This is a preprint of an article published in C. Schwindt, J. Zimmermann (ed.), *Handbook on Project Management and Scheduling Vol. 1*, 163-176 (2015).

Time-Varying Resource Requirements and Capacities

Sönke Hartmann*

HSBA Hamburg School of Business Administration, Alter Wall 38, D-20457 Hamburg, Germany.
E-mail: soenke.hartmann@hsba.de

*supported by the HSBA Foundation

Abstract. This contribution discusses an extension of the classical resource-constrained project scheduling problem (RCPSP) in which the resource requests of the activities and the resource capacities may change over time. We present relationships to other variants of the RCPSP as well as some applications of this problem setting. Subsequently, we analyze the applicability of heuristics which were originally developed for the standard RCPSP and adapt one of them, a genetic algorithm, to this extension. The chapter closes with a few computational results and some remarks on research perspectives.

Keywords. Project scheduling, time-varying resource constraints, makespan minimization, genetic algorithm.

1 Introduction

In the classical resource-constrained project scheduling problem (RCPSP), a set of activities has to be scheduled such that precedence and resource constraints are met. That is, activities may not start before their predecessors have finished, and the resource requests may not exceed the capacities in any period. The goal is to determine a schedule with the shortest possible project duration.

The RCPSP is simple to describe but hard to solve, which has it made attractive for many researchers. One important direction of research has been the development of better solution methods (see Kolisch and Hartmann [13] for an overview of heuristics). Another main research direction has been the definition of alternative and more general problem settings (see Hartmann and Briskorn [8] for a survey).

Popular variants and extensions of the standard RCPSP include multiple execution modes for activities (Chapters 21 and 26 of this handbook), generalized precedence relations (Chapters 5, 6, and 7), and alternative objectives such as maximization of the net present value (Chapter 14). Also several approaches to generalize the resource constraints have been proposed. This includes alternative resource types such as storage resources (Chapter 9), continuous resources (Chapter 10), and partially renewable resources (Chapter 11).

In this contribution, we take a look at another generalization of the resource constraints. Resource requests and capacities are assumed to be constant over time in the standard RCPSP. This might be too restrictive for practical applications. Vacation of staff or planned maintenance of machines lead to capacities which vary over time. Also the resource request of an activity might not be constant during its processing time, depending on the actual application. Consequently, we consider an extension of the RCPSP in which resource availabilities are given for each period of the planning horizon, and resource demands are given for each period of an activity's duration. The resulting problem setting is referred to as RCPSP/t to indicate the time-dependency.

Resource capacities and requests varying with time have not yet gained much attention in the scientific literature. While the concept has been mentioned in a few papers (e.g., Bartusch et al. [1], Sprecher [18], de Reyck et al. [5]), the only dedicated approach is that of Hartmann [7]. There, a priority rule is developed for time-dependent capacities and requests and embedded into a general randomized scheduling framework. Tests based on a large set of test instances revealed that the priority rule performs marginally better than standard RCPSP rules such as the well-known latest start time rule (LST).

The goal of this contribution is twofold. First, we underscore the relevance of the RCPSP/t by pointing out to applications in medical research and aggregated production planning. Second, we discuss heuristics and their performance. We summarize findings concerning the applicability of heuristics that were designed for the standard RCPSP, and we exploit these results to adapt a genetic algorithm to the RCPSP/t.

2 Problem Setting

In this section we will give a more formal outline of the problem setting. We will also have a brief look at other project scheduling problems which are special cases of the problem discussed here or which include the problem discussed here as a special case, respectively.

2.1 Formal Problem Description

The RCPSP with time-varying resource requirements and capacities, the RCPSP/t, can be summarized as follows. As in the standard RCPSP, we consider n activities $1,\ldots,n$ with a processing time or duration p_j for each activity $j \in \{1,\ldots,n\}$. Once started, an activity may not be interrupted. An activity j may not be started before all its predecessor activities $i \in Pred(j)$ have finished, where $Pred(j)$ denotes the set of the immediate predecessors of activity j. Two additional activities 0 and $n+1$ are added. They are dummy activities with $p_0 = p_{n+1} = 0$ and reflect the start and the end of the project, respectively.

