is determined by taking the job sequence of the positions $i = 1, \ldots, q_1$ from the mother, that is,

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

The positions $i = q_1 + 1, \ldots, q_2$ are derived from the father:

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

The remaining positions $i = q_2 + 1, \ldots, J$ are again taken from the mother, that is,

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

Considering again Figure 1 and the above example parents M and F, we obtain for $q_1 = 1$ and $q_2 = 3$ daughter

$$D = (1, 2, 4, 3, 5, 6).$$

The son individual is computed analogously, taking the first and third part from the father and the second one from the mother. Obviously, Theorem 1 can easily be extended to the two-point crossover. Observe also that fixing $q_2 = J$ leads to the one-point variant which therefore is a special case of the two-point crossover.

The third crossover type is called **uniform crossover**. Here, the daughter D is determined as follows: We draw a sequence of random numbers $p_i \in \{0, 1\}$, $i = 1, \ldots, J$. Then we successively fill positions $i = 1, \ldots, J$ in D. If we have $p_i = 1$, we take that activity from the mother's job sequence which has the lowest index among the currently unselected activities, that is,

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

Otherwise, if $p_i = 0$, the activity is analogously derived from the father's job sequence:

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

For the above example individuals M and F and random number sequence $0, 1, 1, 0, 1, 1$, we obtain daughter

$$D = (2, 1, 3, 4, 5, 6).$$

The son S is computed using an analogous procedure which takes the i-th job from the father if $p_1 = 1$ and from the mother otherwise. Note that the uniform crossover generalizes the two-point variant: Fixing $p_i = 1$ for $i \in \{1, \ldots, q_1, q_2 + 1, \ldots, J\}$ and $p_i = 0$ for $i \in \{q_1 + 1, \ldots, q_2\}$ leads to the definition of the daughter in the two-point crossover. With arguments similar to those used in the proof of Theorem 1, one can show that also the uniform crossover produces precedence feasible offspring.

3.4 Mutation

Given a permutation based individual I, the mutation operator modifies the related job sequence as follows: For all positions $i = 1, \ldots, J-1$, activities j_i^I and j_{i+1}^I are exchanged with a probability of p_{mutation}, if the result is a job sequence which fulfills the precedence assumption.

The mutation operator may create job sequences (i.e. gene combinations) that could not have been produced by the crossover operator. However, 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: For example, interchanging two activities in the job sequence which have the same start time changes the individual, but not the related schedule.

3.5 Selection

We consider four alternative selection operators which follow a survival-of-the-fittest strategy as similarly described by e.g. Michalewicz [23]. The first one is a simple **ranking method**: We keep the POP best individuals and remove the remaining ones from the population (ties are broken arbitrarily).

The second variant, the **proportional selection**, can be viewed as a randomized version of the previously described ranking technique. Let $f(I)$ be the fitness of an individual I, and 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 successively 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

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

Next, we consider two versions of the tournament technique. In the **2-tournament selection**, two different randomly chosen individuals I_1 and I_2 compete for (temporary) survival. If individual I_1 is not better than individual I_2, i.e. $f(I_1) \geq f(I_2)$, then it dies and is removed from the population (again, ties are broken arbitrarily). This process is repeated until POP individuals are left. Recall that a lower fitness value implies a better quality of the individual.

Finally, the **3-tournament selection** extends the previously described method by randomly selecting three individuals I_1, I_2, and I_3. If we have $f(I_1) \geq f(I_2)$ and $f(I_1) \geq f(I_3)$, individual I_1 is removed from the population. Again, this step is repeated until POP individuals are left.

4 Priority Value based Genetic Algorithm

This section is devoted to a GA which makes use of a priority value based encoding as similarly used by Lee and Kim [21] within their GA for the RCPSP. We employ the priority value representation into the basic GA scheme that was used for the permutation based GA. This allows us to obtain comparable results when evaluating the GA approaches. We also make use of the selection operator variants here that have been defined for the permutation based GA as they are not encoding specific. In the following, the priority value based representation and the related crossover and mutation operators are discussed.

4.1 Individuals and fitness

In this GA approach, an individual I is represented by a sequence of priority values

$$I = (pv_1^I, \ldots, pv_J^I).$$

For each priority value of activity $j = 1, \ldots, J$, we have $pv_j^I \in [0,1]$. This encoding equals the one employed by Lee and Kim [21]. In contrast to their approach which employs a parallel scheduling scheme to derive the schedule related to an individual, however, we employ a serial scheme similar to that used for the permutation based encoding. We do so because it has been shown by Kolisch [16] that a serial scheme yields better results when a large number of schedules is computed for one project instance. This is due to the fact that using the parallel scheme, one might exclude all optimal solutions from the search space while the search space of the serial scheme always contains an optimal schedule.

Hence, given an individual I, the related schedule is computed as follows: After starting the dummy source activity at time 0, we determine the set EJ of the eligible activities, that is, those activities the predecessors of which are already scheduled. Then we schedule the eligible activity

j with the highest priority value $pv_j^I = \max \{pv_i^I \mid i \in EJ\}$ as early as possible such that neither the precedence nor the resource constraints are violated. Repeatedly scheduling an unscheduled activity, we obtain a feasible active schedule. Again, the fitness of an individual is defined as the makespan of the related schedule. Consider again the instance of Figure 1. Example individual

$$I = (0.58, 0.64, 0.31, 0.87, 0.09, 0.34)$$

is related to the schedule of Figure 2.

Each individual I of the initial population is determined by randomly drawing a priority value $pv_j^I \in [0, 1]$ with a uniform distribution for each activity $j = 1, \ldots, J$.

As was the case for the permutation based encoding, there is some redundancy in the search space also for the priority value based representation. We consider again the above example individual I. Clearly, setting for example $pv_2^I = 0.93$ instead of 0.64, we obtain a different individual. However, both individuals are related to the same schedule, namely the one of Figure 2.

4.2 Crossover

This encoding allows us to employ standard crossover operators. Again, we consider two individuals selected for crossover, a mother M and a father F, from which two offspring individuals are computed. In the following, we only define the daughter D. As for the permutation based representation, the son S is computed analogously to the daughter's definition.

For the **one-point crossover**, we draw a random integer q with $1 \leq q < J$. The first q positions of daughter individual D are taken from the mother while the remaining ones are defined by the father, that is, for each $i = 1, \ldots, J$ we have

$$pv_i^D = \begin{cases} pv_i^M, & \text{if } i \in \{1, \ldots, q\} \\ pv_i^F, & \text{if } i \in \{q+1, \ldots, J\}. \end{cases}$$

Using again Figure 1 and $q = 3$, this definition is illustrated by

$$\begin{aligned} M &= (0.58, 0.64, 0.31, 0.87, 0.09, 0.34), \\ F &= (0.12, 0.43, 0.99, 0.65, 0.19, 0.22), \\ D &= (0.58, 0.64, 0.31, 0.65, 0.19, 0.22). \end{aligned}$$

Similarly to the permutation based encoding, we have considered the **two-point crossover** in addition to the one-point variant defined above. We draw two random integers q_1 and q_2 with $1 \leq q_1 < q_2 \leq J$ and obtain for each $i = 1, \ldots, J$

$$pv_i^D = \begin{cases} pv_i^M, & \text{if } i \in \{1, \ldots, q_1\} \\ pv_i^F, & \text{if } i \in \{q_1+1, \ldots, q_2\} \\ pv_i^M, & \text{if } i \in \{q_2+1, \ldots, J\}. \end{cases}$$

Finally, for the **uniform crossover**, we draw a sequence of random numbers $p_i \in \{0, 1\}$, $i = 1, \ldots, J$. Then we set for each $i = 1, \ldots, J$

$$pv_i^D = \begin{cases} pv_i^M, & \text{if } p_i = 1 \\ pv_i^F, & \text{otherwise.} \end{cases}$$

4.3 Mutation

The mutation for the priority value based encoding is defined as follows: Given an individual I, we modify the related priority value sequence as follows: For all positions $i = 1, \ldots, J$, a new priority value $pv_i^I \in [0, 1]$ is drawn with a probability of p_{mutation}. Clearly, the result is always a feasible priority value sequence. Obviously, the mutation operator may create priority values (i.e. genes) that did not occur in the population before. Again, however, performing a mutation on an individual does not necessarily change the related schedule due to the redundancy discussed above.