K resources are given. Resource k, $k = 1,\ldots,K$ has a capacity of $R_k(t)$ in period $t = 1,\ldots,T$, where T is the planning horizon. An activity j requires $r_{jk}(t)$ units of resource k in the t-th period of its duration, i.e., $t = 1,\ldots,p_j$. We usually assume the parameters such as processing times, capacities, and resource requests to be nonnegative and integer valued. The objective is to determine a schedule (i.e., a start time for each activity) with minimal total project duration (i.e., minimal makespan) such that both the temporal and the resource constraints are fulfilled.

A mathematical model can easily be obtained from incorporating the time-dependency of the resource parameters into the classical formulation of Pritsker et al. [17]. The resulting model can be found in Hartmann [7]. In the three-field classification scheme $\alpha|\beta|\gamma$ of Brucker et al. [3], the RCPSP/t would be denoted by $PSt|prec|C_{\max}$, where $\alpha = PSt$ stands for project scheduling with limited time-varying renewable resources.

2.2 Relationships to other Project Scheduling Problems

Obviously, the RCPSP/t generalizes the standard RCPSP. If all resources have constant capacities and if all activities have constant resource requests, we obtain the standard RCPSP.

Furthermore, the RCPSP/t is a special case of the RCPSP/max, i.e., the RCPSP with generalized precedence constraints. As outlined by Bartusch et al. [1], time-varying resource capacities can be transformed into constant ones by selecting the highest capacity and by defining a dummy activity for each drop in the capacity. Such a dummy activity is then fixed in time by adding a precedence relation with the source activity along with appropriate minimal and maximal time lags. Time-varying resource requests can be obtained in the RCPSP/max by splitting an activity whenever the request for a resource changes. The resulting sub-activities related to the original activity can now be stitched together using minimal and maximal time lags.

Finally, it should be mentioned that time-varying resource capacities are a special case of the so-called partially renewable resources (Böttcher et al. [2] and Chapter 11 of this handbook). A partially re-

newable resource k is associated with a set of period subsets $\Pi_k = \{P_{k1}, \ldots, P_{k\nu}\}$ where $P_{k\mu} \subseteq \{1, \ldots, T\}$ for each $\mu = \{1, \ldots, \nu\}$. For each period subset $P_{k\mu} \in \Pi_k$ there is a total resource capacity $R_k^p(P_{k\mu})$. That is, over the periods $t \in P_{k\mu}$, the total capacity of resource k is $R_k^p(P_{k\mu})$. Time-varying resource capacities can obviously be captured by defining one period subset $\{t\}$ for each period t of the planning horizon and by setting $R_k^p(\{t\}) := R_k(t)$.

3 Applications

The applicability of time-varying resource capacities and requests is straightforward. The capacity of a resource may vary due to vacations of staff or maintenance of machines. Resource requests of an activity are not necessarily constant over time in case of longer or more complex activities. Case studies on project scheduling problems including such a time-dependency can occasionally be found in the literature, see, e.g., Kolisch and Meyer [14]. This section reviews two specific applications of time-varying resource capacities and requests.

3.1 Medical Research Projects

As reported by Hartmann [7], the RCPSP/t can be applied to capture projects in medical research. Many projects in this field consist of experiments which require certain resources such as laboratory staff and equipment. The experiments can be viewed as activities which have to be scheduled such that an early end of the project is achieved. These characteristics suggest to apply project scheduling models like the RCPSP. A few properties of such medical research projects, however, go beyond the standard RCPSP, but they can easily be captured by the RCPSP/t.

The laboratory staff and the equipment often have capacities that vary over time. The staff might be absent due to vacation, or it might be present on certain weekends. Since often several projects are carried out in the same laboratory at the same time, the equipment might be available for a particular project only during certain weeks or on certain weekdays.