5 Priority Rule based Genetic Algorithm

In this section, we describe a GA based on a priority rule representation. This representation type has been developed by Dorndorf and Pesch [8] for the job shop scheduling problem. Özdamar [25] employed it for the multi-mode extension of the RCPSP. Like the priority value based one, also this GA employs the basic scheme and the selection operator variants that have been used for the permutation based GA. Next, the priority rule encoding and the related crossover and mutation operators are provided.

5.1 Individuals and fitness

In this GA variant, an individual I is given by a sequence of priority rules

$$I = (pr_1^I, \dots, pr_J^I)$$

where we have for each position $i = 1, \dots, J$

$$pr_i^I \in \{\text{LFT, LST, MTS, MSLK, WRUP, GRPW}\}.$$

Each of these six priority rules has been suggested in the literature and shown to produce good schedules for the RCPSP, we refer to the study recently performed by Kolisch [16]. Table 1 contains a brief mathematical definition for each priority rule. Here, LFT_j denotes the latest finish time of activity j. It can be determined using traditional backward recursion from an upper bound on the project's makespan. Finally, with f_j we denote the earliest precedence and resource feasible finish time of activity j in the current partial schedule. Within the WRUP rule which has been developed by Ulusoy and Özdamar [34], we have employed the weights that performed best in their study.

Whereas Özdamar [25] employs a parallel scheme to transform an individual into a schedule, we use a serial scheme for the same reason as the one given in the previous section. We proceed as follows: First, we start the dummy source activity at time 0. Then, we compute the set of the eligible activities and use priority rule pr_1^I to select the eligible activity with the highest priority. The selected activity is started at the earliest precedence and resource feasible time. After updating the eligible set, the second activity to be scheduled is selected using priority rule pr_2^I. Repeatedly scheduling an unscheduled activity, we obtain a feasible active schedule. Thereby, the i-th decision which activity to schedule next is made using the i-th priority rule in the sequence. In other words, the i-th activity to be scheduled is selected using rule pr_i^I. The fitness of an individual is again given by the makespan of the related schedule. Considering the project instance of Figure 1, the schedule related to the example individual

$$I = (\text{LST, GRPW, MTS, LST, MSLK, LFT})$$

is again the one of Figure 2.

Priority rule		formula	
LFT	latest finish time	min	LFT_j
LST	latest start time	min	$LFT_j - p_j$
MTS	most total successors	max	$\lvert\overline{S}_j\rvert$
MSLK	minimum slack	min	$LFT_j - f_j$
WRUP	weighted resource utilization and precedence	max	$0.7\,\lvert S_j\rvert + 0.3 \sum_{k \in K} r_{jk}/R_k$
GRPW	greatest rank positional weight	max	$p_j + \sum_{i \in S_j} p_i$

Table 1: Good priority rules

Each individual I of the initial generation is determined by randomly selecting one of the six priority rules for pr_i^I, $i = 1, \dots, J$.

Like the two previously described genetic representations, also the priority rule based encoding contains some redundancy. Consider again example individual I given above. Replacing the first priority rule in the sequence (LFT) with e.g. the GRPW rule, we obtain a different genotype which is, however, also related to the schedule of Figure 2.

While the search spaces for the other two encodings always contain an optimal solution, this is not the case for the priority rule based representation. For the sake of shortness, we do not give a detailed counterexample, mentioning only that in some cases none of the employed priority rules may select an activity that must be scheduled next in order to obtain an optimal schedule from the current partial schedule. This drawback could be overcome by adding a priority rule which allows any eligible activity to be selected, namely the random rule (RAND). However, we did not employ the RAND priority rule because it seems to be incompatible with the GA paradigm: Obviously, selecting an activity with the RAND rule may contribute to the fitness if, by chance, this activity continues the current partial schedule in an advantageous way. However, the RAND rule itself does not contain specific information that is worth to be inherited.