Moreover, the resource requests of the activities representing experiments need not be constant over time. Certain laboratory equipment might be required only on specific days, e.g., on the last day of an experiment. Also, laboratory staff might be required only on certain days of an experiment.

Finally, there are often specific requirements concerning the temporal arrangement of experiments. A typical constraint is that many repetitions of the same experiment are carried out in parallel because this makes it easier and more efficient to handle them. It also leads to a clearer schedule which helps to avoid mistakes when processing the experiments. On the other hand, it is important that not all repetitions are executed in parallel because otherwise possible mistakes in the handling of the experiments might not be detected and the results are distorted.

This can be captured by the RCPSP/t as follows: The repetitions of one experiment are grouped such that each group contains a number of repetitions that should be carried out in parallel. For each such group one activity is defined. For example, if an experiment must be repeated 30 times, one would define three activities corresponding to 10 repetitions each, given that always 10 repetitions should be carried out in parallel. To make sure that these three activities are not carried out in parallel in the sense that they do not start on the same day, we define a fictitious resource with a constant capacity of one unit. Each of the three activities requests one unit this additional resource in the first period of its duration (and no units in the remaing periods).

More details on modeling medical research projects using the RCPSP/t as well as a case study based on real data can be found in Hartmann [7].

3.2 Aggregated Production Scheduling

In what follows, we consider the case of a manufacturer of special machines located in Northern Germany. The company produces highly specialized machines in a make-to-order process. In addition to short-term planning at a detailed level, also long-term planning at an aggregated level is carried out.

The long-term planning approach takes all currently known orders into account. Each order corresponds to a machine of a particular type. While the production process of a machine is related to a project network with several activities, it is not necessary to consider all these individual activities in long-term planning.

Moreover, the only resources that need to be considered in long-term planning are the assembly areas. There are several different assembly areas, and each of these contains a number of so-called cells. The production of a machine type requires one or more cells from several assembly areas.

Each order is related to a deadline which must not be exceeded. The goal of aggregated planning is to determine a schedule in which the production of all ordered machines finishes as early as possible. The latter is of particular importance since it leads to free resource capacities for future orders.

This planning task can be captured as follows. We define one activity for each order. In other words, each project that is related to the production of one ordered machine is represented by a single activity. This activity is derived from the standard schedule that is associated with the production process of the related machine. Within this standard schedule, the start times are assumed to be fixed. The duration of an activity is defined as the total manufacturing time of the machine according to the standard schedule.

Obviously, each assembly area corresponds to one resource with a capacity given by the number of cells that are available. An activity requires some of the assembly areas, the request corresponds to the number of required cells. The request for an assembly area usually varies over time. For example, the first part of an activity might correspond to the manufacturing of certain parts in related assembly areas. The second part might correspond to the assembly of the machine using these parts. For this, another assembly area is needed whereas the assembly areas of the parts are no longer needed. This aggregation of activities to super-activities with time-varying resource requests has also been applied by Heimerl and Kolisch [10] in a similar way.

We are looking for a start time for each activity (which must observe the given deadline of each activity). That is, we are looking for the time at which the production related to an order should start such that the makespan is minimized. Summing up, we obtain the RCPSP/t with an additional deadline constraint.

4 Heuristics for the RCPSP/t

This section deals with heuristics for the RCPSP/t. We discuss in general how heuristics developed for the standard RCPSP can be applied. Subsequently, we consider a genetic algorithm which is extended by so-called delays in order to match the search space of the RCPSP/t.

4.1 Applicability of Heuristics Designed for the Standard RCPSP

Over the last decades, a large number of heuristics have been delevoped for the classical RCPSP, for overviews refer to Kolisch and Hartmann [12, 13]. Since the structure is very similar to the RCPSP/t, many of them can also be applied to the extended problem. One particular reason is that the so-called schedule-generation-schemes (SGS) which are the backbone of most RCPSP heuristics also work for the RCPSP/t.

Two main SGS are available for the standard RCPSP, the serial and the parallel one (Kolisch [11]). The serial SGS picks an activity in each step and schedules it at the earliest precedence and resource feasible start time. The parallel SGS on the other hand considers a point in time and successively picks an activity that can start at this time without violating the precedence and resource constraints. Whenever there is no activity left that can be feasibly started, the next point in time is selected. Note that the SGS only guides the scheduling process. In both SGS, the decision which activity to pick next is made by a priority rule or by a metaheuristic representation.

As shown by Sprecher et al. [19], the search space of the serial SGS always contains an optimal schedule for the RCPSP whereas the parallel one sometimes does not. Thus the parallel SGS might not be able to find an optimal solution for a given instance. This a drawback especially in small search spaces where the serial SGS is superior because it might be able to find an optimal solution. On the

other hand, the parallel SGS is superior when applied to large instances because it produces schedules of better average quality, which is an advantage in large search spaces.

Generally, both SGS can be applied to the RCPSP/t. However, their properties change for this problem class. There are some instances of the RCPSP/t for which the search space of the serial SGS does not contain an (existing) optimal solution. This has been shown by counterexample in Hartmann [7]. This counterexample indicates that one may have to start an activity later than at the earliest possible start time to find an optimal solution.

Another important component of many RCPSP heuristics is the so-called justification or forward-backward improvement approach of Tormos and Lova [20]. The idea is to improve a schedule as follows. In a first step, the schedule is scanned from right to left. Thereby, each activity is shifted to the right as far as possible, but not beyond the dummy sink activity. Next, the activities are scanned from left to right and shifted as far to the left as possible, but not before the new start time of the dummy source activity. Valls et al. [21] have demonstrated that this concept can be added to almost every heuristic and that it improves the results drastically.

This concept is not applicable when resource capacities are varying with time. Activities are shifted to the right to condense the project, but thereby also the dummy start activity is started later. In case of constant capacities, one can simply shift the entire project to the left and hence bring it back to the old start time while keeping the condensed schedule. The latter, however, is not possible if resource capacities vary over time.

Summing up, the SGS for the RCPSP are applicable to the RCPSP/t as well. This also holds for heuristics based on these SGS. Unfortunately, however, both SGS are unable to find an existing optimal solution for some instances of the RCPSP/t. Moreover, justification or forward-backward improvement is not applicable to the RCPSP/t. These issues must be considered when adapting heuristics from the RCPSP to the RCPSP/t.

4.2 An Adapted Genetic Algorithm

In this section, we adapt the genetic algorithm (GA) of Hartmann [6] to the RCPSP/t. This GA was originally proposed for the standard RCPSP. It is based on the so-called activity list representation in which the non-dummy activities are given in some order. This order must be precedence feasible, that is, an activity may not appear before any of its predecessors. The serial SGS is applied to determine a schedule for an activity list. It simply takes the activities in the order given by the activity list and schedules each one at the earliest precedence and resource feasible time. The activity lists for the first generation are constructed using a randomized priority rule, namely the latest finish time rule (LFT).

The crossover operator takes parts of the activity lists from the mother and the father and combines them to form a child. We apply a two-point crossover for which two positions q_1 and q_2 with $1 \leq q_1 < q_2 \leq n$ are drawn randomly. The activities for the child in positions $1,\ldots,q_1$ are copied from the father. Child positions $q_1+1,\ldots,q_2$ are filled successively with activities from the mother. We always take the leftmost activity in the mother's list that does not yet appear in the child's current partial list. The remaining child positions $q_2+1,\ldots,n$ are taken from the father accordingly. Note that this crossover preserves the activities' relative positions in the parents and always produces precedence feasible offspring. The mutation operator swaps two activities with probability π_{mutation}, given that the result is still precedence feasible. For more details, refer to Hartmann [6].

This GA has yielded better results than other genetic prepresentations for the standard RCPSP (Hartmann [6]). In the subsequent years, it has been extended by various researchers. The most noteworthy extension was the concept of justification or forward-backward improvement (see Sect. 4.1). As discussed above, however, it is not applicable to instances of the RCPSP/t. Therefore, we stick with the GA as described above.