5.2 Crossover

We can employ standard crossover operators similar to those used for the priority value encoding. The definitions of the one-point, two-point, and uniform crossover for the priority rule representation are obtained from replacing pv_i^I by pr_i^I in the respective definitions for the priority value encoding.

5.3 Mutation

For the priority rule based encoding, the mutation operator is defined as follows: For each position $i = 1, \dots, J$ of an individual I, a new priority rule

$$pr_i^I \in \{\text{LFT}, \text{LST}, \text{MTS}, \text{MSLK}, \text{WRUP}, \text{GRPW}\}$$

is randomly drawn with a probability of p_{mutation}. Again, due to the redundancy described above, performing a mutation on a genotype does not necessarily change the related schedule.

6 Computational Results

6.1 Experimental Design

In this section we present the results of the computational studies concerning the genetic algorithms introduced in the previous sections. The experiments have been performed on a Pentium-based IBM-compatible personal computer with 133 MHz clock-pulse and 32 MB RAM. For our study, we have implemented four procedures: the three GAs described in the previous sections and, as a benchmark, the priority rule based sampling approach of Kolisch [16]. These procedures have been coded in ANSI C, compiled with the GNU C compiler, and tested under Linux.

Two different standard sets of benchmark instances from the literature have been used. For configuration and analysis of the GA approaches (in Subsections 6.2, 6.3, and 6.4), we used a set of test problems constructed by the project generator ProGen developed by Kolisch et al. [20]. These instances 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 [19]. In our study, we have used the ProGen problem instances with 30 and 60 non-dummy activities. We have 4 renewable resources. For each problem size, a set consists of 480 instances which have been systematically generated by varying three parameters: network complexity, resource factor, and resource strength. The network complexity reflects the average number of immediate successors of an activity. The resource factor is a measure of the average number of resources requested per job. The resource strength describes the scarceness of the resource capacities. These parameters are known to have a big impact on the hardness of a project instance, cf. Kolisch et al. [20].

The set with 30 non-dummy activities currently is the hardest standard set of RCPSP-instances for which all optimal solutions are known, cf. Demeulemeester and Herroelen [7]. In what follows we report the average percentage deviation from the optimal makespan. However, for some of the instances with 60 activities, only heuristic solutions are known. In these cases, we give the average percentage deviation from the best lower and upper bounds as reported in the library PSPLIB at the time this research was performed. The lower bounds were computed by Baar et al. [1] and Heilmann and Schwindt [11]. The upper bounds were obtained by Kolisch and Drexl [17], Kohlmorgen et al. [13], and Baar et al. [1].

Finally, for comparing our GA approach with other procedures presented in the literature (in Subsection 6.5), we use the set of project instances assembled by Patterson [29]. We do so because the Patterson instance set has been used to evaluate heuristics in several studies, see [2, 4, 21, 22, 27, 28, 31]. The Patterson set contains 110 instances with up to 51 activities and up to 3 resources. For all of these test problems, the optimal solutions are known, see e.g. Demeulemeester and Herroelen [6].

6.2 Configuration of the Genetic Algorithms

Crucial for the success of a GA is an appropriate choice of good genetic operators and adequate parameter settings, usually on the basis of computational experiments. We report the outcome of our study to determine the best GA configuration only for the suggested permutation based GA approach of Section 3 and for the ProGen instance set with 30 non-dummy activities, because the results for the other GAs and the other instance sets are similar. For each instance, 1,000 schedules were computed.