In what follows, we discuss two ways of applying this GA to the RCPSP/t. The first approach is fairly straightforward. Considering that the serial SGS yields feasible solutions for the RCPSP/t, we can apply the GA described above. The only minor change is that we replace the method to calculate the first generation with an RCPSP/t-specific procedure, namely the tournament method and the critical

path and resource utilization (CPRU) rule of Hartmann [7]. The main drawback of this GA approach is that it might not find an optimal solution because of the serial SGS (recall the discussion in Sect. 4.1).

This issue leads us to the second approach. Here we allow to delay activities, that is, an activity may be started at a time later than the earliest feasible start time. This extends the search space such that an optimal solution can be found.

The idea behind this is somewhat similar to that of Cho and Kim [4] for the standard RCPSP. They proposed a simulated annealing (SA) heuristic in which solutions are represented by priority values and decoded by the parallel SGS. Recall that, like the serial SGS for the RCPSP/t, also the parallel SGS for the RCPSP might exclude all optimal solutions from the search space. To overcome this restriction, Cho and Kim [4] suggest to allow to delay activities. They indicate a delay by a negative priority value (whereas an activity with a positive priority value is started as early as possible). Moreover, they test different rules to control the number of periods an activity may be delayed.

Our approach is similar as it allows to delay activities as well, albeit using the activity list representation and the serial SGS. Handling delays, however, follows a different concept. For each activity j we define a delay value $\delta(j)$ and include it in the representation. Now the activity list of an individual I includes a delay value for each activity:

$$I = \begin{pmatrix} j_1 & j_2 & \cdots & j_n \\ \delta(j_1) & \delta(j_2) & \ldots & \delta(j_n) \end{pmatrix}$$

The serial SGS is applied to determine the schedule for an individual. It schedules the activities in the order prescribed by the activity list. If an activity j_i has a delay value of $\delta(j_i) = 0$, then this activity is scheduled as early as possible. If it has a delay value of $\delta(j_i) = 1$, the earliest possible start time is ignored and the next feasible start time is selected. Higher delay values imply that more feasible start times are skipped. Generally, the earliest $\delta(j_i)$ feasible start times are skipped and the next feasible start time is selected. Note that $\delta(j_i)$ does not indicate the number of periods an activity is delayed. This is necessary because a feasible start time t for some activity does not mean that start time $t+1$ is always feasible if the serial SGS is applied to the RCPSP/t. Also observe that, in contrast to Cho and Kim [4], the magnitude of the delay is part of the representation and needs not be controlled by additional rules.

The GA based on this extended representation proceeds as follows. In the first generation, each individual is assigned an activity list using the tournament procedure and the CPRU rule. With a probability of π_{delay}, an activity is assigned a delay value of 1 and 0 otherwise (higher delay values can be produced by mutation). The crossover operator is extended in a straightforward way: Each activity j_i simply keeps its associated delay value $\delta(j_i)$. The mutation operator is expanded by changing a delay value with probability π_{mutation}. Then it is either increased by 1 or decreased by 1, each with a probability of 0.5 (a negative delay value is not accepted, though). This concept implies that the delay is subject to inheritance—if delaying an activity is beneficial, this will be passed on to the offspring.

Of course, it is not a priori clear whether or not delaying activities is a promising approach. On one hand, it helps to avoid the drawback of the serial SGS which might exclude all optimal solutions from the search space. On the other hand, scheduling activities later than necessary can be counterproductive if the objective is an early end of the project. Thus computational experiments can provide further insight.

5 Computational Results

In order to analyze the behavior of the heuristics and of the extended genetic algorithm, a computational study has been carried out. In what follows, we briefly describe the sets of test instances and present the computational results.

5.1 Test Sets

For the computational analysis we make use of the sets of test instances generated by Hartmann [7]. These sets are based on sets for the standard RCPSP which can be found in the internet-based project

scheduling problem library PSPLIB, cf. Kolisch and Sprecher [15]. The original RCPSP sets were generated by ProGen (see Kolisch et al. [16]) and have been widely accepted as a standard test bed by the RCPSP community.