Table 2 reports the average and the maximal deviation from the optimum, the percentage of instances for which an optimal solution was found, and the computation time in seconds for different combinations of alternative genetic operators. We have tested all possible GA configurations that arise from combining three mutation probability levels, the three crossover variants, and the four selection variants. A mutation probability of 0.05, the two-point crossover strategy, and the ranking method for selection perform best (for the sake of shortness, Table 2 contains only those configurations that vary this best one in only one point). The two-point crossover operator appears to be capable of inheriting building blocks that contributed to the parents' fitness (for much larger projects, even more than two cuts may probably be advisable). In contrast, the uniform crossover operator (which yields good results for problems with a different structure such as e.g. the multidimensional knapsack problem, cf. Chu and Beasley [5]) does not seem to be well suited for sequencing problems. A randomized selection strategy seems to be advantageous only if a much larger number of individuals is considered.

Table 3 shows that a population size of 40 and 25 generations is the best parameter relationship when calculating 1,000 individuals (i.e. schedules). Finally, Table 4 shows that it pays to use a sampling method instead of a pure random procedure to determine the initial population.

In the further computational studies, the three GAs make use of the best configuration determined here. That is, they apply a mutation probability of 0.05, a two-point crossover, the ranking selection, and the relationship of population size and number of generations given above. Recall, however, that the sampling procedure to determine the initial population is only employed in our permutation based GA. The other two approaches make use of a pure random procedure.

6.3 Comparison of the Genetic Algorithms

In this subsection, we present the experimental results obtained from the comparison of the proposed GA of Section 3 with the GA approaches described in Sections 4 and 5.

p_{mutation}	crossover	selection	average dev.	max. dev.	optimal	CPU-sec
0.01	two-point	ranking	0.64%	9.7%	77.5%	0.54
0.05	two-point	ranking	0.54%	7.9%	81.5%	0.54
0.10	two-point	ranking	0.56%	8.6%	80.4%	0.55
0.05	one-point	ranking	0.65%	9.7%	77.5%	0.54
0.05	two-point	ranking	0.54%	7.9%	81.5%	0.54
0.05	uniform	ranking	0.66%	8.6%	79.6%	0.76
0.05	two-point	ranking	0.54%	7.9%	81.5%	0.54
0.05	two-point	proportional	0.62%	7.7%	78.5%	0.60
0.05	two-point	2-tournament	0.63%	9.3%	79.0%	0.54
0.05	two-point	3-tournament	0.59%	7.3%	80.9%	0.54

Table 2: Alternative genetic operators — permutation based GA, 1,000 schedules, $J = 30$

POP	*GEN*	average dev.	max. dev.	optimal	CPU-sec
50	20	0.56%	11.1%	80.8%	0.54
40	25	0.54%	7.9%	81.5%	0.54
20	50	0.79%	9.2%	76.5%	0.54

Table 3: Impact of population size — permutation based GA, 1,000 schedules, $J = 30$

The first numerical results to be presented are obtained from the ProGen project instance set with 30 activities where 1,000 individuals (i.e. schedules) are computed for each instance. Table 5 gives the average and the maximal deviation from the optimum, the percentage of instances for which an optimal solution was found, and the computation time in seconds. The permutation based encoding yields better results than the priority value based representation while both outperform the priority rule encoding.

Table 5 also shows that the permutation based GA results in the lowest computation times. This is because we have to determine eligible activities and apply priority rules only for the first generation, when computing the schedule related to permutation based individuals. Clearly, when using a heuristic procedure in practice, it is important to obtain good schedules within a reasonable amount of CPU time. Therefore we have additionally tested the three GA variants with time limits instead of fixing the number of schedules to be computed. As a further benchmark, we use a good sampling procedure known from the literature (cf. Kolisch [16]) which is based on the randomized LFT priority rule.

Table 6 displays the average deviations from the optimum obtained for the instances with $J = 30$ from four different time limits. The permutation based GA performs best for all time limits while the priority rule based GA yields the worst results. Note especially that the deviation of the permutation based GA is two times lower than that of the priority rule based GA for a time limit of 0.5 seconds while it is more than four times lower for 4 seconds. That is, the proposed GA is

Initial population	average dev.	max. dev.	optimal	CPU-sec
random sampling (LFT)	0.54%	7.9%	81.5%	0.54
random	0.99%	10.5%	70.8%	0.52

Table 4: Impact of initial population — permutation based GA, 1,000 schedules, $J = 30$