Hartmann [7] added changes to the resource capacities and requests of these instances to adapt them to the RCPSP/t. These changes are based on parameters. Probabilities P^R and P^r control whether or not a reduction is applied to the capacity and the request in a period, respectively. Higher probabilities imply more frequent changes in the resource availabilities and requests. Factors F^R and F^r determine the strength of the reduction for the availability and the request, respectively. The smaller the factor, the stronger the reduction.

The probabilities were set to 0.05, 0.1, and 0.2. The probabilities for changing the capacities and the requests were always kept the same, that is, $P^R = P^r$. The factors were set to 0 and 0.5. Also the factors for capacities and the requests are the same, that is, $F^R = F^r$. This led to 6 different instances derived from the set with $n = 30$ activities (and hence $6 \cdot 480 = 2880$ instances in total) and 6 different instances derived from the set with $n = 120$ activities (thus $6 \cdot 600 = 3600$ instances in total). More details can be found in Hartmann [7].

5.2 Results

We tested the tournament heuristic of Hartmann [7] with three different priority rules. We included the random rule (RND) as a benchmark, the latest start time rule (LST) which is one of the best performing rules for the standard RCPSP (Kolisch [11]), and the critical path and resource utilization rule (CPRU) which was developed for the RCPSP/t (Hartmann [7]). We also tested the GA of Hartmann [6] in its standard form without delay as well as in the extended version designed for the RCPSP/t. In the latter version, different probabilities π_{delay} for controlling the distribution of delay values in the first generation were examined.

To obtain a basis for the comparison, all tested heuristics were stopped after 5000 schedules were computed for an instance (in the GA, this corresponds to a population size of 100 over 50 generations). Table 1 provides a summary of the results obtained for the two test sets with $n = 30$ and $n = 120$ activities, respectively. It reports the average deviation from the lower bound LB/t (which makes use of the critical path and the time-varying resource parameters, see Hartmann [7] for a definition), the percentage of instances for which a feasible solution is found, and the average computation time per instance in seconds.

The tested methods yield rather similar results for the set with $n = 30$, whereas the set with $n = 120$ shows differences between the heuristics. On the latter set, all methods clearly outperform the random approach. The CPRU rule is slightly better than the LST rule for the standard RCPSP, which confirms the findings of Hartmann [7]. The GAs lead to better results than the priority rule methods because they exploit learning effects during the search. This is in line with the results of Kolisch and Hartmann [13].

		$n = 30$			$n = 120$		
heuristic	configuration	deviation	feasible	CPU-sec	deviation	feasible	CPU-sec
tourn.	RND	11.9%	98.3%	0.101	39.1%	100%	0.471
tourn.	LST	11.3%	98.3%	0.103	31.7%	100%	0.627
tourn.	CPRU	11.3%	98.3%	0.104	31.1%	100%	0.643
GA	no delay	11.3%	98.3%	0.039	29.7%	100%	0.191
GA	$\pi_{\text{delay}} = 0.01$	11.2%	98.5%	0.041	30.3%	100%	0.209
GA	$\pi_{\text{delay}} = 0.05$	11.3%	98.5%	0.041	30.5%	100%	0.209
GA	$\pi_{\text{delay}} = 0.10$	11.3%	98.5%	0.042	30.8%	100%	0.216
GA	$\pi_{\text{delay}} = 0.20$	11.4%	98.5%	0.043	31.5%	100%	0.224

Table 1: Heuristic results, limit = 5000 schedules

For the set with $n = 30$, adding the possibility to delay activities in the GA can have a slight positive effect (the impact is fairly marginal, but it was confirmed in several repetitions of the experiment). For the set with $n = 120$ the GA without delays works best. In case of the much smaller solution space related to the $n = 30$ set, it makes sense to explore a search space that always contains an optimal solution. Considering the huge solution space for the $n = 120$ set, however, it seems to be more promising to reduce the search space to schedules of good average quality. The latter is achieved when activities are not delayed. This is very similar to the findings of Hartmann and Kolisch [9] for the standard RCPSP, where a larger search space (due to the serial SGS) is better for smaller projects whereas a smaller search space (due to the parallel SGS) is more promising for larger projects.