GA	average dev.	max. dev.	optimal	CPU-sec
permutation	0.54%	7.9%	81.5%	0.54
priority value	1.03%	10.8%	70.6%	0.64
priority rule	1.38%	17.7%	70.6%	0.91

Table 5: Comparison of genetic algorithms — 1,000 schedules, $J = 30$

Algorithm	type	0.5 sec	1.0 sec	2.0 sec	4.0 sec
GA	permutation	0.71%	0.45%	0.37%	0.24%
GA	priority value	1.16%	0.88%	0.69%	0.54%
GA	priority rule	1.51%	1.33%	1.21%	1.13%
sampling	LFT	1.00%	0.77%	0.66%	0.55%

Table 6: Average deviations w.r.t. time limit — $J = 30$

not only the best for small time limits, its superiority also further increases when the computation time is increased.

Next, we have performed the same experiment on the set of instances with 60 activities. As for some instances optimal solutions are currently unknown, we measure the deviations from the best known lower and upper bound here. The results can be found in Tables 7 and 8, respectively. Again, the permutation based GA performs best for all time limits. In contrast to the instances with $J = 30$, however, here the priority rule based GA outperforms the priority value GA and the sampling approach. This observation can be explained as follows: Selecting $J = 60$ instead of $J = 30$ results in a much larger search space. Within the same time limit, only a much smaller portion of the search space can be examined. Therefore, the strategy to combine several good priority rules corresponds to examining only potentially promising regions of the search space. However, the restriction to the regions identified by the priority rules is disadvantageous for smaller projects and/or higher computation times (or, of course, faster computers).

We remark here that most of the best known upper bounds for the hard instances were computed by Kohlmorgen et al. [13] with their parallel GA approach. Using a massively parallel computer with 16k processing units, they had 16,384 individuals per generation, while our GA could not evaluate more than 4,000 individuals altogether within 4 seconds when applied to the instances with 60 activities. Thus, it is not surprising that the best known upper bounds are on the average 0.59% better than our results (cf. Table 8). Nevertheless, it should be mentioned that the schedules for 3 of the 480 instances with 60 activities found by our GA (with a time limit of 4 seconds) were better than those reported in the library PSPLIB at the time this research was performed.

The results can be summarized as follows: The permutation based GA outperforms the other two GA approaches. Considering again Subsection 6.2, we observe that the choice of an appropriate representation is far more important than other configuration decisions such as crossover and

Algorithm	type	0.5 sec	1.0 sec	2.0 sec	4.0 sec
GA	permutation	5.30%	4.73%	4.37%	4.16%
GA	priority value	7.60%	6.60%	6.17%	5.70%
GA	priority rule	5.92%	5.50%	5.18%	4.96%
sampling	LFT	6.08%	5.77%	5.63%	5.39%

Table 7: Average deviations from best lower bound w.r.t. time limit — $J = 60$

Algorithm	type	0.5 sec	1.0 sec	2.0 sec	4.0 sec
GA	permutation	1.42%	1.10%	0.79%	0.59%
GA	priority value	3.61%	2.88%	2.24%	1.79%
GA	priority rule	2.07%	1.80%	1.52%	1.34%
sampling	LFT	2.27%	1.99%	1.82%	1.62%

Table 8: Average deviations from best upper bound w.r.t. time limit — $J = 60$

Encoding	GA	random
permutation	0.54%	0.82%
priority value	1.03%	1.69%
priority rule	1.38%	1.41%

Table 9: Impact of genetic operators — 1,000 schedules, $J = 30$

selection type or mutation rate.

6.4 Impact of Genetic Operators

In what follows, we examine the question whether the three representations are well suited for application in genetic algorithms. In other words, we want to know if they benefit from the application of genetic operators. Thus, for each encoding, we compare the related GA which computes 1,000 schedules by applying the genetic operators on 40 individuals over 25 generations with what we will call a "random procedure". For each encoding, this random procedure generates 1,000 schedules independently, that is, without applying the genetic operators. Note that it can be viewed as a GA with a population size of 1,000 and only one generation (i.e. the initial one).

For the three encodings, the average deviations from the optimal solutions for both the "real" GAs and the random procedures are listed in Table 9. We observe that the GAs clearly outperform the corresponding random procedures if the permutation and priority value encodings are considered. However, for the priority rule encoding, there are only slight differences between randomly generating 1,000 priority rule sequences (and thus schedules) on the one hand and randomly generating only 40 and applying the genetic operators over 25 generations on the other hand. Hence we can deduce that the priority rule encoding does not exploit the potential of the GA paradigm when applied to solve the RCPSP. In contrast, the other two representations are well suited for application in a GA.

Recall that the initial population of the permutation based GA is computed using a sampling method. That is, the random procedure for the permutation encoding listed in Table 9 is in fact a sampling algorithm. Table 9 shows that it is better to determine only 40 schedules using the sampling procedure and then apply the genetic operators over 25 generations, than to compute all 1,000 schedules with the sampling method.

6.5 Comparison with other heuristics

Finally, we have compared our permutation based GA to further heuristics suggested in the literature. The results are shown in Table 10. Recall, the priority value and the priority rule based GAs can be viewed as modified versions of the GAs proposed by Lee and Kim [21] and Özdamar [25], respectively. In addition to the priority rule based sampling algorithm of Kolisch [16], we have considered the simulated annealing (SA) procedures of Cho and Kim [4] and Lee and Kim [21], the local search approach of Sampson and Weiss [31], and the disjunctive arc based two-phase

Algorithm	Reference	average dev.	optimal	CPU-sec
GA (permutation)	—	0.00%	100.0%	5.0[a]
GA (priority value)	—	0.25%	88.4%	5.0[a]
GA (priority rule)	—	0.78%	74.6%	5.0[a]
sampling (LFT)	Kolisch [16]	0.05%	96.4%	5.0[a]
SA	Cho, Kim [4]	0.14%	93.6%	18.4[b]
SA	Lee, Kim [21]	0.57%	82.7%	17.0[b]
local search	Sampson, Weiss [31]	1.98%	55.5%	10.2[b]
two-phase	Bell, Han [2]	2.60%	44.5%	28.4[c]
GA (problem space)	Leon, Ramamoorthy [22]	0.74%	75.5%	7.5[d]
LCBA	Özdamar, Ulusoy [28]	1.14%	63.6%	0–25[e]

[a]maximal CPU-time on a Pentium 133 MHz
[b]average CPU-time on a Pentium 60 MHz
[c]average CPU-time on a Macintosh plus
[d]average CPU-time on an IBM RS 6000
[e]CPU-time range on an IBM PC 486

Table 10: Comparison of heuristics — Patterson instance set

procedure of Bell and Han [2] as further benchmarks. The results for the latter four heuristics are given as reported in [4]. Furthermore, the local constraint based analysis (LCBA) method of Özdamar and Ulusoy [26] has been included in its iterative variant, see Özdamar and Ulusoy [28]. The results cited here are taken from the study performed by Özdamar and Ulusoy [27, 28]. Finally, we have considered the results obtained by Leon and Ramamoorthy [22] for their GA. Their approach is based on the so-called problem space based representation which, similarly to the priority value representation, encodes a solution using an array of real numbers.

Taking the different computers and computation times into account, we can state that the permutation based GA is the most promising among the these heuristics for the RCPSP. Observe that our GA solves all Patterson instances to optimality within a time limit of 5 seconds. Finally, the results show that the success of a GA mainly depends on the underlying representation.

7 Conclusions

We have proposed a genetic algorithm (GA) for solving the classical resource-constrained project scheduling problem (RCPSP). The representation is based on a precedence feasible permutation of the set of the activities. The genotypes are transformed into schedules using a serial scheduling scheme. Among several alternative genetic operators for the permutation encoding, we chose a ranking selection strategy, a mutation probability of 0.05, and a two-point crossover operator which preserves precedence feasibility. The initial population was determined with a randomized priority rule method. In order to evaluate our approach, we have compared it to two GA concepts from the literature which make use of a priority value and a priority rule representation, respectively. As further benchmarks, seven other heuristics known from the literature were considered. Our in-depth computational study revealed that our GA outperformed the other GAs as well as the other approaches. This outcome suggests to apply our GA to real-world project scheduling problems. In fact, we have obtained promising results for a medical research project documented in [10].