The results also show that higher delay probabilities for setting up the first generation slightly worsens the results. Increasing the delay probability means that too many activities are delayed which leads to inferior solutions. But the results do not deteriorate too much because mutation and selection can eliminate unfavorable delay values from the gene pool. That indicates that the GA is robust because the evolution will discard delays sooner or later if they are not favorable.

The computation times of the priority rule methods are higher than those of the random method because of the time-consuming activity selection method. The lowest computation times are obtained for the GA because it simply picks the next activity from the activity list. Thus, there is no need for a more time-consuming activity selection. Increasing the delay probability leads to slightly higher computation times because in case of a delay the start time calculation of an activity has to be executed more than once. It should also be noted that the methods stop whenever the lower bound LB/t has been reached and thus an optimal solution has been found.

Tables 2 and 3 show the average deviations from the lower bound LB/t for the different instances subsets, that is, for the different probabilities and strengths of the resource capacity and request reduction. Likewise, Tables 4 and 5 display the percentages of those instances for which the lower bound LB/t was met (and hence the solution is proven to be optimal). Here, we restrict ourselves to the most important heuristics.

We observe that the solution gap between upper and lower bound is generally small for the sets with $n = 30$, and for many instances, the solution gap is 0 which means that the lower bound is met. Taking also the sets with $n = 120$ into account, we see that generally the reduction factors of $F^R = F^r = 0$ lead to lower solution gaps and more optimal solutions than factors of $F^R = F^r = 0.5$. Among the sets for $F^R = F^r = 0$, the solution gap is lowest for a high reduction probability ($P^R = P^r = 0.2$). Also note that for the sets with $F^R = F^r = 0$ and $P^R = P^r = 0.2$ there is only a small difference between the random method and the best GA ($n = 120$) or hardly any difference at all ($n = 30$).

A capacity reduction down to 0 means that less degrees of freedom exist for scheduling activities. In such a case, there may be only a few resource feasible start times, especially for activities with a long duration. If such an activity can only start rather late due to the resource capacities, this may determine the makespan to a large degree, and better heuristics cannot find better schedules than simple random methods. Taking these observations into account, future research might focus more on the sets with $F^R = F^r = 0.5$ because this setting leaves more degrees of freedom for scheduling, which helps to identify performance differences between different heuristics.

6 Conclusions

In this contribution, we have summarized the current state of research concerning the RCPSP with time-dependent resource availabilities and requests. We have also adapted a well-known genetic algorithm which was originally proposed for the standard RCPSP. By allowing activities to start later than necessary, we have extended the search space of the genetic algorithm in a way than an optimal solution can be found for the RCPSP/t. The computational experiments showed, however, that delaying activities only leads to better results in case of small project instances.

Future research directions for the RCPSP/t can be promising in two directions. First, further methods tailored for this problem class might lead to improved results. Second, research on real-world applications and case studies can be useful to assess the relevance of the RCPSP/t. It can also be a good idea to

	$F^R = F^r$	0			0.5		
	$P^R = P^r$	0.05	0.1	0.2	0.05	0.1	0.2
tournament	RND	11.7	8.7	5.9	14.7	15.2	14.4
tournament	CPRU	11.2	8.4	5.8	13.9	14.5	13.6
GA	no delay	11.3	8.5	5.9	13.8	14.4	13.6
GA	$\pi_{\text{delay}} = 0.01$	11.2	8.4	5.8	13.9	14.3	13.3

Table 2: Average deviation from lower bound LB/t in % ($n = 30$)

	$F^R = F^r$	0			0.5		
	$P^R = P^r$	0.05	0.1	0.2	0.05	0.1	0.2
tournament	RND	35.6	27.7	15.9	49.8	51.4	54.0
tournament	CPRU	28.3	21.3	12.5	40.5	41.3	43.0
GA	no delay	27.1	20.6	12.3	38.3	39.3	40.8
GA	$\pi_{\text{delay}} = 0.01$	27.5	20.9	12.2	39.2	40.1	41.7