References

[1] T. Baar, P. Brucker, and S. Knust. Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics: Advances and trends in local search paradigms for optimization*, pages 1–8. Kluwer Academic Publishers, 1998.

[2] C. E. Bell and J. Han. A new heuristic solution method in resource-constrained project scheduling. *Naval Research Logistics*, 38:315–331, 1991.

[3] P. Brucker, S. Knust, A. Schoo, and O. Thiele. A branch-and-bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107:272–288, 1998.

[4] J. H. Cho and Y. D. Kim. A simulated annealing algorithm for resource-constrained project scheduling problems. *Journal of the Operational Research Society*, 48:736–744, 1997.

[5] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. Technical report, The Management School, Imperial College, London, England, 1997.

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

[7] E. L. Demeulemeester and W. S. Herroelen. New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43:1485–1492, 1997.

[8] U. Dorndorf and E. Pesch. Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, 22:25–40, 1995.

[9] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[10] S. Hartmann. Scheduling medical research experiments — An application of project scheduling methods. Manuskripte aus den Instituten für Betriebswirtschaftslehre 452, Universität Kiel, Germany, 1997.

[11] R. Heilmann and C. Schwindt. Lower bounds for RCPSP/max. Technical Report WIOR-511, Universität Karlsruhe, Germany, 1997.

[12] H. J. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, 1975.

[13] U. Kohlmorgen, H. Schmeck, and K. Haase. Experiences with fine-grained parallel genetic algorithms. *Annals of Operations Research*, 90:203–319, 1999.

[14] R. Kolisch. Resource allocation capabilities of commercial project management systems — Resource management boosts up the German stock exchange. *Interfaces.* Forthcoming.

[15] R. Kolisch. Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, 14:179–192, 1996.

[16] R. Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90:320–333, 1996.

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

[18] R. Kolisch and K. Hempel. Experimentelle Evaluation der Kapazitätsplanung von Projektmanagementsoftware. *Zeitschrift für betriebswirtschaftliche Forschung*, 48:999–1018, 1996.

[19] R. Kolisch and A. Sprecher. PSPLIB – a project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1996.

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

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

[22] V. J. Leon and B. Ramamoorthy. Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling. *OR Spektrum*, 17:173–182, 1995.

[23] Z. Michalewicz. Heuristic methods for evolutionary computation techniques. *Journal of Heuristics*, 1:177–206, 1995.

[24] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44:714–729, 1998.

[25] L. Özdamar. A genetic algorithm approach to a general category project scheduling problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 29:44–59, 1999.

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

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

[28] L. Özdamar and G. Ulusoy. An iterative local constraint based analysis for solving the resource-constrained project scheduling problem. *Journal of Operations Management*, 14:193–208, 1996.

[29] J. H. Patterson. A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management Science*, 30:854–867, 1984.

[30] C. R. Reeves. Genetic algorithms and combinatorial optimization. In V. J. Rayward-Smith, editor, *Applications of modern heuristic methods*, pages 111–125. Alfred Waller Ltd., Henley-on-Thames, 1995.

[31] S. E. Sampson and E. N. Weiss. Local search techniques for the generalized resource-constrained project scheduling problem. *Naval Research Logistics*, 40:665–675, 1993.

[32] A. Sprecher. Solving the RCPSP efficiently at modest memory requirements. Manuskripte aus den Instituten für Betriebswirtschaftslehre 425, Universität Kiel, Germany, 1996.

[33] A. Sprecher, R. Kolisch, and A. Drexl. Semi-active, active and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 80:94–102, 1995.

[34] G. Ulusoy and L. Özdamar. Heuristic performance and network/resource characteristics in resource-constrained project scheduling. *Journal of the Operational Research Society*, 40:1145–1152, 1989.