Table 3: Average deviation from lower bound LB/t in % ($n = 120$)

	$F^R = F^r$	0			0.5		
	$P^R = P^r$	0.05	0.1	0.2	0.05	0.1	0.2
tournament	RND	57.9	66.5	69.4	38.5	34.0	34.2
tournament	CPRU	58.3	67.3	69.6	39.0	34.8	35.2
GA	no delay	57.7	67.7	69.0	38.8	34.8	34.8
GA	$\pi_{\text{delay}} = 0.01$	58.1	66.9	69.6	39.0	34.6	34.6

Table 4: Percentage of instances for which lower bound LB/t is met ($n = 30$)

	$F^R = F^r$	0			0.5		
	$P^R = P^r$	0.05	0.1	0.2	0.05	0.1	0.2
tournament	RND	25.3	36.7	59.5	6.7	4.5	5.5
tournament	CPRU	34.3	43.8	63.8	16.3	12.3	9.3
GA	no delay	35.7	44.3	63.2	16.0	12.5	9.8
GA	$\pi_{\text{delay}} = 0.01$	36.0	45.7	65.3	16.3	12.8	10.3

Table 5: Percentage of instances for which lower bound LB/t is met ($n = 120$)

use case studies to identfy promising combinations of the RCPSP/t with other extensions of the RCPSP. The case study concerning aggregated production planning has shown that additional deadlines can be relevant in practice. Further extensions might also be possible, e.g., multiple modes to reflect alternative speeds of the production processes. Multiple modes can also be useful when the selection of orders shall be included, since one mode can reflect the decision not to carry out the production. When order selection is included, also a different objective function such as the maximization of the net present value would be needed.

References

[1] M. Bartusch, R. H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16: 201–240, 1988.

[2] J. Böttcher, A. Drexl, R. Kolisch, and F. Salewski. Project scheduling under partially renewable resource constraints. *Management Science*, 45:543–559, 1999.

[3] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112:3–41, 1999.

[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] B. de Reyck, E. L. Demeulemeester, and W. S. Herroelen. Algorithms for scheduling projects with generalized precedence relations. In Weglarz [22], pages 77–106.

[6] S. Hartmann. A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics*, 45:733–750, 1998.

[7] S. Hartmann. Project scheduling with resource capacities and requests varying with time: A case study. *Flexible Services and Manufacturing Journal*, 25:74–93, 2013.

[8] S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207:1–14, 2010.

[9] S. Hartmann and R. Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127:394–407, 2000.

[10] C. Heimerl and R. Kolisch. Scheduling and staffing multiple projects with a multi-skilled workforce. *OR Spectrum*, 32:343–368, 2010.

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

[12] R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In Weglarz [22], pages 147–178.

[13] R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174:23–37, 2006.

[14] R. Kolisch and K. Meyer. Selection and scheduling of pharmaceutical research projects. In J. Jozefowska and J. Weglarz, editors, *Perspectives in Modern Project Scheduling*, pages 321–344. Springer, Berlin, Germany, 2006.

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

[16] 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.

[17] A. A. B. Pritsker, L. J. Watters, and P. M. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16:93–107, 1969.

[18] A. Sprecher. *Resource-constrained project scheduling: Exact methods for the multi-mode case*. Number 409 in Lecture Notes in Economics and Mathematical Systems. Springer, Berlin, Germany, 1994.

[19] 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.

[20] P. Tormos and A. Lova. A competitive heuristic solution technique for resource-constrained project scheduling. *Annals of Operations Research*, 102:65–81, 2001.

[21] V. Valls, F. Ballestin, and M. S. Quintanilla. Justification and RCPSP: A technique that pays. *European Journal of Operational Research*, 165:375–386, 2005.

[22] J. Weglarz, editor. *Project scheduling: Recent models, algorithms and applications*. Kluwer Academic Publishers, 1999.